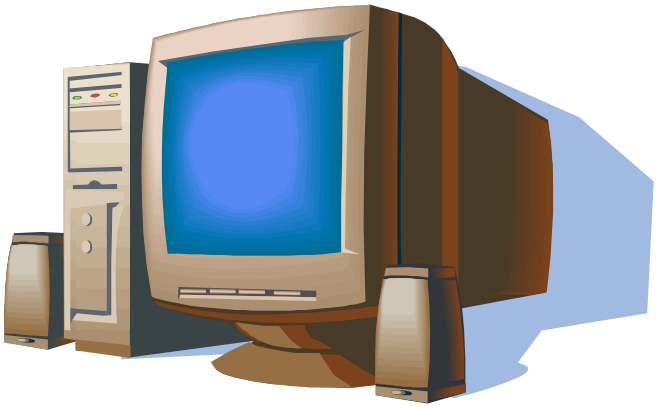
PDF-XChange
Click to buy NOW!
www.docu-track.com

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ГІРНИЧИЙ УНІВЕРСИТЕТ



Основи програмування

Delphi 6.

навчальний посібник

Дніпропетровськ
2013

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ ГІРНИЧИЙ УНІВЕРСИТЕТ

Основи програмування

Delphi 6.

навчальний посібник

Дніпропетровськ
НГУ
2013

PDF-XChange
Click to buy NOW!
www.docu-track.com

УДК 004.438
ББК 32.973.26 - 018.1
О

Затверджено методичними комісіями з напрямів підготовки 6.050101 «Комп'ютерні науки» (протокол № 2 від 31.12.2012) та 6.050103 «Програмна інженерія» за поданням кафедри програмного забезпечення комп'ютерних систем (протокол № 2 від 27.09.2012).

Колектив авторів:
М.О. Алексеєв, д-р техн. наук, проф. (розд. 1.1 – 1.4);
С.П. Кандзюба, канд. техн. наук, доц. (розд. 1.5 – 1.9);
Л.М. Коротенко, канд. техн. наук, доц. (розд. 2.1 – 2.6);
О.С. Шевцова, асистент (розділи 2.7 – 2.11, приклади).

Основи програмування. Delphi 6: навч. посіб. / М.О. Алексеєв,
О С.П. Кандзюба, Л.М. Коротенко, О.С. Шевцова – Д.: Національний гірничий університет», 2013. – 272 с.

У зручній та доступній формі подано основи програмування для операційних систем сімейства Windows з використанням середовища візуального програмування Delphi 6 і мови Object Pascal. Матеріал викладено у двох частинах.

Перша – присвячена вивченню основ мови Object Pascal, яка розглядається на прикладах створення консольних застосувань.

У другій частині наведено технологію створення віконних застосувань – основного виду застосувань в операційних системах сімейства Windows. Головна увага приділяється таким поняттям, як клас, об'єкт та компонент.

Викладення матеріалу супроводжується великою кількістю прикладів та ілюстрацій.

Призначено для студентів напрямів підготовки 6.050101 Комп'ютерні науки і 6.050103 Програмна інженерія вищих навчальних закладів III – IV рівнів акредитації. Буде корисним також для старшокласників, студентів різноманітних спеціальностей, аспірантів та інших користувачів ПК, які цікавляться програмуванням.

УДК 004.438
ББК 32.973.26 - 018.1

Зміст

PDF-XChange
Click to buy NOW!
www.docu-track.com

PDF-XChange
Click to buy NOW!
www.docu-track.com

PDF-XChange
Click to buy NOW!
www.docu-track.com

Передмова

Водночас з масовим розповсюдженням операційних систем сімейства Windows появились і зручні візуальні засоби створення програм, які відразу завоювали велику популярність. Вони суттєво спрощують процес розробки застосувань, дозволяючи розроблювачу зосередитися на розв'язанні конкретної задачі. Нерідко для створення складного застосування досить написати декілька операторів. Процес розробки застосування перетворюється у конструювання зовнішнього вигляду майбутньої програми і встановлення зв'язків між різними її частинами. При цьому знання внутрішньої будови Windows не вимагається.

Все сказане в повній мірі стосується Delphi, яка розвивається дуже динамічно. Перша версія Delphi 1 була випущена у лютому 1995 р. Потім нові версії випускались щорічно.

Задача цього навчального посібника – викладення у зручній та доступній формі основ програмування для операційних систем сімейства Windows з використанням середовища візуального програмування Delphi 6 і мови Object Pascal. Для розв'язання поставленої задачі у посібнику розглядаються всі аспекти, пов'язані з розробкою застосування – від алгоритмізації і мови програмування до використання компонентів і розробки багатовіконих застосувань. Викладення матеріалу супроводжується великою кількістю прикладів та ілюстрацій.

Навчальний посібник складається з двох частин..

Частина 1 «Object Pascal. Створення консольних застосувань» присвячена вивченню основ мови Object Pascal. Мова Object Pascal розглядається на прикладах створення консольних застосувань. Цей прийом, безумовно, буде корисним і початкуючим програмістам, і тим, хто переходить до Delphi із середовища Turbo Pascal.

У частині 2 «Object Pascal. Створення віконних застосувань» розглядається технологія створення віконних застосувань – основного виду застосувань, що використовуються в операційних системах сімейства Windows. У другій частині продовжено вивчення Object Pascal і головна увага приділяється таким поняттям, як клас, об'єкт та компонент.

Навчальний посібник призначено для студентів технологічних, механічних та економічних спеціальностей вищих навчальних закладів III – IV рівня акредитації. Посібник буде корисним також для старшокласників, студентів комп'ютерних спеціальностей, аспірантів та інших користувачів ПК, що цікавляться програмуванням.

Частина 1. Object Pascal. Створення консольних застосувань.

Delphi – це середовище розробки програм, орієнтованих на роботу в операційних системах сімейства Windows. Програми в Delphi створюються на основі сучасної технології візуального проектування, що, у свою чергу, базується на ідеях об'єктно-орієнтованого програмування. Програми в Delphi пишуться мовою Object Pascal, що є спадкоємицею і розвитком мови Turbo Pascal . Мова програмування Turbo Pascal, а також однойменне інтегроване середовище розробки, у якому вона використовувалася, у недавньому минулому завоювали широку популярність як засіб розробки програмних продуктів і особливо як засіб навчання програмуванню. Ця популярність була обумовлена простотою мови, високоякісним компілятором і зручним середовищем розробки. Але програмістські технології не стоять на місці, і фірма Borland (із квітня 1998 року – Inprise Corporation) робить чергове зусилля: на зміну мові Turbo Pascal приходить Object Pascal, що втілив у собі концепцію об'єктно-орієнтованого програмування.

Delphi і Object Pascal є результатами тривалої еволюції і в даний момент – це продукти, у яких відбиті найсучасніші комп'ютерні технології. Зокрема, це означає, що за допомогою Delphi можна створювати самі різні типи програм – починаючи від консольних застосувань і закінчуючи застосуваннями для роботи з базами даних і Internet. У цьому зв'язку виникає закономірне питання: з чого почати вивчення Delphi? На наш погляд, відповідь може бути наступною: знайомство з Delphi варто почати з Object Pascal і використання цієї мови програмування для створення консольних застосувань.

1.1. Основні поняття

1.1.1. Поняття алгоритму і програми

Алгоритм – це послідовність дій (правил), що однозначно приводять до рішення поставленої задачі.

Рішення будь-якої задачі, зокрема складання програми, припускає насамперед розробку алгоритму. Саме на розробку алгоритму рішення задачі іде левина частка зусиль при складанні програм. Але зате, якщо є в розпорядженні алгоритм, для кваліфікованого програміста не складає великої праці записати цей алгоритм алгоритмічною мовою, тобто скласти програму.

З іншого боку, програміст, розробляючи алгоритм, повинний враховувати особливості середовища програмування, у якому буде створюватися програма, свій особистий досвід, раніше розроблені алгоритми і програми, що є в розпорядженні. Це означає, що якщо для рішення задачі існують різні способи, тобто можуть бути складені різні алгоритми, то програміст вибере найбільш придатний для нього.

Таким чином, можна затверджувати, що процес програмування нерозривно зв'язаний з розробкою алгоритму. Зокрема, це знаходить своє вираження в тім, що слово «алгоритмічний» часто входить у назву мови програмування. Хоча варто враховувати і те, що програмістські технології розвиваються стрімко, і зараз у назви мов програмування частіше додаються слова «візуальний», «об'єктно-орієнтованый» і т.д.

Розглянемо приклад складання алгоритму. Нехай потрібно обчислити:

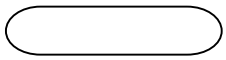
$$z = \frac{x^2 + \sin(2x - 1) + 0{,}5}{\sqrt{x^2 + y^2 + 10^4}} \qquad (1.1)$$

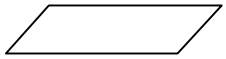
Існують різні способи запису алгоритмів. Найбільш простий з них – словесний опис. Для розглянутого приклада словесний опис може бути таким:

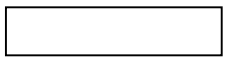
1. Ввести в оперативну пам'ять комп'ютера значення змінних *x* і *y*.
2. Обчислити значення змінної *z* по заданій формулі.
3. Вивести значення змінної *z* на екран дисплея.

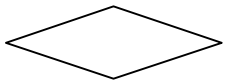
Приведений спосіб опису алгоритму є найпростішим, але не таким наочним, як опис алгоритму за допомогою блок-схем.

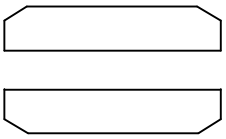
Блок-схема являє собою графічний спосіб опису алгоритму. Кожен блок, що входить у блок-схему, означає яку-небудь дію. У блок-схемах можуть використовуватися блоки різних типів, у залежності від характеру виконуваних дій:


– початок або кінець блок-схеми;


– введення або виведення даних;

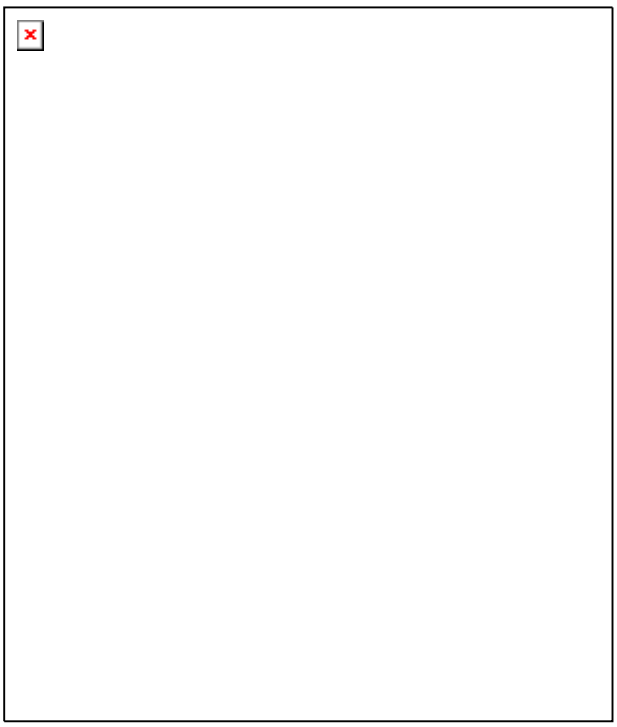

– блок модифікації;


– розвилка;


– блоки для виконання циклу задане число раз.

Мал. 1.1. Типи блоків, використовуваних для складання блок-схем.

З використанням блок-схеми алгоритм обчислення по формулі (1.1) може бути представлений так:



Мал. 1.2. Блок-схема алгоритму обчислення по формулі (1.1).

Всі алгоритми можуть бути розбиті на найпростіші алгоритми, що, у свою чергу, належать до одного з трьох типів:

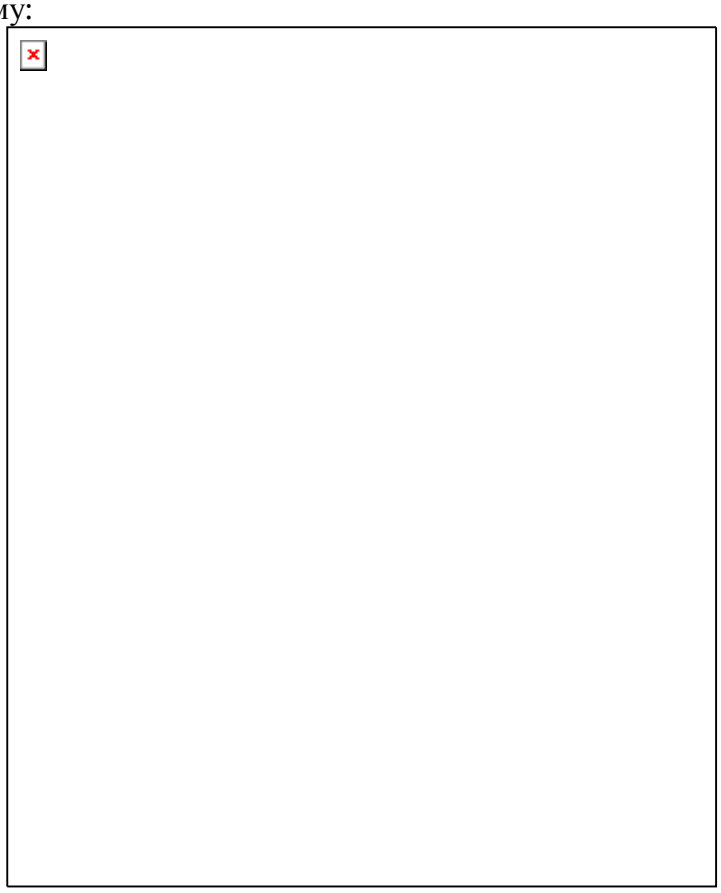
1. Лінійний обчислювальний процес.
2. Розгалужний обчислювальний процес.

3. Циклічний обчислювальний процес.

Лінійний обчислювальний процес – найпростіший з обчислювальних процесів – являє собою послідовність однократно виконуваних дій. Розглянутий вище приклад обчислення по формулі (1.1) являє приклад лінійного обчислювального процесу.

В розгалужному обчислювальному процесі перевіряється деяка умова й у залежності від того, виконується вона чи ні, обчислення йдуть по однієї або іншій галузі.

Нехай, наприклад, потрібно знайти максимальне з двох чисел – *max*(*a*,*b*). Приведемо блок-схему алгоритму:

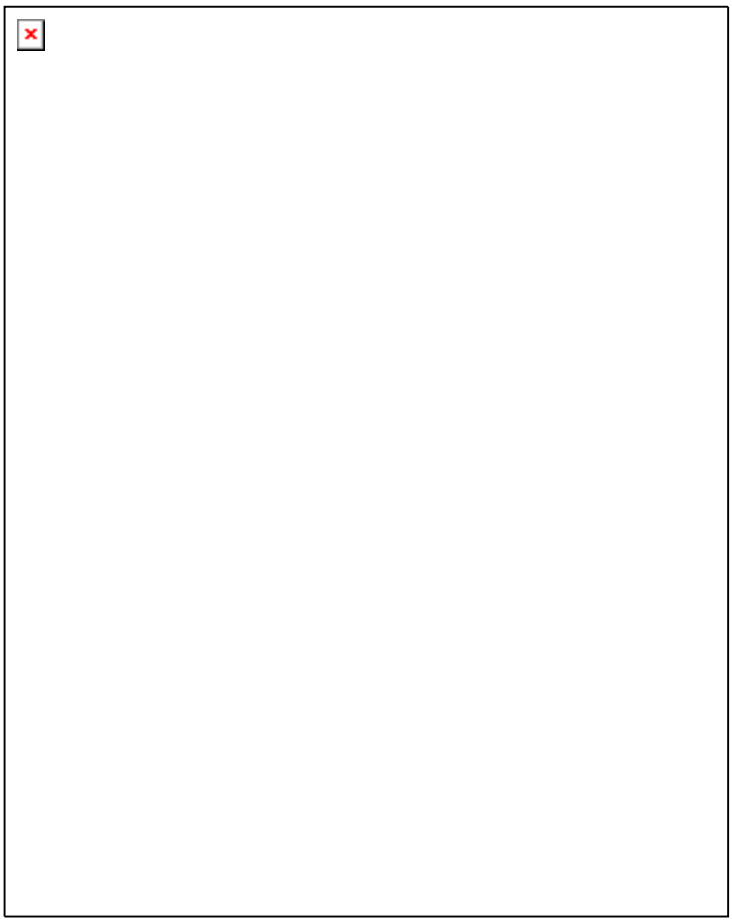


Мал. 1.3. Блок-схема алгоритму обчислення *max*(*a*,*b*).

Циклічним є обчислювальний процес, що містить групу багаторазово повторюваних операторів, тобто цикл. Як приклад циклічного обчислювального процесу розглянемо знаходження суми:

$$S = \sum_{i=1}^{10} \frac{1}{i^2} = 1 + \frac{1}{4} + \frac{1}{9} + \ldots + \frac{1}{100}.$$

Блок-схема алгоритму:



Мал. 1.4. Блок-схема алгоритму обчислення суми.

У недосвідченого читача може викликати здивування рядок $S := S+1/i^2$ у блоці модифікації. Зміст виконуваної в циклі дії полягає в наступному.

Обчислення суми може бути представлене так:

$S := 0;$
для $i = 1$ $S := S+1;$
для $i = 2$ $S := S+1/4;$
для $i = 3$ $S := S+1/9;$
…….....
для $i = 10$ $S := S+1/100,$

тобто для кожного i береться значення змінної S, обчислене для попереднього $(i - 1)$-го кроку, і додається черговий доданок $1/i^2$. Потім отриманий результат поміщується в змінну S замість старого значення. Знак «:=» означає не рівність, як це прийнято в математиці, а присвоювання змінної S значення виразу, що стоїть праворуч від знака «:=».

Виявляється, що, яким би складним ні був алгоритм рішення задачі, він може бути розбитий на найпростіші алгоритми, кожний з який належить до одного з трьох вище перерахованих типів обчислювальних процесів. Очевидно, вірним буде і зворотне. Процес розробки алгоритму подібний конструюванню: із простих деталей збираються складні конструкції.

Після розробки алгоритму рішення задачі можна перейти до написання програми.

Програма – це набір інструкцій (операторів), записаних алгоритмічною мовою і реалізуючих заданий алгоритм.

Нашою найближчою задачею є розгляд основних елементів мови Object Pascal, що можуть бути використані при створенні консольних застосувань.

1.1.2. Просте консольне застосування

Програми, що працюють в операційних системах сімейства Windows, називають **застосуваннями**.

Delphi дозволяє програмісту створювати застосування, у яких для введення даних в оперативну пам'ять із клавіатури використовуються процедури read і readln, а для виведення результатів на екран монітора – процедури write і writeln. Такі застосування називаються **консольними**.

Розглянемо приклад.

Приклад 1.1.

Скласти програму для обчислення значення змінної z по формулі (1.1).

Рішення.

Для складання програми скористаємося блок-схемою, приведеної на мал. 1.2.

Програма.

```
program p1_1;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var x,y,z:real;
begin
  writeln('Enter x,y');
  readln(x,y);
  z := (sqr(x)+sin(2*x+1)+0.5)/
       sqrt(x*x+y*y+1e4);
  writeln('z=',z:7:4);
  readln
end.
```

Результати роботи програми.

Після виконання програми на екрані монітора буде висвітлено:

```
Enter x,y
2.3 -5.41
z= 0.0515
```

Програма починається з **заголовка**. У заголовку вказується зарезервоване слово program і ім'я програми – p1_1. **Зарезервовані слова** можуть бути використані в програмі тільки по своєму прямому призначенню, і їх не можна використовувати в іншому контексті, наприклад, як імена змінних. Зарезервованими словами в нашому прикладі є також слова var, begin, end. **Ім'я програми** – p1_1 – задається програмістом за своїм розсудом.

У фігурних дужках міститься **директива компілятора**, що вказує на те, що створюється консольне застосування. **Компілятор** – це програма, що входить до складу середовища Delphi і призначена для перекладу операторів, написаних мовою Object Pascal на мову машинних кодів, зрозумілих процесору. Це означає, що за допомогою середовища Delphi можна створювати програмні файли (файли з розширенням exe), що надалі можуть бути запущені без використання середовища Delphi.

Інструкція uses SysUtils; означає, що наша програма в процесі своєї роботи може використовувати дані і підпрограми, що знаходяться в **стандартному модулі** SysUtils. Стандартні модулі створені програмістами фірми Inprise і призначені для спрощення процесу розробки застосування.

Зарезервоване слово var означає початок розділу опису змінних. **Змінними** називають величини, що у ході виконання програми можуть змінювати своє значення. У нашому прикладі – це *x*, *y* і *z*. У Object Pascal необхідно строго дотримувати правила: усі змінні, використовувані в програмі, повинні бути описані в розділі опису змінних із указівкою їхніх

типів. Для кожної змінної виділяється область в оперативній пам'яті комп'ютера, розмір якої визначається типом змінної і виміряється в байтах.

Любий **тип даних** визначає множину значень, що приймає змінна. Так, у нашій програмі змінні мають тип real, це означає, що вони можуть приймати практично будь-які дійсні значення: 23.59, -0.001, 1.0, 3.1414926 і т.д.

За розділом опису змінних йде розділ операторів. Він узятий у зарезервовані слова begin і end. Після слова end стоїть крапка, що позначає кінець програми.

У розділі операторів відбувається наступне.

У результаті виконання процедури writeln на екрані з'являється повідомлення:

Enter x,y (тобто – Уведіть x,y).

У програмі це повідомлення узяте в апострофи і являє собою символьний рядок.

Символьним рядком чи просто **рядком** називається послідовність символів, узята в апострофи (одиночні лапки).

Далі в програмі виконується процедура readln, що приводить до припинення виконання програми і чеканню введення з клавіатури значень змінних x і y. Після того як числові значення введені з клавіатури і натиснута клавіша Enter, виконуються наступні оператори програми.

У Object Pascal є декілька операторів, за допомогою яких можуть бути реалізовані усі вище описані типи обчислювальних процесів. Символом := позначається **оператор присвоювання**. Дія оператора присвоювання полягає в тому, що обчислюється значення виразу, що стоїть праворуч від оператора, і це значення присвоюється змінної, що стоїть ліворуч від оператора присвоювання.

Вирази можуть складатися з констант, змінних, функцій, з'єднаних знаками відповідних операцій. Порядок дій у виразі можна змінювати, використовуючи круглі дужки.

Вирази можна розрізняти по типі їхнього результату: цілі, дійсні, символьні, логічні, рядкові. У нашому прикладі, очевидно, у результаті обчислення виразу вийде дійсне значення. Тип змінної z вибирався з розуміння відповідності типу виразу.

Вираз складається з операцій і операндів. У прикладі 1.1 використовуються арифметичні операції + , - , * , / . **Операції** визначають дії, що проваджуються над однією (унарні операції) чи двома (бінарні операції) величинами, іменованими **операндами**. Наприклад, у виразі

sqr(x)+sin(2*x-1)+0.5

для першої операції «+» операндами є sqr(x) і sin(2*x-1), а для другої операції «+» операндами служать результат, отриманий після першого додавання, і 0.5.

Числа 0.5 і 1e4 (тобто 10^4), що фігурують у виразі, – це константи. **Константами** називають будь-які незмінні в процесі виконання програми дані. Якщо та сама константа використовується в програмі багаторазово, їй можна дати ім'я і використовувати в програмі ім'я константи замість указівки самої константи.

Слова sqr, sin, sqrt є іменами **стандартних функцій**, тобто функцій, які можна використовувати, не описуючи їх попередньо у своїй програмі. Розроблювачі Delphi створили велику кількість найрізноманітніших стандартних функцій. Описи стандартних функцій і процедур (загальна назва – підпрограми) знаходяться в численних стандартних модулях. У прикладі 1.1 використовується один з таких стандартних модулів – SysUtils.

Після оператора присвоювання виконується процедура writeln, що виводить результати обчислень на екран. Після імені змінної z через двокрапку стоять числа 7 і 4, які означають, що при виведенні даних на екран монітора для значення змінної z виділяється 7 позицій, з них 4 – для дробової частини.

Останньою в розділі операторів виконується процедура readln без параметрів. Це викликає припинення виконання програми, у результаті чого ми зможемо побачити на екрані результати роботи програми. Для завершення роботи програми треба натиснути клавішу Enter.

1.1.3. Алфавіт мови

Алфавіт – це сукупність припустимих у мові символів чи груп символів, розглянутих як єдине ціле. Алфавіт мови Object Pascal складається з букв, цифр, спеціальних символів і невикористовуваних символів.

До **букв** відносяться великі і малі букви латинського алфавіту – від A до Z і від a до z. При цьому Object Pascal не розрізняє однойменні великі і малі букви, якщо тільки вони не входять у символьний чи рядковий вираз. Буквою є також знак підкреслення «_» .

До **цифр** відносяться арабські цифри від 0 до 9 і шістнадцяткові цифри. Використовуючи шістнадцяткові цифри можна записати шістнадцяткове число. Щоб відрізнити десяткове число від шістнадцяткового, перед останнім ставлять знак долара $. Діапазон шістнадцяткових чисел – від $00000000 до $FFFFFFFF.

Спеціальні символи можна умовно розділити на роздільники, знаки пунктуації, знаки операцій і зарезервовані слова.

Роздільники використовуються для відділення друг від друга ідентифікаторів, чисел і т.д. Як роздільники можна використовувати:

- пробіл;
- будь-який керуючий символ (будь-які символи в діапазоні кодів від 0 до 32);
- коментар.

Декілька підряд розташованих пробілів вважаються одним пробілом. Керуючі символи, до числа яких відноситься, наприклад символ повернення каретки, генеруємий натисканням клавіші Enter, Object Pascal також інтерпретує як пробіли.

Коментарями називається будь-яка послідовність символів, узята у фігурні дужки:

{ це коментар } .

Замість фігурних дужок можна використовувати пари символів (* і *) . Коментар у фігурних дужках може займати будь-як число рядків і під час виконання програми ігнорується. Основне призначення коментарю – пояснення по тексту програми.

Коментарем є також будь-яка послідовність символів, що стоїть після пари символів // і до кінця рядка, наприклад:

```
var x,y:real;     // розділ опису змінних
```

Не слід плутати коментар у фігурних дужках і директиви компілятора. Директиви компілятора починаються з пари символів {$, що розглядаються як одне ціле.

До знаків пунктуації відносяться:

( ) (* *) [ ] (. .) { } ' , . : ; // := .. ^ @ $ # .

Знаками операцій є:

+ - * / = <> <= >= .

Зарезервованими в Object Pascal є наступні слова:

and	**downto**	**in**	**or**	**string**
array	**else**	**inherited**	**out**	**then**
as	**end**	**initialization**	**packed**	**threadvar**
asm	**except**	**inline**	**procedure**	**to**
begin	**exports**	**interface**	**program**	**try**
case	**file**	**is**	**property**	**type**
class	**finalization**	**label**	**raise**	**unit**
const	**finally**	**library**	**record**	**until**
constructor	**for**	**mod**	**repeat**	**uses**
destructor	**function**	**nil**	**resourcestring**	**var**
dispinterface	**goto**	**not**	**set**	**while**
div	**if**	**object**	**shl**	**with**
do	**implementation**	**of**	**shr**	**xor**

Приведені слова використовуються в програмі тільки по своєму прямому призначенню, і не можуть бути використані як ідентифікатори.

Невикористовувані символи, до яких, зокрема, відносяться букви російського алфавіту, можуть бути використані в коментарях, а також символьних і рядкових константах.

1.1.4. Ідентифікатори

Ідентифікатори (імена) використовуються для позначення констант, типів, змінних, процедур, функцій, класів, модулів, програм і т.д. Ідентифікатори складаються з букв і цифр і повинні починатися з букви. Довжина ідентифікатора може бути довільною, але значущими є тільки перші 64 символи. Оскільки знак підкреслення «_» відноситься до букв, він може входити в ідентифікатор. Спеціальні символи і невикористовувані символи не можуть входити до складу ідентифікаторів. Зокрема, зарезервовані слова (program, begin, end і т.д.) не можна використовувати як ідентифікатори.

Приклади припустимих імен:

x y sum p1_13_5 myfirstprogram square_of_rectangle .

Неприпустимими іменами є:

5th	–	починається з цифри;
type	–	зарезервоване слово;
корінь	–	містить українські букви;
sum.6	–	містить спеціальний символ;
y-3	–	те ж;
x 5	–	містить пробіл.

Крім зарезервованих слів у Object Pascal є так називані стандартні імена. До них відносяться імена стандартних функцій, наприклад: sin, cos, exp, ln і т.д. Стандартними іменами є також імена стандартних директив, що використовуються для оголошень у програмі, наприклад: override, virtual, private, protected (див. частина 2) і т.д. На відміну від зарезервованих слів стандартне ім'я можна перевизначити, щоправда, при деяких обмеженнях. Можливість ця скоріше теоретична, і на практиці рекомендується використовувати свої оригінальні імена.

1.1.5. Структура програми

Консольне застосування мовою Object Pascal складається з заголовка і так називаного блоку. Блок складається з розділів. Перелічимо їх:

1. Розділ міток.
2. Розділ констант.
3. Розділ типів.
4. Розділ змінних.
5. Розділ процедур і функцій.
6. Розділ операторів.

Розділ операторів повинний бути узятий в операторні дужки begin...end. У ньому вказується послідовність дій, що повинний виконати комп'ютер. Всі інші розділи носять описовий характер.

У Object Pascal будь-який розділ, крім останнього, може бути відсутнім. І навпаки – будь-який описовий розділ може зустрічатися кілька разів, при цьому порядок слідування розділів може бути довільним. Вибираючи порядок розташування описових розділів, варто керуватися правилом: усі використовувані типи, константи, змінні, функції, процедури повинні бути оголошені чи описані до їхнього першого використання.

Роздільником між розділами й операторами служить крапка з комою. Наприкінці застосування повинна стояти крапка.

У будь-якому місці застосування, де допускається присутність роздільника, наприклад – пробілу, можна вставити коментар. Коментар не змінює змісту застосування і призначений для пояснення

Розглянемо структуру консольного застосування на прикладі.

Приклад 1.2.

Скласти програму, у якій із заданого числа x віднімається величина 2π доти, поки x не потрапить у діапазон $0 \le x \le 2\pi$.

Рішення.

Наша навчальна програма буде містити всі припустимі в консольному застосуванні розділи, хоча на практиці текст програми може бути набагато простіше.

Програма.

```
program P1_2;
{$APPTYPE CONSOLE}
uses
  SysUtils;

label lb,33;
const p=3.1415926;
type answer = (No,Yes);
var x:real;
function prov(v:real):answer;
  var z:answer;
  begin
    if (v>=0) and (x<=2*p) then z := Yes
                           else z := No;
    result := z
  end;
procedure vych(var w:real);
  begin
    w := w-2*p
  end;

begin
  writeln('Enter x');
  readln(x);
lb:vych(x);
  if prov(x) = Yes then goto 33
                   else goto lb;
33:writeln('x=',x:6:4);
  readln
end.
```

Результати роботи програми.

```
Enter x
100
x=5.7522
```

Заголовок програми починається зі службового слова program, після якого йде ім'я програми – правильний ідентифікатор.

Розділ міток починається словом **label**, за яким розташований список міток. Мітка являє собою правильний ідентифікатор чи ціле число без знака. Мітки дозволяють позначити будь-який оператор, щоб можна було передати керування з будь-якого місця програми.

Розділ констант починається словом const, за яким йдуть конструкції виду

< ім'я константи > = < значення > ,

що дозволяють привласнити константі ім'я і використовувати його в тексті програми.

Розділ опису типів починається службовим словом type, за яким йдуть конструкції виду

< ім'я типу > = < опис > ,

що дозволяють програмісту створювати власні типи.

Розділ опису змінних починається зі службового слова var. Тут повинні бути вказані всі змінні, використовувані в розділі операторів програми, а також їхній тип. У нашому прикладі використовується одна змінна x типу real.

Розділ опису процедур і функцій не виділяється спеціальним службовим словом, оскільки кожна підпрограма має свій заголовок. У нашому прикладі використовуються дві підпрограми: функція prov і процедура vych. Кожна з них може мати структуру, подібну до основної програми, у тому числі використовувати власні, т.зв. локальні змінні. У функції prov – це змінна z типу answer.

Розділ операторів починається службовим словом begin і закінчується словом end, після якого ставиться крапка – ознака кінця програми. Між цими ключовими словами розміщаються оператори мови Object Pascal, які необхідно виконати для рішення задачі.

1.1.6. Типи даних

Типи даних у Object Pascal можна розділити на стандартні, тобто визначені в мові, і визначені програмістом. До стандартних типів відносяться наступні: цілі, дійсні, символьні, рядкові, вказівники, логічні і variant. Програміст може визначати типи безпосередньо при описі змінної в розділі var, або, найчастіше, у спеціально призначеному для цього розділі опису типів – type. Після зарезервованого слова type можуть йти оголошення створюваних програмістом типів, що у загальному виді мають вид:

< ім'я типу > = < опис типу>;

Наприклад, у застосування можна помістити наступні оголошення:

```
type TColor = (Red, Blue, Black);
var color1, color2:TColor;
```

У розділі type задається ім'я нового типу – TColor і вказуються припустимі для даного типу значення - Red, Blue і Black. А в розділі var описуються дві змінні – color1 і color2, що мають визначений програмістом тип.

Описати змінні color1 і color2 можна і безпосередньо в розділі var:

```
var color1, color2:(Red, Blue, Black);
```

Це оголошення еквівалентне двом попереднім.

Стандартні типи не потрібно описувати в розділі type, а можна відразу використовувати при описі змінних у розділі var.

Наявні в Object Pascal стандартні типи можна класифікувати в такий спосіб:

- Прості
  - Порядкові
    - Цілі
    - Символи
    - Логічні
    - Перелічувані
    - Обмежені
  - Дійсні
- Рядки
- Структури
  - Множини
  - Масиви
  - Записи
  - Файли
  - Класи

- Інтерфейси

❖ Вказівники
❖ Процедурні
❖ Variant

Мал. 1.5. Класифікація типів Object Pascal.

1.1.7. Константи

У мові Object Pascal існують константи двох видів: звичайні й іменовані.

Звичайна константа – це число, символ, рядок чи логічне значення. Числові константи можуть бути цілими чи дробовими, позитивними чи негативними. У дробових константах ціла частина від дробової відокремлюється крапкою. Перед негативним числом ставиться знак «мінус», наприклад:

325 0.0 -627.15 0 .

Перед позитивним числом знак «плюс» можна не ставити.

Дробові константи можуть бути записані у виді числа з рухомою крапкою, тобто у виді

±ae±n ,

де a – число, за модулем менше 10;
e – спеціальний символ;
n – порядок числа;
± – знак «мінус» або «плюс» перед a і n .

Наприклад:

Вихідне число	Перетворене число	Константа з рухомою крапкою
1000	1×10^{4}	1.00000000000000e+0004
-627,15	$-6,2715\times10^{2}$	-6.2715000000000e+0002
0.000217	$2,17\times10^{-4}$	2.17000000000000e-0004

Таку форму констант із рухомою крапкою можна побачити на екрані дисплея при виведенні результатів роботи програми, якщо не використовується форматне виведення (див. п. 1.1.12.2.). Безпосередньо в програмі константи з рухомою крапкою можуть бути записані коротше, наприклад:

1e4 -6.2715e2 2.17e-4 .

Рядкові і символьні константи беруться в одинарні лапки:

'Середовище візуального програмування Delphi5'

'Іванов И.И.' '5.62' 'A' 'a' 'ъ' .

Можна вважати, що символ – це рядок одиничної довжини. Символ може бути також записаний за допомогою указання його внутрішнього коду, який можна визначити по таблиці кодів ANSI, якщо ви працюєте в операційній системі сімейства Windows. Перед кодом символу ставиться знак # .Так, наприклад, з огляду на те, що в таблиці кодів ANSI усі великі латинські букви розташовуються підряд і код букви A дорівнює 65, символьний рядок

'ABCDEFGH'

еквівалентний наступному

#65#66#67#68#69#70#71#72

Логічних констант дві: true (істина) і false (неправда).

Іменована константа відрізняється від звичайної тем, що в неї є ім'я. Тому замість указання в програмі значення константи можна використовувати її ім'я. Це зручно, наприклад, у тому випадку, коли константа використовується багаторазово чи коли константа має велику довжину. Оголошення іменованої константи повинне бути поміщене в розділ опису констант і в загальному виді виглядає в такий спосіб:

< ім'я константи > = < значення >;

наприклад:

const

PDF-XChange Click to buy NOW! www.docu-track.com

```
Price = 250;
Name = 'Петров П.М.';
radius = 2.514;
```

Іменована константа може бути визначена за допомогою так називаного константного виразу:

< ім'я константи > = < константний вираз >;

У константному виразі можуть бути використані звичайні константи, раніше описані іменовані константи, знаки операцій, а також деякі стандартні функції:

Abs	*Lo*	*Round*
Chr	*Low*	*SizeOf*
Hi	*Odd*	*Succ*
High	*Ord*	*Swap*
Length	*Pred*	*Trunc* .

Наприклад:

```
const
    Nalog = 300*0.2;
    Name = 'Олексій'+'Петрович';
    Gamma = abs(1.18*1.5);
```

Обчислюються значення константних виразів на етапі компіляції, і на етапі виконання програми використовуються обчислені значення іменованих констант, що прискорює роботу програми.

Використання іменованих констант робить текст програми більш осмисленим і полегшує при необхідності зміну значення константи у всьому тексті.

Як ми вже знаємо, константи – це дані програми, що не можуть змінювати своє значення під час виконання програми. Але в цьому правилі є виключення – типізовані константи.

Типізована константа визначається в такий спосіб:

< константа >:< тип > = < константний вираз >;

Наприклад:

```
const
    A:integer = 6;
    S:real = 18.263-trunc(18.263);
```

Значення типізованих констант можна змінювати під час виконання програми, але за умови, що була виконана директива компілятора {$J+}. Якщо була виконана директива {$J-}, то змінювати значення типізованих констант не можна і вони перетворюються в звичайні іменовані константи. За замовчуванням діє директива {$J+}.

1.1.8. Змінні

Змінні – це дані програми, що у процесі її виконання можуть змінювати своє значення. Як значення змінні можуть містити дані будь-яких типів. Кожній змінній в оперативній пам'яті комп'ютера виділяється область, розмір якої залежить від типу змінної. Зміна значення змінної означає зміну даних, розташованих у відповідній області оперативної пам'яті.

Кожна змінна, що з'являється в тексті програми, повинна бути попередньо описана в розділі опису змінних. В описі вказується ім'я змінної і її тип:

< ім'я змінної > : < тип >;

Як ім'я може бути використаний будь-який правильний ідентифікатор, тобто послідовність з букв латинського алфавіту, цифр і знака підкреслення, що починається з букви. Оскільки компілятор мови Object Pascal не розрізняє великі і маленькі букви, то, наприклад, ідентифікатори ALFA, Alfa і alfa означають ту саму змінну.

Розділ опису змінних починається зарезервованим словом var, наприклад:

```
var
```

```
        x:real;
        i:integer;
        c:char;
        h:boolean;
```

Якщо в програмі є декілька змінних однакового типу, то при описі їх можна об'єднати в групу, перелічивши їх через кому і вказавши тип після останньої з них, наприклад:

```
var
        summa,x,y:real;
        i,j,k:integer;
        st1,st2:string;
```

Змінні можна розділити на локальні і глобальні. Докладніше ми будемо це обговорювати в главі 1.4.4., а поки відзначимо наступне.

Змінні, оголошені в процедурах і функціях, є локальними. Вони існують тільки під час виконання відповідних процедур і функцій. Змінні, оголошені в розділі var основної програми, є глобальними. Глобальній змінній можна присвоювати початкове значення при її описі в розділі var чи, іншими словами, ініціалізувати її, наприклад:

```
var
        i:integer=23;
        sum:double=0.11*length('0123456789');
```

Локальні змінні ініціалізувати не можна.

1.1.9. Операції й операнды. Вирази

У мові Object Pascal визначені наступні операції:

@, **not**, ^, *****, /, **div**, **mod**, **and**, **shl**, **shr**, **as**, +, –, **or**, **xor**, =, >, <, <>, <=, >=, **in**, **is**.

Операції обумовлюють дії, які треба виконати над даними. Операції застосовуються до операндів. Наприклад, у виразі

x+y

змінні x і y є операндами, а «+» - операція. Операція називається бінарною, якщо вона застосовується до двох операндів, і унарною, – якщо до одного. Будь-яка операція визначена тільки для відповідних типів операндов, наприклад, операції ділення і множення визначені тільки для операндів цілого і дійсних типів, а логічні операції and і or – тільки для операндів логічного типу.

Як приклад приведемо опис арифметичних операцій, тобто операцій, застосовуваних до даних цілого і дійсного типів:

Бінарні арифметичні операції

Операція	Назва	Тип операндів	Тип результату
+	додавання	цілі	цілий
		хоча б один – дійсний	дійсний
–	віднімання	те ж	те ж
*	множення	те ж	те ж
/	дійсне ділення	цілі чи дійсні	дійсний
div	цілочислене ділення	цілі	цілий
mod	залишок від ділення цілих чисел	цілі	цілий

Унарні арифметичні операції

Операція	Назва	Тип операнда	Тип результату
+	тотожність знака	цілий чи дійсний	збігається з типом операнда
-	заперечення знака	те ж	те ж

Зміст операцій div і mod можна зрозуміти з наступних прикладів:

Вираз	Результат
8/4	2.0
8 mod 4	0
8 div 4	2
6/4	1.5
6 mod 4	2
6 div 4	1

Інші операції будуть розглянуті в главах, присвячених відповідним типам даних.

Вирази складаються з операцій і операндов. У найпростішому випадку вираз може містити одну бінарну чи унарную операцію. У бінарних операціях використовується звичайне алгебраїчне представлення, наприклад: a+b. В унарных операціях операція завжди передує операнду, наприклад: -b.

У більш складних виразах порядок, у якому виконуються операції, відповідає пріоритету операцій (див. Таблицю 1.1).

Порядок виконання операцій. Таблиця 1.1.

Операції	Пріоритет
@, not	перший (вищий)
*, /, div, mod, and, shl, shr, as	другий
+, -, or, xor	третій
=, <>, <, >, <=, >=, in, is	четвертий (нижчий)

Для визначення порядку виконання операцій у виразі варто користатися наступними правилами:

- операнд, що знаходиться між двома операціями з різними пріоритетами, зв'язується з операцією, що має більш високий пріоритет;
- операнд, що знаходиться між двома операціями з рівними пріоритетами, зв'язується з тією операцією, що знаходиться ліворуч від нього;
- вираз, узятий в дужки, перед виконанням обчислюється, як окремий операнд.

Приведені правила дозволяють зробити важливий практичний висновок: керувати порядком виконання операцій можна за допомогою круглих дужок.

Помітимо також, що операції з рівним пріоритетом звичайно виконуються зліва на право, хоча іноді компілятор при генерації оптимального коду може переупорядкувати операнди, але це не впливає на кінцеве значення виразу.

Вирази розрізняють по типі їхнього результату: цілі, дійсні, символьні, логічні, рядкові і т.д. Вирази, що мають результат цілого і дійсного типу, називаються арифметичними.

Приклади виразів різних типів будуть приведені при розгляді відповідних типів.

1.1.10. Стандартні функції і процедури

У Object Pascal існує великий набір так званих стандартних функцій і процедур, тобто підпрограм, створених розроблювачами Delphi, їх можна безпосередньо використовувати в

програмі без попереднього опису. Стандартні процедури і функції умовно можна розбити на наступні категорії:

- математичні;
- перетворення типів;
- обробки рядків;
- обробки числових масивів;
- доступу і керування файлами;
- інші процедури і функції.

Як приклад приведемо наявні в Object Pascal математичні стандартні функції:

Математичні стандартні функції. Таблиця 1.2.

Стандартна функція	Назва	Тип аргументу	Тип результату
Abs(X)	абсолютне значення	цілий чи дійсний вираз	збігається з типом аргументу
ArcTan(X)	арктангенс	те ж	дійсний
Cos(X)	косинус	те ж	те ж
Exp(X)	експонента	те ж	те ж
Frac(X)	дробова частина аргументу: X-Int(X)	те ж	те ж
Int(X)	ціла частина аргументу	те ж	цілий
Ln(X)	натуральний логарифм	те ж	дійсний
Pi	число : 3.14159265358979323 85	аргумент відсутній	те ж
Round(X)	найближче ціле аргументу	цілий чи дійсний вираз	цілий
Sin(X)	синус	те ж	дійсний
Sqr(X)	квадрат аргументу	те ж	збігається з типом аргументу
Sqrt(X)	квадратний корінь	те ж	дійсний
Trunc(X)	відсікання дробової частини дійсного числа	те ж	цілий

Функції Int, Round і Trunc можуть розглядатися також, як функції перетворення типу, оскільки перетворять вираз дійсного типу в значення цілого типу.

Інші стандартні процедури і функції будуть приведені при розгляді відповідних типів даних.

1.1.11. Оператори

Основне призначення програми – виконання деяких дій по обробці даних. Для опису цих дій і призначені **оператори**. Оператори умовно можна розділити на прості і складні.

Прості оператори – це оператори, що не містять як складові частини інші оператори. До простих відносяться: оператор присвоювання, оператор процедури, оператор переходу goto, порожній оператор. До складних відносяться: складений оператор, умовний оператор if, оператор вибору case, оператор циклу з параметром for, оператор циклу з передумовою while, оператор циклу з післяумовою repeat, і оператор приєднання with.

Оператори можна позначати мітками і посилатися на них в операторах переходу goto. Порожній оператор не виконує ніяких дій, до нього не входять які-небудь символи. Наприклад, якщо в програмі йдуть два підряд символи «крапка з комою», те це означає, що між ними знаходиться порожній оператор. Чи, якщо перед зарезервованим словом end стоїть «;» , то це теж означає, що між «;» і end стоїть порожній оператор. Порожній оператор може бути позначений міткою і, таким чином, бути використаним для передачі керування в програмі.

PDF-XChange Click to buy NOW! www.docu-track.com

Оператор присвоювання призначений для заміни поточного значення змінної новим значенням, що задається.

У загальному виді оператор записується:

< змінна > := <вираз >;

Змінна і вираз повинні бути ідентичного чи сумісного для присвоювання типу. Наприклад, змінної дійсного типу можна привласнити значення виразу дійсного чи цілого типу. Змінній ж цілого типу не можна привласнити значення виразу дійсного типу.

Приклади використання оператора присвоювання:

```
a := b+c;
x := (0<a) and (a<3);
w := sin(sqr(t))/(s+ln(v));
s := 'рядок';
d := [ red, white, black, blue ];
```

1.1.12. Введення-виведення інформації

У консольному застосуванні введення даних із клавіатури здійснюється за допомогою стандартних процедур read і readln, а виведення на екран дисплея – за допомогою процедур write і writeln.

1.1.12.1. Процедури Read иReadln

У Object Pascal має дві основні процедури введення Read і Readln, що використовуються для читання даних, які вводяться з клавіатури. Загальний формат цих процедур наступний:

```
Read(елемент1,елемент2,...); або
Readln(елемент1,елемент2,...);
```

де кожен елемент являє собою змінну цілого, дійсного, символьного чи рядкового типу, наприклад:

```
read(x,y,z);
readln(name,age);
```

При виконанні процедури read відбувається наступне.

Програма припиняє свою роботу і чекає, поки на клавіатурі будуть набрані потрібні дані і натиснута клавіша Enter.

Після натискання клавіші Enter, уведені значення присвоюються змінним, імена яких зазначені в процедурі read.

Дані, що вводяться з клавіатури, являють собою символьні рядки, незалежно від того, що вони містять – числа чи символи. При виконанні процедури read ці рядкові значення автоматично перетворяться до типу тих змінних, котрим вони призначені. Наприклад, у результаті виконання процедури

```
read(x0);
```

і введення з клавіатури рядка 32, значенням змінної x0 буде ціле число 32, якщо x0 мало тип integer, і дійсне число 32.0 , якщо x0 має тип real.

Одна процедура read дозволяє ввести значення декількох змінних. Якщо вводяться числові значення, то вони повинні бути набрані в одному рядку і розділені пробілами. Наприклад, якщо тип змінних x,y і z – real, те в результаті виконання процедури

```
read(x,y,z);
```

і введення з клавіатури рядка

5.25 10 –0.05

змінні будуть мати наступні значення:

x=5.25; y=10.0; z=-0.05.

Якщо в рядку екрана після запуску програми набрано більше чисел, чим задано елементів у списку введення процедури read, то частина рядка, що залишилася, буде

ігнорована чи оброблена наступною процедурою read, якщо така мається. Наприклад, якщо i, k, l – змінні цілого типу, то в результаті виконання процедур:

```
read(i,k);
read(l);
```

і введення з клавіатури рядка

5 10 15

змінні одержать наступні значення:

I=5, k=10; l=15.

На відміну від процедури read процедура readln після уведення всіх зазначених у процедурі елементів списку введення здійснює перехід до наступного рядка дисплея. Звідси випливає, що якщо в списку введення процедури readln зазначено менше елементів, чим розташовано значень у рядку дисплея, те надлишкова частина рядка буде загублена, а наступна процедура read чи readln буде вимагати введення даних з нового рядка дисплея.

Наприклад, у результаті виконання процедур

```
readln(s,t,w);
read(q);
```

і введення з клавіатури рядка

6 9 12 15

змінні одержать наступні значення (у припущенні, що вони усі мають цілий тип):
s=6, t=9, w=12. Після чого програма буде очікувати натискання клавіші Enter і введення нового числа з нового рядка дисплея, щоб присвоїти його змінній q.

Розглянемо рішення наступної задачі. Нехай потрібно ввести дані:
a=3,7; b=-1,6·10^{-6}; k=101; l=-25; c='*'; d='?'; p=true і зобразити рядок введення даних на екрані дисплея.

При рішенні цієї задачі варто враховувати, що дані цілого і дійсного типів відокремлюються друг від друга пробілами в рядку введення. Це значення змінних a,b,k і l. Ціла частина від дробової відокремлюється крапкою. Дані символьного типу (значення змінних c і d) записуються в рядку введення підряд, без використання пробілу як роздільника. Дані логічного типу (значення змінної p) не можна вводити з використанням процедури введення. Одним з варіантів рішення цієї задачі може бути наступний:

```
…….
const p=true;
…….
readln(a,b,k,l);
readln(c,d);
…….
```

Рядок введення даних:
3.7 –1.6e-6 101 –25
*?

1.1.12.2. Процедури Write і Writeln

Призначення процедури writeln – виводити інформацію на екран. У загальному виді вона записується в такий спосіб:

```
writeln(елемент1,елемент2,...);
```

де кожен елемент – це те, що потрібно надрукувати на екрані. Елементом списку виведення може бути, наприклад, ціле чи дійсне число (3, 42, -1732.3), символ ('a', 'Z'), рядок ('Hello, world') чи логічне значення (true). Крім того, їм може бути іменована константа, змінна, роз'іменований вказівник чи звертання до функції, якщо вона повертає значення, що має цілий, дійсний, символьний, рядковий чи логічний тип. Всі елементи друкуються в одному рядку дисплея в заданому порядку. Після цього курсор встановлюється в початок наступного рядка. Якщо ви хочете залишити курсор в тім же рядку після останнього елемента, то використовуйте процедуру:

write(елемент1,елемент2,...);

Коли роздруковуються елементи списку виведення за допомогою процедури writeln, між ними пробіли автоматично не вставляються. Тому, якщо ви хочете розділити виведені значення пробілами, то вони повинні бути зазначені в списку висновку безпосередньо, наприклад:

writeln(a,' ',b,' ',...);

Припустимо, що змінні x,y,z і student мають значення: x = 2; y:= 4; z := 6; student := 'Ivan'. Тоді виконання процедур виведення приведе до появи на екрані наступних рядків:

Процедури виведення	Результат
writeln(x,y,z);	246
writeln(x,' ',y,' ',z);	2 4 6
writeln('Hello',student);	HelloIvan
writeln('Hello, ',student,'.');	Hello, Ivan.

Для поліпшення зовнішнього вигляду виведених на екран дисплея даних можна використовувати формат (опис поля виведення) для визначення кількості позицій, що займе на екрані дисплея елемент зі списку виведення. У цьому випадку процедура виведення буде мати вид:

writeln(елемент1:ширина1, елемент2:ширина2,...)

де ширина – цілий вираз (константа, змінна, звертання до функції чи комбінація з них), що визначає загальну довжину поля, у якому повинний бути записаний елемент. Наприклад, для змінних цілого типу, що мають значення k=50, m=3, n=200 форматне виведення на екран дисплея дасть наступний результат:

Процедури виведення	Результат
writeln(k,m,n);	503200
writeln(k:2,m:2,n:2);	50 3200
writeln(k:3,m:3,n:3);	50 3200
writeln(k,m:2,n:4);	50 3 200

Відзначимо, що елемент доповнюється початковими пробілами ліворуч у тому випадку, коли задана ширина поля перевищує кількість цифр у числі. Усередині поля значення вирівнюється за правою границею поля.

Що відбудеться, якщо ширина поля менше, ніж необхідно? В другій процедурі writeln із приведеного вище приклада для змінної n, що має значення 200, зазначена ширина поля 2, хоча необхідна ширина 3. Ми бачимо, що в цьому випадку Object Pascal збільшив ширину поля до мінімально необхідного розміру.

Розглянутий вище спосіб завдання формату можна використовувати для цілих і дійсних значень, символів, рядків і логічних виразів. Однак при такім завданні ширини поля дійсні числа роздруковуються в експонентній формі. Так для w = 123.45 будемо мати:

Процедури виведення	Результат
writeln(w);	1.23450000000000E+02
writeln(w:8);	1.2E+0002

Тому в Object Pascal для дійсних значень передбачено після ширини поля через двокрапку вказувати і другий параметр – кількість позицій, що відводяться для дробової частини з загальної ширини поля. Цей другий параметр указує роздрукувати дійсне число у форматі з фіксованою крапкою і визначає, скільки цифр помістити після десяткової крапки. Наприклад, змінну w можна вивести на екран у такий спосіб:

Процедури висновку	Результат
writeln(w:6:2);	123.45
writeln(w:8:2);	123.45
writeln(w:8:4);	123.4500

Приклад 1.3.

Складемо програму для введення і виведення даних, раніше використаних нами в пункті 1.1.12.1: a=3,7; b=-1,6·10^{-6}; k=101; l=-25; c='*'; d='?'; p=true . Виведемо дані по

стандартному форматі (тобто без указання формату), за форматом із зазначеною шириною поля і зобразимо рядки виведення.

Рішення.

При використанні процедур уведення read і readln прийнято перед виконанням кожної з них видавати на екран дисплея відповідне повідомлення-підказку про те, що необхідно ввести. Це можна зробити, використовуючи процедури write чи writeln.

Консольне застосування створюється в операційній системі Windows, де символи кодуються відповідно до таблиці кодів ANSI. При запуску консольного застосування створюється так називана віртуальна MS-DOS-машина, що використовує операційну систему MS-DOS і, відповідно, таблицю кодів ASCII. Таблиці кодів ANSI і ASCII містять по 256 кодів символів, включаючи коди букв англійського і російського алфавіту. Букви англійського алфавіту знаходяться в першій половині таблиці, а російського – у другий. Відрізняються обидві таблиці саме другою половиною, що містить букви російського алфавіту. Тому, якщо в консольному застосуванні повідомлення-підказки в процедурах write чи writeln будуть набрані російською мовою, то при виконанні застосування усередині віртуальної MS-DOS-машини повідомлення будуть перекручені. Щоб уникнути цього, можна, наприклад, записувати повідомлення англійською мовою. Можна, звичайно, писати повідомлення і німецькою, і французькою, і навіть російською мовою – тільки при цьому необхідно усе-таки використовувати букви англійського алфавіту. Зазначене обмеження має місце тільки для консольних застосувань. Для віконних застосувань, створюваних і виконуваних в одній операційній системі – Windows – це обмеження відсутне.

Програма.

```
program p1_3;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const p=true;
var a,b:real; k,l:integer; c,d:char;
begin
   writeln('enter a,b');
   readln(a,b);
   writeln('enter k,l');
   readln(k,l);
   writeln('enter c,d');
   readln(c,d);
   writeln('standard format');
   writeln(a);
   writeln(b);
   writeln(k,l);
   writeln(c,d,p);
   writeln('given format');
   writeln(a:3:1,b:11:7);
   writeln(k:4,l:4);
   writeln(c:2,d:2,p:5);
   readln
end.
```

Результати роботи програми

PDF-XChange Click to buy NOW! www.docu-track.com

```
enter a,b
3.7 -1.6e-6
enter k,l
101 -25
enter c,d
*?
standard format
 3.70000000000000E+0000
-1.60000000000000E-0006
101-25
*?TRUE
given format
3.7 -0.0000016
 101 -25
 * ? TRUE
```

Як видно з результатів роботи програми, при виведенні даних у стандартному форматі значення цілого, символьного і логічного типів займають на екрані стільки позицій, скільки в них міститься цифр чи букв. Дані дійсного типу виводяться у формі з рухомою крапкою і займають 23 позиції.

1.1.13. Створення консольного застосування

Створити консольне застосування можна різними способами. Найбільш простий з них наступний.

Після запуску середовища Delphi у головному меню виберіть пункт File, а потім у розкривному меню виберіть пункт New. У результаті з'явиться ще одне меню, у якому треба вибрати пункт Other... . Надалі для стислості подібну последовательность дій будемо позначати: File|New|Other... . Після цього відкриється вікно так називаного репозитория (архіву) Delphi, призначеного для нагромадження типових форм і проектів (див. мал. 1.5) :

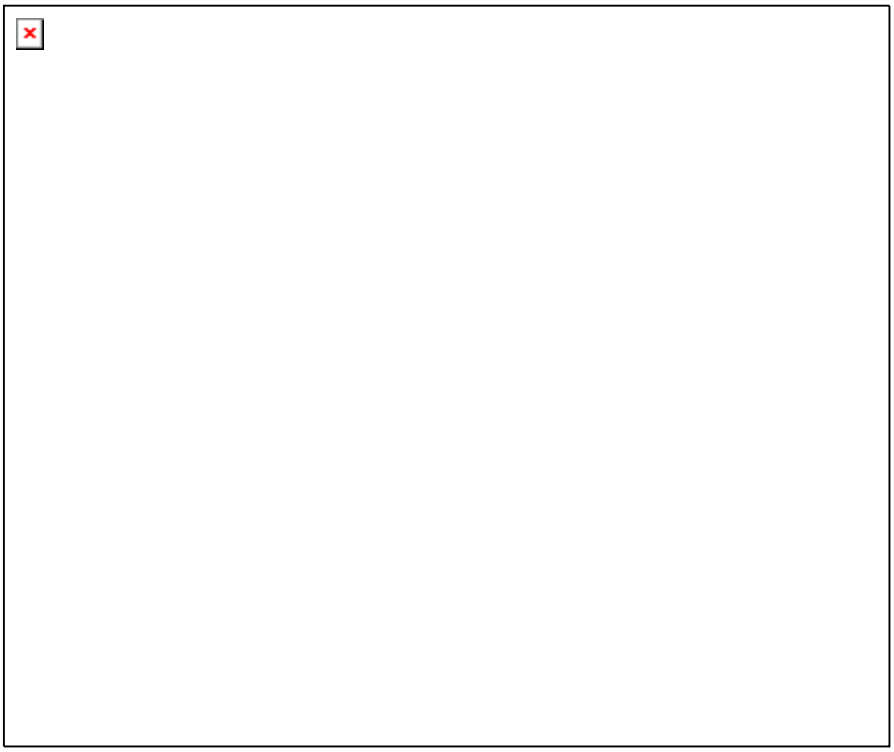
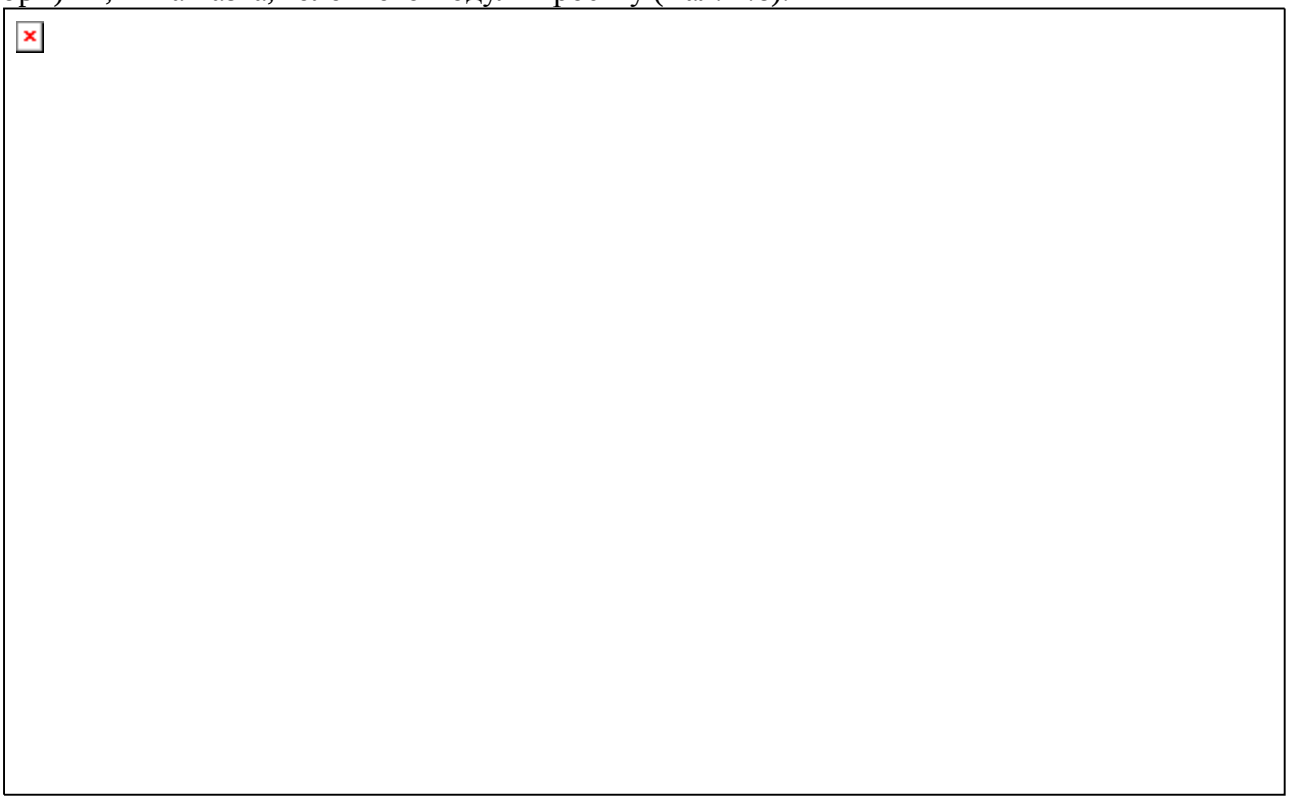


Рис 1.5. Вікно архіву Delphi.

Далі щигликом миші треба вибрати піктограму з підписом «Console Application» і натиснути кнопку OK. Після цього розкриється вікно файлу проекту (він має розширення dpr) чи, інша назва, головного модуля проекту (мал. 1.6):



Мал. 1.6. Вікно файлу проекту.

Замість коментарю

{ TODO -oUser -cConsole Main : Insert code here }

наберемо у вікні файлу проекту текст програми, у якій уводяться два числа й обчислюється їхня сума (мал. 1.7):

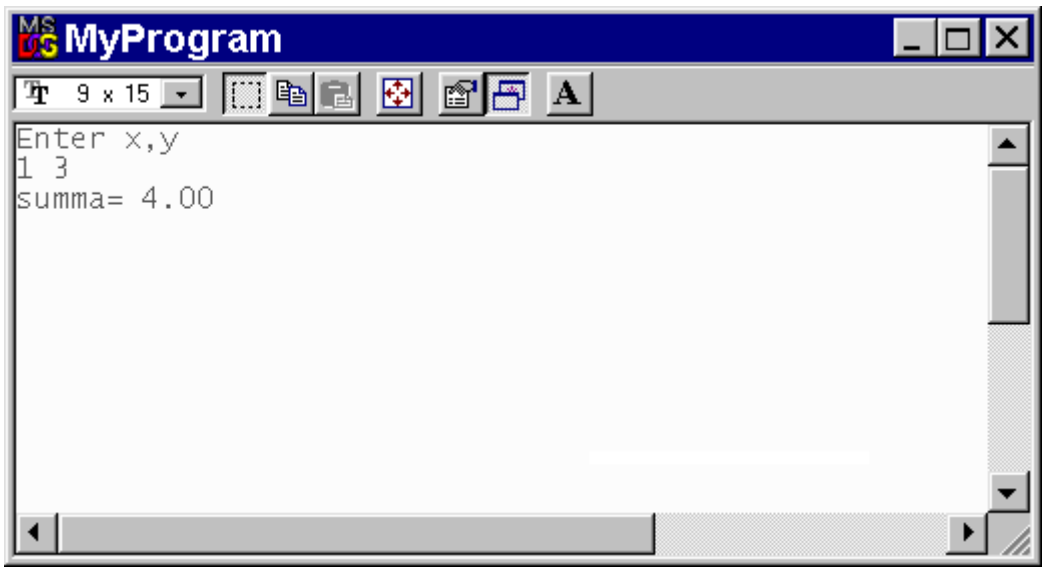
Рис 1.7. Текст консольного застосування.

Перед тим як запустити нашу програму на виконання, необхідно зберегти її. Зробити це можна за допомогою команди File | Save All . Кожен проект рекомендується зберігати в окремій папці. Помітимо, що за замовчуванням файл проекту має ім'я ProjectN.dpr, а програма – ProjectN, де N – деяке число. Ми можемо зберегти файл проекту під будь-яким ім'ям, наприклад MyProgram.dpr . Автоматично Delphi змінить ім'я програми на MyProgram. З огляду на це, можна сформулювати наступне правило: ті рядки програмного коду, що формуються середовищем Delphi, редагувати не можна. У противному випадку ваша робота буде ускладнена різними неприємностями, наприклад, повідомленнями про помилки.

Після того як ви зберегли проект, запустимо його на виконання. Це можна зробити, виконавши команду Run|Run , або натиснути функціональну клавішу F9, або вибрати кнопку Run на інструментальній панелі швидких кнопок:

Після успішної компіляції і запуску програми на виконання на екрані з'явиться стандартне вікно програми DOS. На мал. 1.8. зображене DOS-вікно, у якому працює створене нами в Delphi консольне застосування:

PDF-XChange
Click to buy NOW!
www.docu-track.com



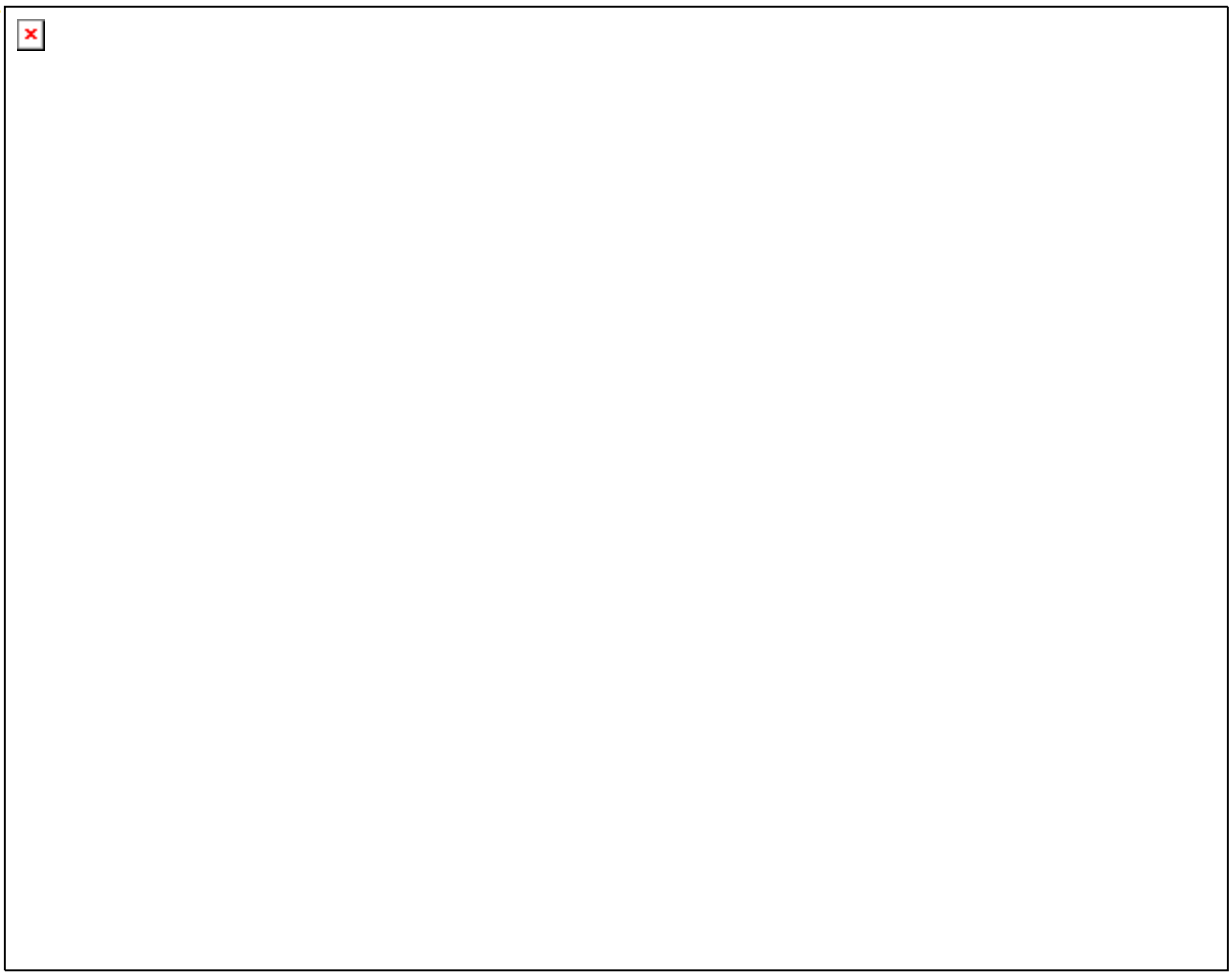
Мал. 1.8. DOS-вікно, у якому працює консольне застосування.

Після виведення символьного рядка 'Enter x,y' програма очікує введення значень змінних, про що свідчить миготливий курсор. Після введення значень на екрані з'явиться рядок, що містить відповідь.

Останньою в нашій програмі виконується процедура readln без параметрів. Її дія зводиться до того, що вона припиняє виконання програми до натискання на клавішу Enter. Таким чином, для того щоб завершити роботу програми і повернутися в середовище Delphi, нам варто натиснути клавішу Enter.

Другий спосіб створення консольного застосування наступний.

Після запуску Delphi чи виконання команди File | New | Application необхідно закрити вікно форми (вікно з заголовком Form1) і вікно модуля застосування (вікно з заголовком Unit1.pas). Ці вікна зображені на малюнку 1.9.

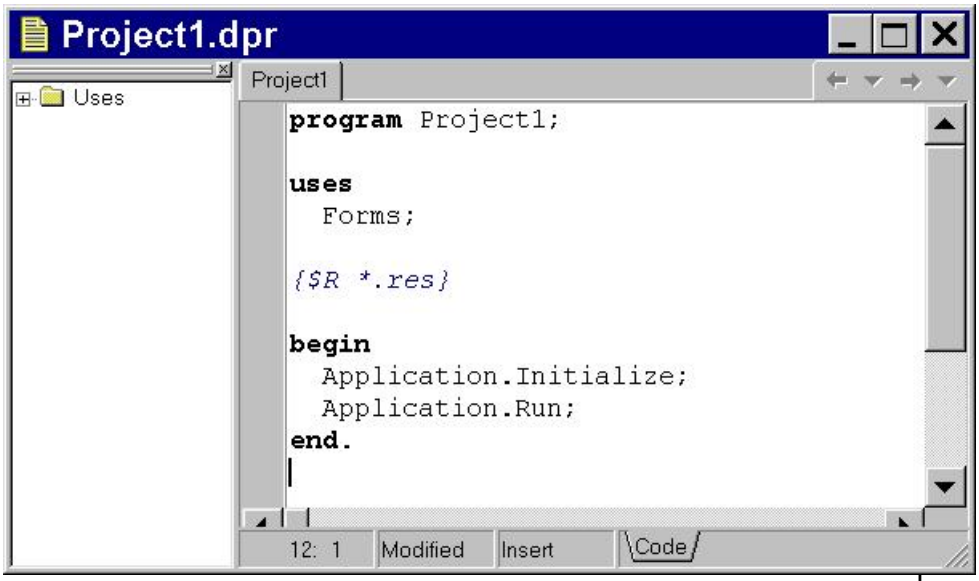


Мал. 1.9. Вікна, використовувані в середовищі Delphi.

При закритті вікна модуля Delphi виводить запит:

Save changes to Unit1.pas? (Зберегти зміни в Unit1.pas?) ,

на який треба відповісти «Ні», тобто натиснути кнопку No. У результаті на екрані залишаться головне вікно Delphi (заголовок: Delphi 6 – Project1), вікно Інспектора Об'єктів (заголовок: Object Inspector) і вікно Дерева Об'єктів (заголовок: Object TreeView) . Два останніх вікна теж можна закрити. Після цього варто виконати команду Project|View Source. У результаті відкриється вікно Project1.dpr файлу проекту:



Мал. 1.10. Вікно файлу проекту, отримане після виконання команди Project | View Source.

Далі файл проекту треба зберегти. Якщо ім'я файлу буде змінено, то це відіб'ється також у заголовку програми. Потім видалимо всі рядки, крім рядків, що містять ключові слова program, uses, begin і end. Після цього можна набрати текст програми, подібно тому, як це приведено на мал. 1.7.

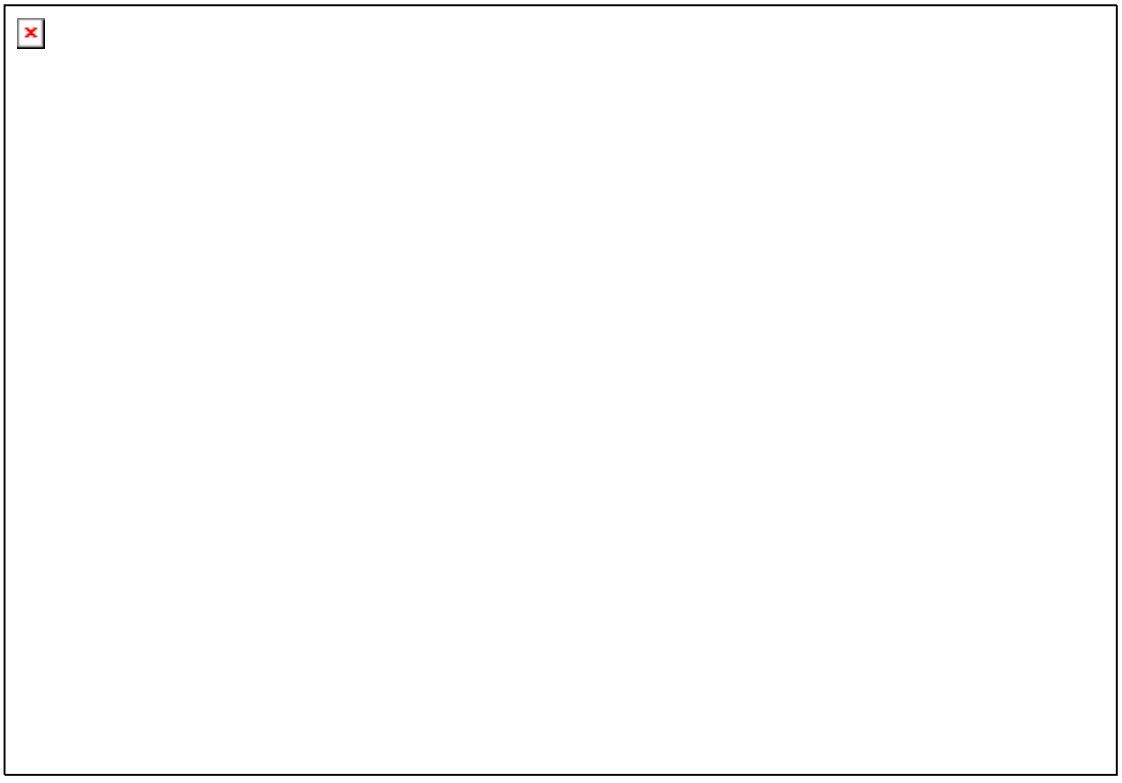
Якщо подивитися вміст папки, у якій ми зберегли проект, то ми знайдемо там наступні файли:

- MyProgram.dpr – файл проекту (головний модуль проекту).
- MyProgram.exe – файл застосування або виконуваний файл. Він буде створений компілятором, якщо в процесі компіляції не буде виявлено синтаксичних помилок. Іншими словами, якщо вам удалося запустити свою програму на виконання, наприклад при натисканні на клавішу F9, те це і буде означати , що виконуваний файл створений. Виконуваний файл є автономним виконуваним файлом, тобто для його роботи не вимагаються які-небудь інші файли. Ви можете запустити його на виконання як будь-яку іншу програму, виконавши, наприклад, команду «Відкрити» у програмі Провідник.
- MyProgram.cfg – файл конфігурації проекту. Він містить установки проекту, наприклад використовувані директиви компілятора.
- MyProgram.dof – файл опцій проекту. У ньому зберігається інформація, необхідна для правильної роботи проекту в цілому або, більш точно, установки опцій проекту.

Файли опцій проекту і конфігурації проекту створюються Delphi автоматично, одночасно зі створенням файлу проекту.

Крім перерахованих файлів, часто в папці можна знайти файли з розширенням ~dpr , наприклад MyProgram.~dpr . Це резервна копія файлу проекту. Вона створюється при редагуванні існуючого файлу проекту і містить попередній варіант тексту програми. Резервна копія може придатися, якщо ви вирішите відмовитися від внесених у програму виправлень. При цьому вам досить буде забрати символ «~» з імені файлу.

При запуску програм ви рано чи пізно зштовхнетеся з так називаними помилками компіляції. Рядок, у якому допущена помилка, буде підсвічений коричневим кольором, а в нижній частині вікна файлу проекту з'явиться повідомлення про помилку (див. мал. 1.11):



Мал. 1.11. Повідомлення компілятора про синтаксичну помилку.

Наприклад, у даному випадку компілятор повідомляє про те, що змінна x не описана в розділі var програми. Компілятор мови Object Pascal дозволяє досить легко визначати місце і характер допущеної помилки. Найчастіше допускаються такі помилки, як неправильно записані ключові слова чи оператори, неописані змінні, відсутність апострофа наприкінці символьного рядка і т.д. Для боротьби з цим злом є тільки один спосіб – більше часу приділяти програмуванню в Delphi, здобуваючи тим самим необхідний досвід. Більш докладно про різні помилки ми поговоримо в главі 1.9, а поки відзначимо наступне.

Якщо після запуску програми компілятор видав повідомлення про помилку, то її треба знайти і виправити. Потім знову запустити програму на виконання. Цей процес повторюється доти, поки всі синтаксичні помилки не будуть виправлені, після чого програма запуститься на виконання.

1.2. Прості типи

Простими є порядкові, дійсні типи і тип дата-час.

Порядкові типи характеризуються тим, що відповідні їм значення утворять скінченну упорядковану множину і кожне значення має свій порядковий номер.

Значеннями дійсних типів є числа, що мають дробову частину, тобто дійсні числа. Кожен дійсний тип являє собою деяку підмножину дійсних чисел.

Тип дата-час призначений для збереження дати і часу.

1.2.1. Порядкові типи

До порядкових типів відносяться цілі, логічні, символьні, перелічувані типи і тип-діапазон. Для виразів порядкового типу визначені наступні функції:

- Ord(x) – повертає порядковий номер значення даного виразу. Для цілих типів повертає саме значення x, для логічного 0 чи 1, для символьного – значення в діапазоні від 0 до 255, для перелічуваного – значення в діапазоні від 0 до 65535. Для типу-діапазону результат залежить від властивостей базового порядкового типу.
- Pred(x) – повертає величину, що передує значенню даного виразу.
- Succ(x) – повертає величину, що слідує за значенням даного виразу.

Приклади.

Вираз	Значення
Ord(10)	10
Ord(false)	0
Ord(‘z’)	90
Pred(10)	9
Pred(true)	false
Pred(‘z’)	‘y’
Succ(10)	11
Succ(false)	true
Succ(‘y’)	‘z’

Помітимо, що функція Pred не визначена для найменшого значення порядкового типу, а Succ – для самого більшого.

Для констант і змінних порядкового типу визначені також наступні функції:

- High(x) – повертає максимальне можливе значення для аргументу x.
- Low(x) – повертає мінімальне можливе значення для аргументу x.

1.2.2. Числові типи

Під числовими типами розуміють цілі і дійсні типи.

1.2.2.1. Цілі типи

Цілі типи даних призначені для представлення цілих чисел. У таблиці 1.3 перераховані використовувані в Delphi 6 цілі типи. Для кожного типу приведений діапазон значень, а також зазначено, скільки байтів займають значення відповідного типу в оперативній пам'яті ПК.

Цілі типи в Delphi 6 — Таблиця 1.3.

Тип	Діапазон значень	Розмір (у байтах)
Integer	–2147483648..2147483647	4
Cardinal	0..4294967295	4
Shortint	–128..127	1
Smallint	–32768..32767	2
Longint	–2147483648..2147483647	4
Int64	-2^{63} ..2^{63} –1	8
Byte	0..255	1
Word	0..65535	2
Longword	0..4294967295	4

Найбільша продуктивність центрального процесора й операційної системи досягається при використанні типів Integer і Cardinal. Всі інші цілі типи, крім Int64, являють собою підмножини двох вище зазначених типів.

При застосуванні до даних цілого типу операцій *, div, mod, +, - отриманий результат буде також цілого типу. Те ж можна сказати і про стандартні функції abs і sqr.

Якщо в арифметичному виразі використовуються значення тільки якого-небудь одного з цілих типів, то результат виразу буде мати такий же тип. Якщо ж у виразі використовуються значення різних цілих типів, то результат буде мати тип Integer.

При роботі з даними цілого типу необхідно стежити за тим, щоб значення змінних чи виразів не виходили за припустимі границі діапазону значень. За замовчуванням діє директива компілятора {$R-}, яка означає, що перевірка виходу значень із припустимого діапазону скасована. Це приводить до того, що при виході за припустимі границі значення буде змінюватися циклічно, тобто величина циклу буде дорівнює кількості значень, що входять у діапазон значень для відповідного типу. Наприклад, якщо для змінної g типу byte виконати наступні оператори

```
g:=1;
g:=g+255;
```

те вона прийме значення 0. А при виконанні операторів

```
g:=1;
g:=g-2;
```

змінна g прийме значення 255.

Для того щоб контролювати вихід за границі діапазону значень, треба в програмі помістити директиву компілятора {$R+}. У цьому випадку при виході за границы діапазону буде генеруватися виняток і на екран буде виведене повідомлення про помилку.

1.2.2.2. Дійсні типи

Дійсні типи призначені для представлення дійсних чисел, тобто чисел, що мають дробову частину. Оскільки дані в комп'ютері зберігаються у виді двоичных кодів, то дійсні числа представляються приблизно, хоча і з великим ступенем точності. Якщо провести порівняння з цілими даними, то на відміну від дійсних чисел, дані цілого типу представляються в двоичном виді точно. У Dephpi 6 є сім дійсних типів, що дозволяють одержати потрібний результат практично з будь-яким заданим ступенем точності:

Дійсні типи в Delphi 6 — Таблиця 1.4.

Тип	Діапазон значень	Кількість значущих цифр	Розмір (у байтах)
Real	$5.0*10^{-324}$ .. $1.7*10^{308}$	15–16	8

Real48	$2.9*10^{-39}$ .. $1.7*10^{38}$	11–12	6
Single	$1.5*10^{-45}$ .. $3.4*10^{38}$	7–8	4
Double	$5.0*10^{-324}$ .. $1.7*10^{308}$	15–16	8
Extended	$3.6*10^{-4951}$ .. $1.1*10^{4932}$	19–20	10
Comp	$-2^{63}+1$ .. $2^{63}-1$	19–20	8
Currency	–922337203685477.5808.. 922337203685477.5807	19–20	8

Найбільша продуктивність центрального процесора досягається при використанні типу Real. У Delphi 6 йому еквівалентний тип Double.

Тип Real48 призначений тільки для сумісності з ранніми версіями Delphi. При його використанні продуктивність процесора мінімальна.

Тип Extended дозволяє робити розрахунки з максимальною точністю. Більш того, арифметичний співпроцесор виконує операції з дійсними числами, представленими у форматі Extended. Якщо ж ми користаємося типами Real, Single чи Double, то результат обчислень усікається до відповідної точності. Однак при використанні типу Extended варто пам'ятати, що дані цього типу займають в оперативній пам'яті ПК більше всього місця – 10 байт.

Типи Comp і Currency застосовуються для бухгалтерських розрахунків. У типі Comp дробова частина відсутня, а в типі Currency вона обмежена чотирма знаками. В оперативній пам'яті ПК значення цих типів представляються як дані цілого типу, що займають 8 байт. Це дозволяє мінімізувати помилки округлення в грошових розрахунках. З іншого боку, значення типів Comp і Currency сумісні зі значеннями інших дійсних типів, тобто над ними можуть виконуватися всі дійсні операції і їм можна присвоїти значення змінних і виразів інших дійсних типів. Але при цьому не можна забувати про те, що буде відбуватися усікання значень до чотирьох знаків у дробовій частині для типу Currency і повне відкидання дробової частини для типу Comp.

За умови, що хоча б один з операндов дійсного типу (іншій може бути і цілим), операції *, /, +, - дають дійсний результат.

Для роботи з дійсними даними в Delphi 6 є стандартні функції. Вони приведені в таблиці 1.2. (пункт 1.1.10).

1.2.2.3. Модуль Math

Приведені в таблиці 1.2 стандартні математичні функції можуть бути використані в будь-якій програмі без усяких додаткових описів. Такою властивістю володіють підпрограми, описані в модулі System, що автоматично підключається до будь-якого проекту, або якщо обробка таких підпрограм покладена безпосередньо на компілятор. Для того щоб скористатися процедурами і функціями, що знаходяться в інших модулях, необхідно ці модулі підключити до нашого застосування, тобто вказати ім'я модуля в операторі uses. Так, наприклад, до складу пакета Delphi входить модуль Math, що містить на додаток до розглянутих функцій широкий набір математичних, тригонометричних, статистичних і інших функцій. Деякі часто використовувані функції з цього модуля приведені в таблиці 1.5.

Математичні функції з модуля Math. Таблиця 1.5.

Функція	Опис	Тип аргументу(ів)	Тип результату
Power(a,b)	обчислює a^b; при $a<0$ ступінь b повинний бути цілим	Extended	Extended
Log(a,b)	обчислює $\log_a(b)$	те ж	те ж
Tan(x)	обчислює $tg(x)$	те ж	те ж

CoTan(x)	обчислює $ctg(x)$	те ж	те ж
ArcSin(x)	обчислює $\arcsin(x)$	те ж	те ж
ArcCos(x)	обчислює $\arccos(x)$	те ж	те ж

Помітимо, що для обчислення $\sqrt[n]{a}$ можна скористатися функцією Power з показником ступеня *1/n:*

Power(a,*1/n*)

1.2.2.4. Арифметичні вирази

В арифметичному виразі арифметичні операції застосовуються до дійсних і цілих чисел. Опис арифметичних операцій +, -, *, /, div і mod приведено в пункті 1.1.9.

При запису арифметичних виразів важливо вміти правильно визначити тип результату. Це можна зробити, керуючись наступними правилами.

- Результат виразу A/B завжди має тип Extended, незалежно від того, мають операнди A і B цілий чи дійсний тип.
- Якщо у виразах A*B, A+B, A-B хоча б один операнд є дійсним, то результат має тип Extended.
- Якщо у виразях A*B, A+B, A-B обидва операнди цілі й один з них має тип Int64, то результат теж буде мати тип Int64. Для всіх інших цілих типів результат буде мати тип Integer.
- Операції div і mod застосовуються до цілочислених операндів, і результат їхнього виконання має тип Integer.

Порядок обчислень в арифметичному виразі такий же, як і в математиці. Спочатку обчислюються значення стандартних функцій, потім операції множення і ділення (*, /, div, mod), потім операції додавання і віднімання (+, -). Якщо у виразі мається декілька операції з одним пріоритетом, то вони виконуються зліва на право. Якщо потрібно змінити порядок виконання операцій, то варто використовувати круглі дужки.

Розглянемо деякі приклади арифметичних виразів.

Приклад 1.4.

Визначити значення виразу

150 div 25 mod 5.

Рішення.

Цілочислені операції div і mod мають однаковий пріоритет виконання. Операції одного пріоритету виконуються злива на право у порядку їхньої появи у виразі, тому першою виконається операція 150 div 25 (результат: 6), другою 6 mod 5 (результат: 1).

Відповідь: 1.

Приклад 1.5.

Визначити значення виразу:

pred(succ(trunc(2.7)+round(3.4))).

Рішення.

Відповідно до правил обчислення арифметичних виразів, у першу чергу обчислюються вирази в самих внутрішніх круглих дужках, тому порядок обчислень наступний:

Обчислення функції чи операція	Результат
trunc(2.7)	2
round(3.4)	3
2+3	5
succ(5)	6
pred(6)	5

Відповідь: 5.

Приклад 1.6.

Записати за правилами мови Object Pascal вираз

$$\frac{\sin^5 a - b}{|b| + tgb} + 10^{-5} .$$

Рішення.

При рішенні цієї задачі варто скористатися функціями Power і Tan, опис яких міститься в модулі Math:

(power(sin(a),5)-b)/(abs(b)+tan(b))+1e-4 .

Приклад 1.7.

Записати за правилами мови Object Pascal вираз:

$$\sqrt[3]{e^{x-1/\sin x}} .$$

Рішення.

Приведений вираз можна спростити, тобто

$$\sqrt[3]{e^{x-1/\sin x}} = e^{\frac{(x-1/\sin x)}{3}} .$$

Тоді маємо:

exp((x-1/sin(z))/3).

При використанні цього виразу в програмі необхідно виконувати перевірку того, що sin(x) не дорівнює 0.

Приклад 1.8.

Записати мовою Object Pascal оператор присвоювання:

$$y = \ln|arctg(x) - \sin(ax)| + \sqrt[3]{ax} .$$

Рішення.

y:=ln(abs(arctan(x)-sin(a*x)))+exp(1/3*ln(a*x));

Приклад 1.9.

Записати мовою Object Pascal оператор присвоювання:

$$b = \arcsin^2 z + \sqrt{|x - y|} .$$

Рішення.

Обчислення arcsin(x) може бути здійснене за допомогою однойменної функції з модуля Math. У результаті одержимо:

b:=sqr(arcsin(z))+sqrt(abs(x-y));

Помітимо, що аргумент функцій arcsin і arccos повинний лежати в межах від –1 до 1. Функція arcsin повертає результат у межах від –pi/2 до pi/2, а результат, що повертається функцією arccos, знаходиться в межах від 0 до pi.

1.2.3. Символьні типи

Дані символьного типу призначені для збереження одного символу. У Delphi 6 є три символьних типи:

Символьні типи в Delphi 6

Таблиця 1.6.

Тип	Розмір у байтах
ANSIChar	1
WideChar	2
Char	1

Тип ANSIChar являє собою так називані Ansi-символи. Це символи, що використовуються в операційних системах сімейства Windows. Кожному символу відповідає число, або, інакше говорять, – код ANSI, що розшифровується як American National Standard Institute – американський національний інститут стандартизації, у якому розроблений цей код. Нижче приведена таблиця відповідності символів і кодів у відповідності зі стандартом ANSI.

Кодування символів ANSI, використовуване в операційних системах сімейства Windows.

Таблиця 1.7.

32	55 7	78 N	101 e	124 |	147 "	170 Є	193 Б	216 Ш	239 п
33 !	56 8	79 O	102 f	125 }	148 "	171 «	194 В	217 Щ	240 р
34 "	57 9	80 P	103 g	126 ~	149 •	172 ¬	195 Г	218 Ъ	241 с
35 #	58 :	81 Q	104 h	127 □	150 –	173 -	196 Д	219 Ы	242 т
36 $	59 ;	82 R	105 i	128 Ђ	151 —	174 ®	197 Е	220 Ь	243 у
37 %	60 <	83 S	106 j	129 Ѓ	152 □	175 Ї	198 Ж	221 Э	244 ф
38 &	61 =	84 T	107 k	130 ‚	153 ™	176 °	199 З	222 Ю	245 х
39 '	62 >	85 U	108 l	131 ѓ	154 љ	177 ±	200 И	223 Я	246 ц
40 (	63 ?	86 V	109 m	132 „	155 ›	178 І	201 Й	224 а	247 ч
41)	64 @	87 W	110 n	133 …	156 њ	179 і	202 К	225 б	248 ш
42 *	65 A	88 X	111 o	134 †	157 ќ	180 ґ	203 Л	226 в	249 щ
43 +	66 B	89 Y	112 p	135 ‡	158 ћ	181 µ	204 М	227 г	250 ъ
44 ,	67 C	90 Z	113 q	136 €	159 џ	182 ¶	205 Н	228 д	251 ы
45 -	68 D	91 [	114 r	137 ‰	160	183 ·	206 О	229 е	252 ь
46 .	69 E	92 \	115 s	138 Љ	161 Ў	184 ё	207 П	230 ж	253 э
47 /	70 F	93]	116 t	139 ‹	162 ў	185 №	208 Р	231 з	254 ю
48 0	71 G	94 ^	117 u	140 Њ	163 J	186 є	209 С	232 и	255 я
49 1	72 H	95 _	118 v	141 Ќ	164 ¤	187 »	210 Т	233 й	
50 2	73 I	96 `	119 w	142 Ћ	165 Ґ	188 j	211 У	234 к	
51 3	74 J	97 a	120 x	143 Џ	166 ¦	189 S	212 Ф	235 л	
52 4	75 K	98 b	121 y	144 ђ	167 §	190 s	213 Х	236 м	
53 5	76 L	99 c	122 z	145 '	168 Ё	191 ї	214 Ц	237 н	
54 6	77 M	100 d	123 {	146 '	169 ©	192 А	215 Ч	238 о	

Усього в таблиці міститься 256 символів, що кодуються числами від 0 до 255. У таблиці 1.7 не приведені символи з номерами від 0 до 31, тому що вони є службовими символами, тобто призначені не для відображення інформації, а для керування відображенням інформацією. Наприклад, символ з кодом 9 вставляє в текст знак табуляції, а символ з кодом 13 означає кінець абзацу, тобто еквівалентний натисканню клавіші Enter.

Тип WideChar призначений для збереження так званих Unicode-символів, що на відміну від Ansi-символів займають два байти. Це дозволяє кодувати символи числами від 0 до 65535 і використовується для представлення різних азіатських алфавітів. Перші 256 символів у стандарті Unicode збігаються із символами Ansi.

Оскільки тип WideChar призначений для використання в операційній системі Windows, то його варто використовувати при створенні віконних застосувань (див. частину 2).

Тип Char у Delphi 6 еквівалентний типу AnsiChar і забезпечує найбільшу продуктивність.

Для відображення множини символів у підмножину натуральних чисел і назад є наступні дві стандартні функції:

ord(c) – дає порядковий номер символу c;

chr(i) – дає символ з порядковим номером i.

Функція chr є циклічною з величиною періоду рівною 256. Її значення приведені в таблиці 1.7 для віконних застосувань і в таблиці 1.8 для консольних застосувань. Аргументами функції можуть бути як числа, що перевищують 255, так і негативні. Наприклад, звертання до функції chr(90) поверне символ 'Z'. Це ж значення буде повернуто при звертаннях chr(90+256) і chr(90-256). Функція ord виконує зворотню операцію, тобто ord('Z') поверне 90.

Замість функції chr можна скористатися оператором #, що також поверне символ, код якого зазначений після оператора. Наприклад, якщо змінна s має тип char, то ми можемо записати наступні оператори, що будуть еквівалентні:

```
s := chr(72);
s := #72;
```

Оскільки символьні типи відносяться до порядкових, для них визначені такі функції, як Pred, Succ. Наприклад, Pred('B') поверне символ 'A', а Succ('B') поверне 'C'.

До значень символьних типів можна застосовувати операції відношення: <, >, <=, >=, <=, <>, =. При порівнянні символів порівнюються відповідні їм коди, і більшим буде символ, що має більший код. Наприклад, істинними будуть наступні відношення:

```
'A'<'B' ,
'9'<'A' ,
'F'<'Ф' ,
'a'<'A' .
```

Як уже відзначалося раніше, консольне застосування виконується під керуванням операційної системи MS DOS, що эмулируется операційними системами сімейства Windows. На відміну від Windows у MS DOS використовується кодування символів ASCII (див. табл. 1.8.) :

Кодування символів ASCII, використовуване в операційній системі MS DOS.

Таблиця 1.8.

32		55	7	78	N	101	e	124	|	147	у	170	к	193	┴	216	╪	239	я
33	!	56	8	79	O	102	f	125	}	148	ф	171	л	194	┬	217	┘	240	Ё
34	"	57	9	80	P	103	g	126	~	149	х	172	м	195	├	218	┌	241	ё
35	#	58	:	81	Q	104	h	127	▯	150	ц	173	н	196	─	219	▉	242	Є
36	$	59	;	82	R	105	i	128	А	151	ч	174	о	197	┼	220	▄	243	є
37	%	60	<	83	S	106	j	129	Б	152	ш	175	п	198	╞	221	▌	244	Ї
38	&	61	=	84	T	107	k	130	В	153	щ	176	▒	199	╟	222	▐	245	ї
39	'	62	>	85	U	108	l	131	Г	154	ъ	177	▓	200	╚	223	▀	246	Ў
40	(	63	?	86	V	109	m	132	Д	155	ы	178	▓	201	╔	224	р	247	ў
41	)	64	@	87	W	110	n	133	Е	156	ь	179	│	202	╩	225	с	248	°
42	*	65	A	88	X	111	o	134	Ж	157	э	180	┤	203	╦	226	т	249	∙
43	+	66	B	89	Y	112	p	135	З	158	ю	181	╡	204	╠	227	у	250	·
44	,	67	C	90	Z	113	q	136	И	159	я	182	╢	205	═	228	ф	251	√
45	-	68	D	91	[	114	r	137	Й	160	а	183	╖	206	╬	229	х	252	№
46	.	69	E	92	\	115	s	138	К	161	б	184	╕	207	╧	230	ц	253	¤
47	/	70	F	93	]	116	t	139	Л	162	в	185	╣	208	╨	231	ч	254	■
48	0	71	G	94	^	117	u	140	М	163	г	186	║	209	╤	232	ш	255	
49	1	72	H	95	_	118	v	141	Н	164	д	187	╗	210	╥	233	щ		
50	2	73	I	96	`	119	w	142	О	165	е	188	╝	211	╙	234	ъ		
51	3	74	J	97	a	120	x	143	П	166	ж	189	╜	212	╘	235	ы		
52	4	75	K	98	b	121	y	144	Р	167	з	190	╛	213	╒	236	ь		
53	5	76	L	99	c	122	z	145	С	168	и	191	┐	214	╓	237	э		
54	6	77	M	100	d	123	{	146	Т	169	й	192	└	215	╫	238	ю		

Порівнюючи таблиці 1.7. і 1.8., можна помітити, що перші їхні половини, тобто символи з кодами 0..127, збігаються, а другі половини – коди із символами 128..255 – різні. У першій половині містяться цифри і букви англійського алфавіту, а в другий – букви російського алфавіту. Оскільки консольне застосування створюється в операційній системі Windows, а виконується як програма MS DOS, то спроби вивести російські символи на екран через розходження в кодуваннях ANSI і ASCII приречені на невдачу. Але засмучуватися не варто. По-перше, повідомлення можна виводити англійською мовою. По-друге, маючи у своєму розпорядженні таблиці 1.7. і1.8., можна скласти свою функцію для перекодування символів.

1.2.4. Логічні типи

1.2.4.1. Логічні вирази

У Object Pascal дані логічного (булева) типу можуть приймати одне з двох значень: true (істина) чи false (неправда). У таблиці 1.9. перераховані логічні типи, що визначені в Delphi 6:

Логічні типи в Delphi 6

Таблиця 1.9.

Тип	Розмір у байтах
Boolean	1
ByteBool	1
WordBool	2
LongBool	4

Основним логічним типом у Delphi 6 є Boolean. Інші типи потрібні для сумісності з логічними даними, використовуваними в Windows і деяких інших системах програмування, наприклад Visual C.

Оскільки логічні типи відносяться до порядкових, то для них визначені стандартні функції Ord, Succ, Pred. Особливістю застосування цих функцій до значень різних логічних типів є те, що для різних типів результат буде різним. Наприклад, якщо в програмі описані наступні змінні:

```
var
a:Boolean;
b:ByteBool;
c:WordBool;
d:LongBool;
```

то результат застосування до них функцій Ord, Succ і Pred можна представити у виді наступної таблиці:

Значення стандартних функцій Ord, Succ і Pred для логічних типів

Таблиця 1.10.

Змінна	Значення	Ord	Succ	Pred
a	true	1	true	false
b	true	-1	false	true
c	true	-1	false	true
d	true	-1	false	true
a	false	0	true	true
b	false	0	true	true
c	false	0	true	true
d	false	0	true	true

Як видно з таблиці 1.10., відмінності значень функцій маються між типом Boolean, з одного боку, і типами ByteBool, WordBool, LongBool, з іншого боку.

Як було сказано раніше, надалі ми будемо користатися переважно типом Boolean. Інші типи використовуються, як правило, при виклику стандартних підпрограм, що мають параметри цього типу.

Крім логічних констант і змінних, важливе значення в процесі програмування мають логічні вирази.

Логічним називається вираз, що може приймати одне з двох значень: true чи false. Простий логічний вираз складається з двох операндів і операції порівняння:

< операнд1 > < операція > < операнд2 > ,

де операнд1 і операнд2 – операнди логічного виразу, у якості яких можуть виступати вирази сумісних для порівняння типів, операція – це одна з операцій порівняння:

> (більше), < (менше), = (дорівнює), <> (нерівно),
>= (більше чи дорівнює), <= (менше чи дорівнює).

Наприклад:

```
Summa < 2000,
x = y,
Int(x) <> Round(y).
```

Операція порівняння (відношення) повертає true – істина, якщо зазначене співвідношення операндов виконується, і false – неправда, якщо співвідношення не виконується.

Операнди повинні мати сумісні типи, за винятком цілих і дійсних типів, що можуть порівнюватися один з одним:

вираз	результат
5 > 1.253	true
10 > 25.339	false
0 = 0.0	true

Рядки порівнюються по кодах символів. Нагадаємо, що для консольних застосувань використовується кодування ASCII, а для віконних – ANSI. При порівнянні двох рядків послідовно порівнюються коди символів, що стоять в однакових позиціях. Символи в рядках проглядаються зліва на право. Якщо в черговій парі виявляються різні символи, то більшим вважається той рядок, символ якого має більший код. На цьому порівняння припиняється.

Якщо порівнюються рядки різної довжини, причому один рядок збігається з початком іншого, то більшим буде більш довгий рядок.

Наприклад:

вираз	результат
'ABC' < 'XYZ'	true
'2' > '123'	true
'Іванов' < 'Іванов И.И.'	true

Символьний тип трактується як рядок одиничної довжини. Тому все сказане вище про стоки відноситься і до окремих символів. Більш докладне порівняння символів розглянуте в пункті 1.2.3.

Відзначимо також, що незалежно від того, який тип мають деякі логічні змінні x і y – Boolean, ByteBool, WordBool чи LongBool, якщо x дорівнює true, а y – false, то результатом виразу

x > y

буде true.

З простих операцій відношення можна побудувати складні логічні вирази з застосуванням до них як до операндів булевых операцій: not – заперечення, and – логічне И, or – логічне ЧИ і xor – логічне виключальне ЧИ. Слова not, and, or і xor, що позначають булевы операції, є зарезервованими словами мови програмування Object Pascal.

Результат застосування булевых операцій not, and, or і xor до операндів логічного типу true і false представлені у таблиці 1.1.

Результат застосування булевых операцій not, and, or і xor.

Таблиця 1.11.

X	Y	X and Y	X or Y	X xor Y	not X
false	false	false	false	false	true
false	true	false	true	true	true
true	false	false	true	true	false
true	true	true	true	false	false

Компілятор Delphi підтримує два режими виконання операцій and і or – повний і скорочений. У першому випадку логічний вираз завжди обчислюється до кінця, навіть якщо результат ясний заздалегідь. В другому випадку обчислення припиняються, як тільки стане очевидним кінцевий результат. Наприклад, якщо при обчисленні результату операції and з'ясувалося, що один операнд дорівнює false, те другий операнд можна не обчислювати, оскільки, очевидно, результатом усього виразу буде false. Директива компілятора {$B-}, що працює за замовчуванням, забезпечує скорочений режим обчислення, а директива {$B+} – повний.

При запису логічних виразів із застосуванням булевых операцій потрібно пам'ятати, що першою виконується операція not, потім and, потім or і xor. Операції відношення

виконуються в останню чергу. Тому при запису логічних виразів важливу роль грає правильне розміщення дужок.

Розглянемо приклад. Визначити порядок виконання і значення логічного виразу:

(-3>=5) or not (7<9) and (0<=3) .

Операції відношення, що стоять ліворуч і праворуч від знака булевой операції, повинні бути узяті в дужки, оскільки булевы операції мають більш високий пріоритет.
Позначимо цифрами порядок виконання операцій:

$$(-3 \overset{1}{>=} 5) \text{ or not } (7 \overset{2}{<} 9) \text{ and } (0 \overset{3}{<=} 3).$$

Результат виконання операцій:

1 – false, 2 – true, 3 – true, 4 – false, 5 – false, 6 – false.

Таким чином, значення даного виразу – false.

1.2.4.2. Логічні порозрядні операції

Логічні порозрядні операції призначені для порозрядної обробки цілочислених операндів, що представлені у двійковому виді

Логічні порозрядні операції.

Таблиця 1.12.

Операція	Назва	Типи операндів	Тип результату
not	порозрядне заперечення	цілий	цілий
and	порозрядне І	цілий	цілий
or	порозрядне ЧИ	цілий	цілий
xor	порозрядне виключальне ЧИ	цілий	цілий
shl	порозрядний зсув вліво	цілий	цілий
shr	порозрядний зсув вправо	цілий	цілий

Розглянемо дію логічних поразрядных операцій на прикладі одного з цілих типів – byte. Нагадаємо, що значення цього типу займають в оперативній пам'яті комп'ютера один байт і можуть змінюватися в діапазоні від 0 до 255. Відповідне представлення діапазону в двійковому виді має вид: 00000000...11111111.

Нехай змінні a і b типу byte мають значення 3 і 5 відповідно. При виконанні, наприклад, логічної порозрядної операції xor ці значення будуть представлені у двійковому виді: 00000011 і 00000101 відповідно. Операція xor буде застосовуватися порозрядно, тобто до кожної пари чисел, що стоять в однакових позиціях. Визначити результат можна за допомогою таблиці 1.11, якщо подумки замінити 0 словом false, а 1 – словом true. Отже, результат застосування порозрядної операції xor буде дорівнювати 00000110. Переводячи отримане значення в десяткову систему числення, одержимо остаточний результат – 6.

Нижче приведені приклади застосування логічних порозрядних операцій до зазначених вище змінних a і b.

Операція	Результат	Результат у двійковому представленні
not a	252	11111100
a and b	1	00000001
a or b	7	00000111
a xor b	6	00000110
a shl 1	6	00000110
a shr 1	1	00000001

Операції shl і shr зсувують значення змінної a уліво чи вправо на зазначену кількість битов (у прикладі – на 1 біт). При цьому початкові чи кінцеві біти губляться, а біти, що з'являються, містять нульові значення. Це еквівалентно множенню чи діленню на 2 у ступені, рівній кількості розрядів, на яку відбувається зсув.

Відзначимо, що застосування логічних порозрядних операцій до даних іншого цілого типу у загальному випадку приведе до інших результатів, оскільки різні типи мають різну довжину в байтах і, отже, різну кількість розрядів.

Тип результату при виконанні логічних порозрядних операцій визначається в такий спосіб.

Результат операцій not, shl і shr має той же тип, що і її операнд (у прикладах – змінна a).

Результат операцій and, or і xor буде мати найменший цілий тип, діапазон значень якого містить діапазони значень типів операндів.

1.2.5. Перелічуваний тип

Перелічуваний тип задає упорядковану множину значень шляхом перерахування імен, що позначають ці значення.

Наприклад:

```
type
    color = (black,green,yellow,blue,red,white);
    fam = (Ivanov,Petrov,Sidorov,Fedorov,Gavrilov);
```

При цьому кожне значення перелічуваного типу є ім'я. Числа, логічні і символьні константи не можуть бути значеннями перелічуваних типів.

Застосування перелічуваного типу підвищує наочність програми і дозволяє автоматично контролювати значення змінних. Так, наприклад, якщо в програмі є опис

```
var x:fam;
```

те це означає, що змінна x типу fam може приймати тільки одне з п'яти заданих значень.

Значення, вказані в опису перелічуваного типу, є константами цього типу. Це означає, що, маючи, наприклад, у програмі опис

```
var a,b:color; c,d:fam;
```

можна використовувати оператори присвоювання:

```
a := black;
b := green;
c := Ivanov;
d := Petrov;
```

При цьому варто пам'ятати, що не можна присвоювати змінної значення константи з іншого типу. Так, помилковими будуть наступні оператори:

```
a := Ivanov;
c := green;
```

Не можна також використовувати ту саму константу при описі двох різних перелічуваних типів, наприклад:

```
type
    day = (mon,tues,wed,thur,fri,sat,sun);
    free = (sat,sun);
```

Константи sat і sun використовуються при опису двох різних типів.

Виявляється, що відомий нам уже тип Boolean також може бути визначений як перелічуваний тип:

```
type
    Boolean = (false,true);
```

Звідси, зокрема, випливає, що між константами false і true виконується співвідношення

false < true .

Операції відношення <, >, <>, =, <=, >= застосовні до всіх простих типів, якщо, звичайно, порівнюються значення сумісних типів. Тому істинними будуть наступні відношення для констант типів color і fam:

black < green, green < yellow;
Ivanov < Petrov, Petrov < Sidorov .

Для аргументів перелічуваного типу визначені стандартні функції succ, pred і ord. При використанні функції ord варто враховувати, що порядковий номер першої з перерахованих у списку констант – нуль. Нижче приведені приклади використання цих функцій:

Вираз	Значення
succ(yellow)	blue
ord(black)	0
pred(Petrov)	Ivanov

У мові Object Pascal не можна безпосередньо вводити і виводити на зовнішні пристрої значення перелічуваних типів. Це обумовлено тим, що використовуються перелічувані типи насамперед для того, щоб зробити більш зрозумілим програмний код. Зокрема, багато типів у Delphi є перелічувані, що спрощує роботу з ними, оскільки дає можливість працювати не з абстрактними числами, а з осмисленими значеннями. Проілюструємо це на прикладі наступної програми.

Приклад 1.10.

Співробітники відділу мають наступну заробітну плату: Іванов і Петров – 6000р., Сидоров – 5000р., Федоров і Гаврилов – по 4000р. Скласти програму для обчислення сумарної зарплати співробітників відділу.

Рішення.

При складанні програми будемо використовувати перелічуваний тип fam, оператор циклу for (див. п.1.3.5.) і оператор вибору case (див. п.1.3.3.) .

Програма.

```
program p1_10;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type fam=(Ivanov,Petrov,Sidorov,Fedorov,Gavrilov);
var S:real; i:fam;
begin
  S := 0;
  for i:= Ivanov to Gavrilov do
    case i of
      Ivanov,Petrov: S := S+6000;
      Sidorov: S := S+5000;
      Fedorov,Gavrilov: S := S+4000
    end;
  writeln('Summa=',S:6:0);
  readln
end.
```

Результати роботи програми

```
Summa= 25000
```

1.2.6. Тип-діапазон

Тип-діапазон можна визначити, накладаючи обмеження на уже визначений деякий порядковий тип – його називають базовим типом. Обмеження визначається завданням діапазону, тобто мінімальної і максимальної константи в цьому діапазоні:

type < ім'я типу > = < мінімальна константа > .. < максимальна константа >;

Накладати обмеження на дійсний тип не допускається.

Наприклад:

```
type
        color = (white,read,blue,yellow,green,black); // перелічуваний тип
```

```
color1 = red..green; // обмеження на color
digit = 0..9; // обмеження на integer
symb = 'A'..'Z'; // обмеження на char
```

До змінних типу-діапазону застосовні всі операції і стандартні функції, що припустимі при роботі зі змінними відповідного базового порядкового типу.

Спроба присвоїти змінній типу-діапазону значення, що не входить у заданий діапазон, приведе до виникнення помилки на етапі виконання програми. Таким чином, у програміста з'являється додаткова можливість контролювати значення, що присвоюваються змінним. Для компілятора Object Pascal існує директива {$R+}, що дозволяє включати перевірку діапазону при присвоюванні значень змінної типу-діапазону. Виключити перевірку діапазону можна за допомогою директиви {$R-}.

Приклад 1.11.

Скласти програму для обчислення середнього бала, отриманого абітурієнтом на вступних екзаменах з математики, фізиці і хімії.

Рішення.

Будемо використовувати тип-діапазон для контролю оцінок, що вводяться.

Програма.

```
program p1_11;
{$APPTYPE CONSOLE}
{$R+}
uses
  SysUtils;
type ball = 2..5;
var math,phiz,chem:ball;
     ave:real;
begin
  writeln('Enter math,phiz,chem');
  readln( math,phiz,chem );
  ave := (math+phiz+chem)/3;
  writeln('ave =',ave:5:2);
  readln
end.
```

Результати роботи програми

```
Enter math,phiz,chem
4 3 5
ave = 4.00
```

1.2.7. Тип дата-час

1.2.7.1. Опис типу TDataTime

Тип дата-час призначений для одночасного збереження дати і часу. Визначається цей тип за допомогою стандартного ідентифікатора TDataTime. Значення цього типу являють собою 8-ми байтові дійсні числа з фіксованою крапкою. Ціла частина такого числа позначає кількість діб, що пройшли з 30 грудня 1899 року, а дробова – частину доби, що пройшли з 0 годин. Таким чином, ціла частина дозволяє визначити дату, а дробова – час.

Завдяки такому підходу, над даними типу дата-час визначені ті ж операції, що і для дійсних даних. З іншого боку, вимагаються додаткові зусилля при введенні і виведення дат і часу, щоб одержувати дані в звичному для нас виді. Так, наприклад, при введенні порядок дій може бути таким:

1. Дата і час представляються у виді символьного рядка і вводяться, як звичайно, процедурою readln. Поняття рядка буде докладно розглянуто в главі 1.5. Поки нагадаємо лише, що рядок – це довільний набір символів, узятий в апострофи, наприклад: 'Рядок'. Дата і час у виді рядка можуть бути представлені, наприклад, так:

'04.05.2000 15:05:26' ,

що означає – 4 травня 2000 року, 15 годин, 5 хвилин, 26 секунд.

2. Далі, за допомогою стандартної функції StrToDateTime варто перетворити символи рядка в дату і час.

При виведенні порядок дій зворотний. Для перетворення дати і часу в рядок можна використовувати функції DateTimeToStr, FormatDateTime чи процедуру DataTimeToString. Опис цих і інших підпрограм для роботи з типом дата-час приведений у пункті 1.2.7.2.

1.2.7.2. Процедури і функції для роботи з датами і часом

Нижче приведений опис основних стандартних підпрограм для роботи з датами і часом.

function Date: TDateTime;
Функція Date повертає поточну дату. Дозволяє одержати поточну дату як величину типу TDateTime.

function DateTimeToStr(DateTime: TDateTime): string;
Функція DateTimeToStr перетворює величину DateTime типу TDateTime у символьний рядок.

procedure DateTimeToString(var Result: string; const Format: string; DateTime: TDateTime);
Процедура DateTimeToString перетворює величину DateTime типу TDateTime у рядок символів Result, використовуючи рядок форматування Format. Опис символів-специфікаторів, що можуть входити в рядок Format, приведено нижче при описі функції FormatDateTime.

function DateToStr(Date: TDateTime): string;
Функція DateToStr перетворює величину Date типу TDateTime у рядок. При перетворенні використовується формат, заданий у глобальній змінній ShortDateFormat.

function DayOfWeek(Date: TDateTime): Integer;
Функція DayOfWeek повертає день тижня для аргументу Date типу TDateTime як ціле число, що приймає значення від 1 до 7, де 1 – це неділя, а 7 – субота.

procedure DecodeDate(Date: TDateTime; var Year, Month, Day: Word);
Процедура DecodeDate розбиває величину Date типу TDateTime на рік, місяць і день, значення яких заносяться у виді цілих чисел у параметри Year, Month і Day відповідно. Якщо значення аргументу Date менше нуля чи дорівнює нулю, то параметрам Year, Month і Day будуть присвоєні нульові значення.

procedure DecodeTime(Time: TDateTime; var Hour, Min, Sec, MSec: Word);
Процедура DecodeTime аналогічна процедурі DecodeDate, тільки аргумент Time розбивається на години, хвилини, секунди і миллисекунды.

function EncodeDate(Year, Month, Day: Word): TDateTime;
Функція EncodeDate виконує операцію, зворотню стосовно функції DecodeDate – поєднує рік, місяць і день, що містяться в параметрах Year, Month і Day відповідно, у значення типу TDateTime. Значення року повинне бути цілим числом, що знаходиться в діапазоні 1..9999. Місяць повинний приймати значення в діапазоні 1..12 . Значення дня повинне відповідати

PDF-XChange Click to buy NOW! www.docu-track.com

реально існуючому в календарі дню, тобто, наприклад, не можна в лютому задавати номер дня 30 чи 31. Більш того, якщо рік не є високосним, то не можна в лютому задавати номер дня 29.

function EncodeTime(Hour, Min, Sec, MSec: Word): TDateTime;
Функція EncodeTime подібно функції EncodeDate поєднує години, хвилини, секунди і мілісекунди, що містяться в параметрах Hour, Min, Sec і MSec відповідно, у значення типу TDateTime. Параметр Hour повинний приймати значення в діапазоні 0..23, параметри Min і Sec – у діапазоні 0..59, параметр MSec – у діапазоні 0..999 .Функція EncodeTime поверне дійсне число, розташоване між 0(включно) і 1(винятково), що буде представляти частину доби. Так, наприклад, 0 відповідає півночі, а 0.5 – півдню.

function FormatDateTime(const Format: string; DateTime: TDateTime): string;
Функція FormatDateTime перетворює величину DateTime типу TDateTime у рядок символів, використовуючи рядок форматування Format. У рядку Format можуть використовуватися наступні символи-специфікатори:

c – Відображає дату і час у виді дд.мм.гг. чч.мм.сс .Наприклад: 23.02.18 15.20.45 . Час не відображається, якщо дробова частина параметра DateTime дорівнює нулю.
d – Відображається день як число без ведучого нуля (1-31).
dd – Відображається день як число з ведучим нулем (01-31).
ddd – Відображається назва дня в скороченому виді. Наприклад: Sun, Sat – для нерусифікованих версій Windows, або Вс, Сб – для русифікованих версій Windows. Оскільки назви місяців повертаються в кодуванні ANSI, то в русифікованих версіях Windows використовувати цей специфікатор має сенс тільки для віконних застосувань.
dddd – Специфікатор, аналогічний попередньому, але на відміну від його назви днів відображаються цілком. Наприклад: Sunday, Saturday для нерусифікованих версій Windows, або – неділя, субота – для русифікованих версій Windows.
ddddd – Відображає дату у форматі дд.мм.гг. Наприклад: 23.10.99 .
dddddd – Відображає дату у форматі д місяць м. Наприклад: 23 October 1999 для нерусифікованих версій Windows, або 23 Октябрь 1999 – для русифікованих версій Windows. Оскільки назви місяців повертаються в кодуванні ANSI, то в русифікованих версіях Windows використовувати цей специфікатор має сенс тільки для віконних застосувань.
m – Відображається місяць як число без ведучого нуля (1-12). Якщо специфікатор m безпосередньо слідує за специфікаторами h чи hh, то відображається не місяць, а хвилини.
mm – Відображається місяць як число з ведучим нулем (01-12). Якщо специфікатор m безпосередньо випливає за специфікаторами h чи hh, то відображається не місяць, а хвилини.
mmm – Відображається назва місяця в скороченому виді. Наприклад: Jan, Dec – для нерусифікованих версій Windows, або янв, дек – для русифікованих версій Windows. Оскільки назви місяців повертаються в кодуванні ANSI, то в русифікованих версіях Windows використовувати цей специфікатор має сенс тільки для віконних застосувань.
mmmm – Відображається повна назва місяця. Наприклад: January, December – для нерусифікованих версій Windows, або Январь, Декабрь – для русифікованих версій.
yy – Відображається рік як число, що складається з двох цифр (00-99).
yyyy – Відображається рік як число, що складається з чотирьох цифр (0000-9999).
h – Відображаються години без ведучого нуля (0-23).
hh – Відображається година з ведучим нулем (00-23).
n – Відображаються хвилини без ведучого нуля (0-59).
nn – Відображаються хвилини з ведучим нулем (00-59).
s – Відображаються секунди без ведучого нуля (0-59).
ss – Відображаються секунди з ведучим нулем (00-59).
z – Відображаються мілісекунди без ведучого нуля (0-999).

PDF-XChange Click to buy NOW! www.docu-track.com

zzz – Відображаються мілісекунди з ведучим нулем (000-999).
t – Відображається час у форматі чч:мм . Наприклад: 15:53 .
tt – Відображається час у форматі чч:мм:сс. Наприклад: 15:53:24 .
am/pm – Відображається час у так називаному дванадцятигодинному форматі. Слово am означає, що вказується час до півдня, а слово pm означає, що вказується час після півдня. Наприклад, час 15:53:24 при використанні специфікаторів hh:mm am/pm буде відображено в такий спосіб: 05:53 pm.
a/p – Відображається час у дванадцятигодинному форматі. Буква a означає, що вказується час до півдня, а буква p означає, що вказується час після півдня. Наприклад, час 15:53:24 при використанні специфікаторів hh:mm a/p буде відображено в такий спосіб: 05:53 p.
ampm – Відображається час у дванадцятигодинному форматі, але без використання слів am і pm. Наприклад, час 15:53:24 при використанні специфікаторів hh:mm ampm буде відображено в такий спосіб: 05:53 .
/ – Відображається роздільник дати, використовуваний у Windows. Наприклад, дата 23 жовтня 1999 року при використанні специфікаторів dd/mm/yy буде відображена в такий спосіб: 23.10.99 .
: – Відображається роздільник часу, використовуваний у Windows. Наприклад, час 15 годин 53 хвилини 24 секунди при використанні специфікаторів hh:nn:ss буде відображено в такий спосіб: 15:53:24 .
'xx'/"xx" – Будь-які символи xx, відмінні від символів-специфікаторів, поміщені в рядок Format, відображаються на екрані дисплея без яких-небудь перетворень. Наприклад, якщо у віконному застосуванні рядок Format має вид: 'hh годин, nn хвилини, ss секунди' , то час 15:53:24 буде відображено в такий спосіб: 15 годин, 53 хвилини, 24 секунди .

Символи-специфікатори можуть бути записані як малими літерами, так і прописними – результат буде той самий.

Якщо рядок Format виявиться порожній, тобто не буде містити символів-специфікаторів, то в цьому випадку значення типу TDateTime буде форматуватися так, ніби був зазначений символ-специфікатор c.

function IsLeapYear(Year: Word): Boolean;
Функція IsLeapYear визначає, чи є зазначений рік Year високосним роком.

function Now: TDateTime;
Функція Now повертає поточну дату і час.

function StrToDate(const S: string): TDateTime;
Функція StrToDate перетворює рядок S у значення типу TDateTime
Рядок S повинний містити два чи три числа, розділених символами, визначеними в глобальної змінній DateSeparator у послідовності, визначеною глобальною змінною ShortDateFormat. У русифікованих версіях Windows роздільником звичайно є крапка, а послідовність має вид – день.місяць.рік. Якщо задані тільки два числа, вони сприймаються як місяць і день поточного року.
Рік, заданий у діапазоні 0..99 інтерпретується в залежності від значення глобальної змінної TwoDigitYearCenturyWindow. Якщо TwoDigitYearCenturyWindow дорівнює нулю, то рік належить поточному сторіччю. Якщо TwoDigitYearCenturyWindow більше нуля, то сторіччя визначається в такий спосіб. Нехай, наприклад, TwoDigitYearCenturyWindow дорівнює 50, а поточним є 2000 рік. Крапкою відліку стає рік 2000-50 = 1950. Це означає, що ми можемо використовувати роки з діапазону 1950..2049. Тоді, якщо рік заданий числом з діапазону 0..49, то в результаті одержимо рік з нового сторіччя, тобто в діапазоні 2000..2049. Якщо ж рік заданий числом з діапазону 50..99, то одержимо рік у поточному сторіччі, тобто в діапазоні 1950..1999

PDF-XChange
Click to buy NOW!
www.docu-track.com

function StrToDateTime(const S: string): TDateTime;
Функція StrToDateTime перетворює рядок у величину типу TDateTime.
Рядок повинний містити правильну дату і час. Для русифікованих Windows це означає, що параметр S повинний бути представлений у виді: 'день.місяць.рік година:хвилина:секунда'. Перетворення року з двозначного числа в чотиризначне здійснюється так само, як і для функції StrToDate.

function StrToTime(const S: string): TDateTime;
Функція StrToTime аналогічна функції StrToDateTime і перетворює рядок S, що містить для русифікованих Windows час у виді година:хвилина:секунда, у величину типу TDateTime.

function TimeToStr(Time: TDateTime): string;
Функція TimeToStr повертає рядок, у який перетвориться час, що міститься в параметрі Time.

Розглянемо простий приклад використання даних типу дата-час.
Приклад 1.12.
Визначити дату і день тижня, що наступлять через 10000 днів, 13 годин, 15 хвилин і 6 секунд після вашого моменту народження.
Рішення.
Як момент народження візьмемо 25 квітня 1973 року, 23 години, 35 хвилин і 22 секунди.
Програма.

```
program p1_12;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var bd:TDateTime; S:string;
    k:integer;
begin
  writeln('Enter bd');
  readln(S);
  bd := StrToDateTime(S);
  bd := bd+StrToTime('13:15:6')+10000;
  S := FormatDateTime('dd.mm.yyyy  tt',bd);
  k := DayOfWeek(bd);
  writeln('S=',S);
  writeln('k=',k);
  readln
end.
```

Результати роботи програми

```
Enter bd
25.04.1973 23:35:22
S=11.09.2000  12:50:28
k=2
```

1.3. Оператори

1.3.1. Складений оператор

Складеним оператором називається група операторів, узята в операторні дужки begin…end. Зарезервовані слова begin і end називаються операторними дужками. Оператори, що входять у складений, виконуються в порядку їхнього написання. Звідси, зокрема, випливає, що тіло будь-якої програми являє собою один складений оператор. Складений оператор використовується в тих випадках, коли за правилами мови Object Pascal дозволяється використовувати один оператор, а програмісту для рішення задачі необхідно виконати групу операторів. Найчастіше такі ситуації виникають при використанні складних операторів, наприклад, таких, як if, for, while і т.д. Наприклад:

```
if x > y then
  begin
    max := x; min := y
  end
else
  begin
    max := y; min := x
  end;
```

Крапка з комою призначена для поділу операторів, тобто не входить до складу оператора. Якщо поставити крапку з комою перед зарезервованим словом end, то це буде означати, що між крапкою з комою і словом end знаходиться порожній оператор. Порожній оператор не виконує ніяких дій, і його поява ніяким образом не вплине на хід виконання програми. Теоретично його можна позначити, тобто поставити перед ним мітку і використовувати його для передачі керування в програмі. Однак практично необхідність у цьому виникає вкрай рідко.

1.3.2. Умовний оператор IF

Оператор if призначений для реалізації розгалужного обчислювального процесу. У загальному виді оператор записується в такий спосіб:

if < логічний вираз > then < оператор1 > else < оператор2 >;

На початку оператор if обчислює значення логічного виразу. Якщо значення логічного виразу дорівнює true, то виконується оператор1, що стоїть за словом then. Якщо логічний вираз дорівнює false, то виконується оператор2, що стоїть за словом else.

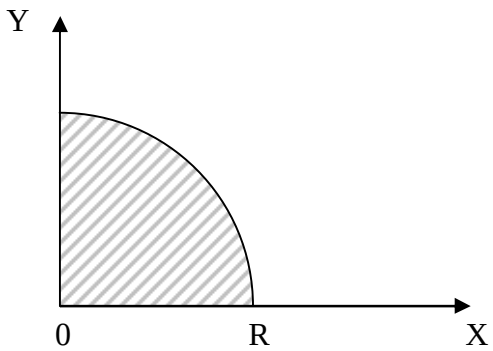
Якщо яка-небудь дія повинна бути виконана тільки при виконанні визначеної умови і пропущена у випадку невиконання цієї умови, то оператор if може бути записаний у скороченій формі:

if < логічний вираз > then < оператор1 > ;

Розглянемо приклади програм з використанням оператора if.

Приклад 1.13.

Скласти програму, що виводить на екран слово Yes, якщо точка з координатами (x,y) належить заштрихованій області, і No – у противному випадку.



Мал. 1.12. Задана область

Рішення.

Заштрихована область являє собою частину круга радіуса R, розташованого в першій чверті. Як відомо, точки (x,y), що лежать усередині круга з центром на початку координат і радіусом R, задовольняють нерівності $x^2 + y^2 \le R^2$. Крім того, точки, що лежать у першій чверті координатної площини, повинні задовольняти умовам: $x \ge 0$ і $y \ge 0$. Зважаючи на те, що точки, які лежать усередині заштрихованої області, повинні одночасно задовольняти всім трьом умовам, відповідний логічний вираз може бути записаний в такий спосіб:

(x*x+y*y<=R*R) and (x>=0) and (y>=0) .

Програма.

```
program p1_13;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var x,y,R:real;
begin
  writeln('Enter x,y,R');
  readln(x,y,R);
  if ( x*x+y*y<=R*R) and (x>=0) and (y>=0)
    then writeln('Yes')
    else writeln('No');
  readln
end.
```

Результати роботи програми

```
Enter x,y,R
1 1 2
Yes
```

Приклад 1.14.

Скласти програму для обчислення значення функції

$$y = \begin{cases} \dfrac{1}{2\sqrt{x}} & \text{, если } x \ge 1; \\ \dfrac{1}{3\sqrt[3]{x}} & \text{, если } 0 < x < 1; \\ \dfrac{1}{4\sqrt[4]{x}} & \text{, в остальных случаях..} \end{cases}$$

Рішення.

Для рішення цієї задачі можна скористатися вкладеними операторами if.

Програма.

```
program p1_14;
{$APPTYPE CONSOLE}
uses
  SysUtils,Math;
var x,y:real;
begin
  writeln('Enter x');
  readln(x);
  if x>=1 then y := 1/sqrt(x)/2
    else
      if (x>0) and (x<1) then y := 1/power(x,1/3)/3
```

```
        else
          y := 1/power(x,1/4)/4;
    writeln('y=',y);
    readln
end.
```

Результати роботи програми

```
Enter x
10
y= 1.58113883008419E-0001
```

Зверніть увагу, що перед else крапка з комою не ставиться.

При використанні вкладених конструкцій if можуть виникати неоднозначності в розумінні того, до якій із вкладених конструкцій if відноситься елемент else. Компілятор завжди вважає, що else відноситься до останньої з конструкцій if, у якій не було розділу else. Наприклад, у конструкції

if < умова1 > then if < умова2 > then < оператор1 > else < оператор2 >;

else буде віднесено компілятором до другої конструкції if, тобто оператор2 буде виконуватися у випадку, якщо перша умова істинна, а друга хибна.

Якщо ж ви хочете віднести else до першого if, це треба записати в явному виді за допомогою операторных дужок begin…end:

if<умова1 > then begin if < умова2 > then < оператор1 > end else < оператор2 >;

Відзначимо також, що обчислити y з умови приклада 1.14 можна і з використанням скороченої форми оператора if:

```
y := 1/power(x,1/4)/4;
if x>=1 then y := 1/sqrt(x)/2;
if (x>0) and (x<1) then y := 1/power(x,1/3)/3;
```

Приклад 1.15.

Скласти фрагмент програми для обчислення значень функції відповідно до графіка:

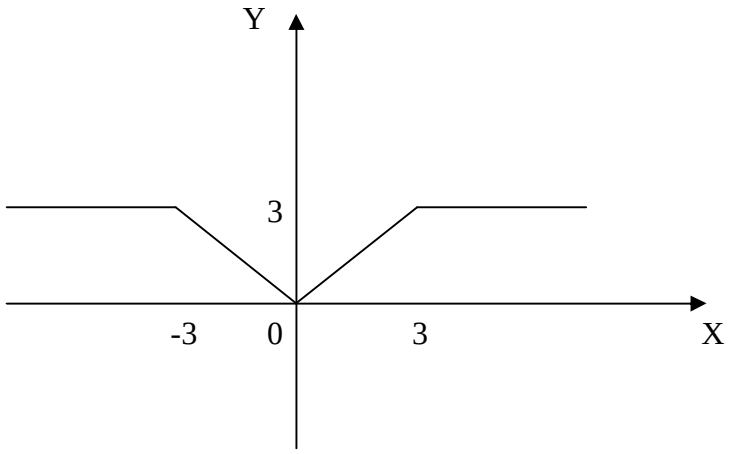


Рис 1.13. Функція, задана графічно.

Рішення.

Перш ніж написати фрагмент програми, необхідно записати аналітичний вираз для даної функції. Отже

$$y = \begin{cases} 3 & , якщо \quad |x| \geq 3; \\ |x| & , якщо \quad -3 < x < 3. \end{cases}$$

Тоді фрагмент програми буде мати наступний вид:

```
if abs(x) >= 3 then y := 3 else y := abs(x);
```

Приклад 1.16.

Скласти програму для обчислення коренів квадратного рівняння

$$ax^2 + bx + c = 0 \qquad (a \neq 0)$$

в області дійсних чисел.

Рішення.

Ведемо позначення: a, b, c – коефіцієнти рівняння; d – дискримінант; x1 і x2 – корені рівняння.

Програма.

```
program p1_16;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var a,b,c,d,x1,x2:real;
begin
  writeln('Enter a,b,c');
  readln(a,b,c);
  d := sqr(b)-4*a*c;
  if d<0 then writeln('d<0')
    else
      if d=0 then
         begin
         x1 := -b/(2*a);
         x2 := x1
         end
        else
         begin
         x1 := (-b+sqrt(d))/(2*a);
         x2 := (-b-sqrt(d))/(2*a);
         end;
  writeln('x1=',x1:5:2,'  x2=',x2:5:2);
  readln
end.
```

Результати роботи програми

```
Enter a,b,c
1 -4 4
x1= 2.00  x2= 2.00
```

Нагадаємо, що якщо після зарезервованих слів then чи else необхідно використовувати не один, а групу операторів, то треба використовувати складений оператор, тобто група операторів береться в операторні дужки begin...end .

Приклад 1.17.

Дано три дійсних числа a, b і c. Скласти програму для визначення максимального з цих чисел.

Рішення.

Скористаємося наступним алгоритмом. На початку знайдемо найбільше з двох значень, наприклад, з a і b. Знайдене найбільше значення помістимо в змінну max. Потім знайдемо найбільше значення, що міститься в змінних max і c.

Програма.

```
program p1_17;
{$APPTYPE CONSOLE}
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
uses
  SysUtils;
var  a,b,c,max:integer;
begin
  writeln('Enter a,b,c');
  readln(a,b,c);
  if a>b then max := a else max := b;
  if c>max then max := c;
  writeln('max= ',max);
  readln
end.
```

Результати роботи програми

```
Enter a,b,c
3 6 2
max= 6
```

1.3.3 Оператор вибору CASE

На відміну від оператора if, оператор case дозволяє вибрати і виконати один оператор не з двох, а з декількох операторів. У загальному випадку оператор case може бути записаний у такий спосіб:

```
case < вираз > of
< список _міток_1 > : < оператор1 >;
< список _міток_2 > : < оператор2 >;
…….....
< список _міток_N > : < оператор >;
else
< оператор >
end;
```

У приведеному форматі оператора < вираз > – це вираз порядкового типу (див. пункт 1.2.1), що визначає подальший хід виконання програми. Використання порядкового типу означає, що не можна використовувати вирази, що повертають дійсні числа чи рядки. Список міток являє собою список констант, розділених комами. Якщо константи представляють діапазон чисел, то замість списку можна вказати першу й останню константи діапазону, розділені двома крапками. Наприклад, список 1,2,3,4 можна записати як 1..4.

Виконується оператор case у такий спосіб.

На початку обчислюється значення виразу, що стоїть за словом case. Далі отримане значення послідовно порівнюється з константами зі списків міток, що стоять перед двокрапками.

Якщо значення виразу збігається з константою з якого-небудь списку міток, то виконується відповідний цьому списку міток оператор. На цьому виконання оператора case завершується. Помітимо, що як оператор може бути використаний і складений оператор.

Якщо значення виразу не збігається з жодною константою з усіх списків міток, то виконується оператор, що стоїть за зарезервованим словом else.

Розділ else не обов'язково повинний бути присутнім в операторі case. Якщо він відсутній і значення виразу не збіглося з жодною константою зі списків міток, то жоден оператор не буде виконаний.

Приклад 1.18.

Скласти програму, що виводить на екран дисплея назва пори року, у залежності від уведеного номера місяця.

Рішення.

При рішенні даної задачі оператор case може бути записаний різними способами. На початку розглянемо найпростіший випадок, коли списки міток являє собою безпосереднє перерахування номерів місяців.

Програма.

```
program p1_18;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var n:integer; s:string;
begin
  writeln('Enter n');
  readln(n);
  case n of
    1,2,12  : s := 'Winter';
    3,4,5   : s := 'Spring';
    6,7,8   : s := 'Summer';
    9,10,11 : s := 'Autumn'
  end;
  writeln(s);
  readln
end.
```

Результати роботи програми

```
Enter n
3
Spring
```

Для даної задачі в операторі case можуть бути використані діапазони:

```
case n of
  1,2,12  : s := 'Winter';
  3..5    : s := 'Spring';
  6..8    : s := 'Summer';
  9..11   : s := 'Autumn'
end;
```

При використанні розділу else оператор case прийме наступний вид:

```
case n of
  3..5    : s := 'Spring';
  6..8    : s := 'Summer';
  9..11   : s := 'Autumn';
  else      s := 'Winter';
end;
```

1.3.4. Оператор переходу GOTO

Оператор переходу goto – це простий оператор, що передає керування в програмі на інший оператор, перед яким стоїть мітка, зазначена в операторі goto.

У загальному виді оператор записується в такий спосіб:

goto < мітка >;

Мітка в Object Pascal – це довільний ідентифікатор або ціле число без знака. Будь-яка мітка повинна з'явитися в розділі опису міток, перш ніж вона зустрінеться в тілі програми. У розділі операторів мітка може стояти тільки перед одним оператором і відокремлюється від нього двокрапкою.

Не можна передавати керування усередину складних операторів (if, case, оператори циклів), а також із визивної програми в підпрограму.

PDF-XChange Click to buy NOW! www.docu-track.com

Приклад 1.19.

Скласти програму для обчислення суми $\sum_{i=1}^{\infty}\frac{1}{i^2}$ з точністю до 0.001.

Рішення.

У програмі використовується алгоритм нагромадження суми. Процес є циклічним і припиняється, як тільки черговий доданок стане менше 0.001.

Програма.

```
program p1_19;
{$APPTYPE CONSOLE}
uses
  SysUtils;
label lb1,lb2;
var i:integer; s:real;
begin
    s := 0;
    i := 1;
lb1:s := s+1/sqr(i);
    i := i+1;
    if 1/sqr(i) < 0.001 then goto lb2 else goto lb1;
lb2:writeln('s=',s);
    readln
end.
```

Результати роботи програми

```
s= 1.61319070032792E+0000
```

Оператор переходу варто використовувати у виняткових ситуаціях, оскільки він утруднює розуміння програми, робить її заплутаною і складною у налагоджуванні. Використання операторів if, case і операторів циклу дозволяє реалізувати будь-який тип обчислювального процесу.

1.3.5. Оператори циклів

Часто при рішенні задач необхідно багаторазово виконувати визначену послідовність дій. Такі повторювані дії називаються циклами. У мові Object Pascal можлива організація 3-х видів циклів:

1. З параметром (оператор for).
2. З передумовою (оператор while).
3. З післяумовою (оператор repeat).

1.3.5.1. Оператор FOR

Оператор циклу for організує виконання послідовності операторів заздалегідь відоме число раз. Існують два варіанти оператора.

Перший варіант (зі збільшенням лічильника):

for < лічильник > := < початкове значення > to < кінцеве значення > do < оператор >;

Другий варіант (зі зменшенням лічильника):

for < лічильник > := < початкове значення > downto < кінцеве значення > do < оператор >;

де

< лічильник > – змінна порядкового типу (див. пункт 1.2.1.) – параметр циклу;

< початкове значення > і < кінцеве значення > – вирази, що повинні бути сумісними для присвоювання з параметром циклу;

< оператор > – це оператор, що виконується в циклі й утворює так називане тіло циклу.

Якщо в циклі необхідно виконати групу операторів, то її варто перетворити в один складений оператор, узяв в операторні дужки begin…end.

Оператор for діє в такий спосіб.

На початку обчислюються початкове і кінцеве значення лічильника. Далі лічильнику присвоюється початкове значення. Потім значення лічильника порівнюється з кінцевим значенням. Далі, поки лічильник менше чи дорівнює кінцевому значенню (у першому варіанті) або більше чи дорівнює кінцевому значенню (у другому варіанті), виконується чергова ітерація циклу. У противному випадку відбувається вихід з циклу.

Виконання чергової ітерації містить у собі спочатку виконання тіла циклу, а потім присвоєння лічильнику (параметру циклу) наступного більшого значення (у першому варіанті) чи наступного меншого значення (у другому варіанті). Якщо параметр циклу цілого типу, то це означає збільшення або зменшення його на 1.

Якщо в першому варіанті початкове значення більше кінцевого чи в другому варіанті – менше кінцевого, то оператор (тіло циклу) не виконається жодного разу.

Таким чином, тіло циклу виконується < кінцеве значення > - < початкове значення > + 1 раз для першого варіанта оператора for i < початкове значення > - < кінцеве значення > + 1 раз для другого варіанта.

Приклад 1.20.

Обчислити $\sum_{i=1}^{n}\frac{1}{i^2}=1+\frac{1}{4}+\frac{1}{9}+...+\frac{1}{n^2}.$

Рішення.

Для обчислення суми будемо використовувати так називаний алгоритм нагромадження, який полягає в тім, що для обчислення суми n доданків необхідно n раз виконати оператор:

S := S+ i-ий доданок;

де i = 1, 2, 3,…,n; i-ий доданок дорівнює $1/i^2$. Якщо зазначений оператор виконується перший раз, тобто i = 1, то S повинна містити 0.

Програма.

```
program p1_20;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var i,n:integer; s,ai:real;
begin
  write('Enter n=');
  readln(n);
  s := 0;
  for i := 1 to n do
    begin
    ai := 1/sqr(i);
    s := s+ai
    end;
  writeln('s=',s:7:3);
  readln
end.
```

Результати роботи програми

```
Enter n=6
s=  1.491
```

Приклад 1.21.

Обчислити $\sum_{i=1}^{n}(-1)^i \frac{x}{i} = -x + \frac{x}{2} - \frac{x}{3} + ... + (-1)^n \frac{x}{n}.$

Рішення.

Даний ряд називається знакоперемінним. Знаки доданків утворюють послідовність: -,+,-,+,... . Уведемо позначення: znak – коефіцієнт $(-1)^i$, ai – поточний (i-тий) доданок, s – шукана сума. Тоді для обчислення суми можна скористатися оператором:

s := s+ai;

де

ai := znak*x/i;

i = 1,2,3,…,n.

Якщо відоме значення змінної znak на i-тому кроці, то на i+1 кроці її значення буде

znak := -znak;

Початкове значення змінної znak дорівнює –1, початкове значення змінної s дорівнює 0.

Програма.

```
program p1_21;
{$APPTYPE CONSOLE}
uses
,  SysUtils;
var i,n,znak:integer;
    s,ai,x:real;
begin
  write('Enter n=');
  readln(n);
  write('Enter x=');
  readln(x);
  s := 0;
  znak := -1;
  for i:= 1 to n do
    begin
    ai := znak*x/i;
    s := s+ai;
    znak := -znak;
    end;
  writeln('s=',s:7:3);
  readln
end.
```

Результати роботи програми

```
Enter n=10
Enter x=2
s= -1.291
```

Приклад 1.22.

Обчислити $\sum_{k=1}^{n} \frac{x}{k!} = x + \frac{x}{2!} + \frac{x}{3!} + ... + \frac{x}{n!}.$

Рішення.

Знайдемо рекурентну формулу для обчислення a_k – k-того доданка суми. Відомо, що k! = 1*2*3*…*k . Тоді

при k=1 $a_1 = x/(1!) = x/1;$
при k=2 $a_2 = x/(2!) = x/(1*2) = a_1/2;$
при k=3 $a_3 = x/(3!) = x/(1*2*3) = a_2/3;$
……
при k=n $a_n = x/(n!) = x/(1*2*3*\ldots*n)=a_{n-1}/n.$
У такий спосіб $a_k = a_{k-1}/k$, k=1,2,3,…,n; $a_0=x$.
Програма.

```
program p1_22;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var k,n:integer;
    s,a,x:real;
begin
  write('Enter n=');
  readln(n);
  write('Enter x=');
  readln(x);
  s := 0;
  a := x;
  for k := 1 to n do
    begin
    a := a/k;
    s := s+a;
    end;
  writeln('s=',s:7:3);
  readln
end.
```

Результати роботи програми

```
Enter n=10
Enter x=2
s=  3.437
```

Приклад 1.23.

Обчислити $\prod_{i=1}^{n}\left(1+\frac{x^i}{2^i}\right)=\left(1+\frac{x}{2}\right)\cdot\left(1+\frac{x^2}{2^2}\right)\cdot\left(1+\frac{x^3}{2^3}\right)\cdot\ldots\cdot\left(1+\frac{x^n}{2^n}\right).$

Рішення.

Обчислення скінченого добутку здійснюється аналогічно обчисленню скінченої суми. Позначимо: p – добуток, що обчислюється, ai – i-тий співмножник. Щоб обчислити добуток n співмножників, скористаємося правилом:

p := p*ai;

де $ai = 1+x^i/2^i$, i=1,2,3,…,n . Початкове значення змінної p дорівнює 1.
Програма.

```
program p1n23;
{$APPTYPE CONSOLE}
uses
  SysUtils,Math;
var i,n:integer;
    p,a,x:real;
begin
```

```
  write('Enter n=');
  readln(n);
  write('Enter x=');
  readln(x);
  p := 1;
  for i:=1 to n do
    begin
    a := 1+power(x/2,i);
    p := p*a
    end;
  writeln('p=',p);
  readln
end.
```

Результати роботи програми

```
Enter n=12
Enter x=3
p= 2.92551407080879E+0014
```

Приклад 1.24.

Обчислити $C_n^m = \frac{n!}{m!(n-m)!}$.

Рішення.

C_n^m – це число сполучень з n елементів по m елементів. Щоб знайти число сполучень, потрібно обчислити n!, m!, (n-m)! . Як відомо факторіал обчислюється по формулі:

k! = 1*2*3*...*k .

Зважаючи на те, що факторіал є частковим випадком добутку, ми можемо записати:

$$k! = 1*2*3*\ldots*k=\prod_{i=1}^{k} i \ .$$

Позначимо: c – число сполучень з n елементів по m; fact1, fact2, fact3 – значення факторіалів n!, m! і (n-m)! відповідно.

Програма.

```
program p1_24;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var i,n,m,c:integer;
    fact1,fact2,fact3:real;
begin
  write('Enter n=');
  readln(n);
  write('Enter m=');
  readln(m);
  fact1 := 1;
  for i := 2 to n do
    fact1 := fact1*i;
  fact2 := 1;
  for i := 2 to m do
    fact2 := fact2*i;
  fact3 := 1;
```

```
  for i := 2 to n-m do
    fact3 := fact3*i;
  c := round(fact1/(fact2*fact3));
  writeln('c=',c:5);
  readln
end.
```

Результати роботи програми

```
Enter n=10
Enter m=3
c=  120
```

Помітимо, що для обчислення факторіалів використовувалися змінні fact1, fact2, fact3, що мають тип real, хоча по визначенню факторіал є цілим числом. Це зроблено для того, щоб уникнути переповнення, оскільки тип real має набагато більш широкий діапазон значень.

Функція round використовується для рішення зворотної задачі – приведення результату дійсного виразу до цілого типу.

Приклад 1.25.

Обчислити $a^{-1} \cdot a^{\frac{1}{2}} \cdot a^{-\frac{1}{3}} \cdot \ldots \cdot a^{\frac{(-1)^n}{n}}$.

Рішення.

Визначимо формулу для обчислення загального члена добутку b_i. Очевидно, що

$$b_i = \begin{cases} \dfrac{1}{a^{\frac{1}{i}}}, & \text{якщо } i \text{ не кратне } 2; \\ a^{\frac{1}{i}}, & \text{якщо } i \text{ кратне } 2; \\ i = 1,2,3,\ldots,n. \end{cases}$$

Для перевірки кратності зручно використовувати операцію mod, що дозволяє визначити залишок при діленні цілих чисел.

Програма.

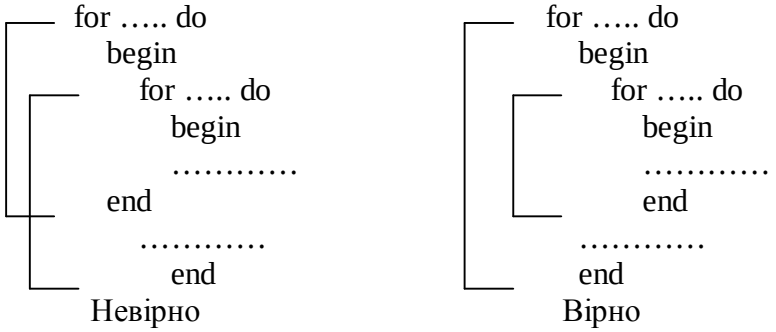
```
program p1_25;
{$APPTYPE CONSOLE}
uses
  SysUtils,Math;
var n,i:integer;
    p,a,b:real;
begin
  write('Enter n=');
  readln(n);
  write('Enter a=');
  readln(a);
  p := 1/n;
  for i := 2 to n do
    begin
    if i mod 2 = 0 then b := power(a,1/i)
      else b := 1/power(a,1/i);
    p := p*b
    end;
  writeln('p=',p:7:4);
```

```
  readln
end.
```

Результати роботи програми

```
Enter n=10
Enter a=3
p= 0.1476
```

Серед операторів, що повторюються під час виконання циклу, можуть зустрічатися інші оператори циклів. Такі цикли називаються вкладеними. Якщо усередині одного оператора циклу міститься інший, то перший називають зовнішнім циклом, а другий – внутрішнім. Внутрішній цикл повинний цілком розміщатися усередині зовнішнього і ні в якому разі не повинний з ним перетинатися (див. мал. 1.14).



Мал. 1.14. Правильне і неправильне вкладення циклів

Розглянемо приклад з використанням вкладених циклів.

Приклад 1.26.

Вивести на екран дисплея таблицю Пифагора.

Рішення.

Таблиця Пифагора має 10 рядків і 10 стовпців. Для формування рядків будемо використовувати зовнішній цикл з параметром i, а для формування значень усередині рядка – внутрішній цикл з параметром j.

Програма.

```
program p1_26;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var s,i,j:integer;
begin
  for i := 1 to 10 do
    begin
    for j := 1 to 10 do
      begin
      s:= i*j;
      write(s:4)
      end;
    writeln
    end;
  readln
end.
```

Результати роботи програми

1	2	3	4	5	6	7	8	9	10
2	4	6	8	10	12	14	16	18	20
3	6	9	12	15	18	21	24	27	30
4	8	12	16	20	24	28	32	36	40
5	10	15	20	25	30	35	40	45	50
6	12	18	24	30	36	42	48	54	60
7	14	21	28	35	42	49	56	63	70
8	16	24	32	40	48	56	64	72	80
9	18	27	36	45	54	63	72	81	90
10	20	30	40	50	60	70	80	90	100

1.3.5.2. Оператор циклу з передумовою WHILE

Оператор while має наступний формат:

while < логічний вираз > do < оператор >;

Виконується оператор у такий спосіб. Спочатку обчислюється значення логічного виразу. Якщо це значення дорівнює true, то виконується оператор, що стоїть після слова do. Після цього керування знову передається на початок оператора, знову обчислюється значення логічного виразу і процес повторюється. Цикл припиняється, якщо при черговому обчисленні логічного виразу його значення виявиться рівним false. Якщо при першому обчисленні значення логічного виразу виявиться, що воно дорівнює false, то оператор циклу не виконається жодного разу.

Відзначимо також наступне.

Оскільки логічний вираз обчислюється при виконанні кожної ітерації, його варто робити по можливості більш простим.

Якщо в циклі потрібно виконати не один оператор, а декілька, то їх варто узяти в операторні дужки begin...end, тобто використовувати складений оператор.

Приклад 1.27.

Визначити число членів нескінченного ряду

$$\sum_{n=0}^{\infty}\frac{1}{2^n}=1+\frac{1}{2}+\frac{1}{4}+....$$

при якому його сума обчислюється з похибкою $\varepsilon = 10^{-5}$.

Рішення.

Уведемо позначення: s – сума, що накопичується, t – черговий доданок, n – номер поточного члена ряду, eps – похибка.

Програма.

```
program p1_27;
{$APPTYPE CONSOLE}
uses
  SysUtils,Math;
var eps,s,t:real;
    n:integer;
begin
  writeln('Enter eps');
  readln(eps);
  s := 0; t := 1; n := 0;
  while t>eps do
    begin
    s := s+t;
    n := n+1;
    t := power(2,-n);
    end;
```

```
  writeln('s=',s:7:3);
  writeln('number of members of row n=',n:3);
  readln
end.
```

Результати роботи програми

```
Enter eps
0.00001
s=  2.000
number of members of row n= 17
```

Приклад 1.28.

Обчислити з похибкою ε = 10^{-4} значення функції y=cos(x), використовуючи розкладання в ряд:

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots = \sum_{n=0}^{\infty} (-1)^n \frac{x^{2n}}{(2n)!}.$$

Рішення.

При рішенні подібних задач доцільно використовувати рекурентну формулу для одержання чергового члена ряду. Цю формулу можна одержати в такий спосіб.

Запишемо вираз для загального члена ряду:

$$t_n = (-1)^n \frac{x^{2n}}{(2n)!}.$$

Підставимо в цю формулу замість n величину n-1:

$$t_{n-1} = (-1)^{n-1} \frac{x^{2(n-1)}}{[2(n-1)]!}.$$

Знайдемо відношення:

$$\frac{t_n}{t_{n-1}} = \frac{(-1)^n x^{2n}(2n-2)!}{(2n)!(-1)^{n-1} x^{2n-2}} = \frac{-x^2(2n-2)!}{(2n)!} = \frac{-x^2 \cdot 1 \cdot 2 \cdot \ldots \cdot (2n-2)}{1 \cdot 2 \cdot \ldots \cdot (2n-2)(2n-1)2n} = -\frac{x^2}{(2n-1)2n}.$$

Відкіля одержуємо шукану рекурентну формулу:

$$t_n = -\frac{x^2}{(2n-1)2n} t_{n-1}.$$

Для перевірки коректності роботи алгоритму і програми поряд зі значенням суми ряду, що міститься в змінної s, будемо виводити на екран дисплея також точне значення cos(x), що знаходиться в змінної y.

Програма.

```
program p1_28;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var eps,s,t,f,x,y:real;
    n:integer;
begin
  writeln('Enter x,eps');
  readln(x,eps);
  s := 0; t := 1; n := 0;
  while abs(t)>eps do
    begin
    s := s+t;
    n := n+1;
```

```
    f := -sqr(x)/(2*n*(2*n-1));
    t := t*f;
    end;
  y := cos(x);
  writeln('row amount s=',s:8:5);
  writeln('proper value y=',y:8:5);
  readln
end.
```

Результати роботи програми

```
Enter x,eps
4.25 0.000001
row amount s=-0.44609
proper value y=-0.44609
```

1.3.5.3. Оператор циклу з післяумовою REPEAT

Оператор repeat має наступний формат:

repeat < оператор1 >; < оператор2 >; ...; < операторN > until < логічний вираз >;

Виконується оператор repeat у такий спосіб. На початку виконується група операторів – оператор1, оператор2,..., операторN. Потім обчислюється значення логічного виразу. Якщо воно дорівнює true, то цикл припиняється, у противному випадку виконується наступна ітерація циклу.

Оскільки умова припинення циклу перевіряється наприкінці циклу, то оператори, що утворять тіло циклу, виконуються принаймні один раз.

На відміну від операторів for і while, синтаксис яких передбачає виконання в циклі тільки одного оператора, у тіло циклу оператора repeat може входити довільна кількість операторів. Це означає, що використання операторных дужок begin...end в операторі repeat не обов'язково, хоча їхня присутність не буде помилкою.

Приклад 1.29.

Визначити число членів нескінченного ряду

$$\sum_{n=0}^{\infty}\frac{1}{2^n}=1+\frac{1}{2}+\frac{1}{4}+....$$

при якому його сума обчислюється з похибкою $\varepsilon = 10^{-5}$.

Рішення.

Ця задача була вирішена нами з використанням оператора while у прикладі 1.27. Покажемо, що її можна вирішити і з використанням оператора циклу з післяумовою. Нагадаємо, що: s – сума, що накопичується, t – черговий доданок, n – номер поточного члена ряду, eps – похибка.

Програма.

```
program p1_29;
{$APPTYPE CONSOLE}
uses
  SysUtils,Math;
var eps,s,t:real;
    n:integer;
begin
  writeln('Enter eps');
  readln(eps);
  s := 0; t := 1; n := 0;
  repeat
    s := s+t;
```

```
    n := n+1;
    t := power(2,-n);
  until t<eps;
  writeln('s=',s:7:3);
  writeln('number of members of row n=',n:3);
  readln
end.
```

Результати роботи програми

```
Enter eps
0.00001
s=  2.000
number of members of row n= 17
```

Даний приклад дозволяє зробити висновок про те, що для організації циклу при розв'язанні однієї і теж задачі можуть бути використані різні оператори циклу. З іншого боку, розуміння і використання наявних у цих операторів відмінностей, дозволяє вибрати найбільш ефективний шлях реалізації алгоритму рішення задачі.

Приклад 1.30.

Знайти суму ряду

$$\sum_{i=0}^{\infty}(-1)^i\frac{i}{2i+1}=1-\frac{1}{3}+\frac{1}{5}-\frac{1}{7}+...;$$

с похибкою ε=0.001.

Рішення.

Для обчислення коефіцієнта

$$(-1)^n=\begin{cases}1, & якщо\ n\ кратне\ 2,\\ -1, & якщо\ n\ не\ кратне\ 2,\end{cases}$$

крім функції power, можна використовувати стандартну функцію odd(x), що повертає true, якщо аргумент x – непарний і false, якщо x – парний.

Програма.

```
program p1_30;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var eps,s,t:real;
    n:integer;
begin
  writeln('Enter eps');
  readln(eps);
  s := 0; t := 1; n := 0;
  repeat
    if odd(n) then s := s-t else s :=s+t;
    n := n+1;
    t := 1/(2*n+1);
  until t<eps;
  writeln('s=',s:7:3);
  writeln('number of members of row n=',n:3);
  readln
end.
```

Результати роботи програми

```
Enter eps
0.001
s=  0.785
number of members of row n=500
```

1.4. Процедури і функції

Дуже часто процес рішення якої-небудь задачі може бути подумки представлений як послідовність рішення більш простих підзадач. Наприклад, задача обчислення C_n^m – числа сполучень з n елементів по m елементів, розглянута в прикладі1.24, – може бути представлена як послідовність підзадач обчислення n!, m! і (n-m)! . Якщо програма в цілому призначена для рішення всієї задачі, то процедури і функції призначені для рішення окремих підзадач.

Процедури і функції по своїй структурі подібні звичайним програмам і мають загальну назву – підпрограми. Застосування підпрограм дає можливість зменшити число повторень однієї і тієї ж послідовності операторів, а також конструювати програму, як набір окремих підпрограм. Для складних задач це істотно спрощує процес програмування.

У програмі опис процедур і функцій звичайно розташовується між розділами змінних і операторів. Хоча, як відомо, Object Pascal допускає довільну послідовність опису констант, змінних, типів, міток і підпрограм. Але при цьому необхідно пам'ятати основне правило Object Pascal: будь-яке ім'я в програмі повинне бути обов'язково описане перед тим, як воно з'явиться серед операторів, що виконуються. Кожна процедура чи функція визначається тільки один раз, але може використовуватися багаторазово.

У процедурах і функціях, як і в звичайних програмах, можуть бути описані власні мітки, константи, типи, власні змінні і навіть власні процедури і функції.

1.4.1. Опис процедури. Оператор процедури

Опис кожної процедури починається з заголовка, у якому задається ім'я процедури і список формальних параметрів із зазначенням їхніх типів; процедура може бути і без параметрів, тоді в заголовку вказується тільки її ім'я. За допомогою параметрів здійснюється передача вихідних даних у процедуру, а також передача результатів роботи назад у програму, що викликає її.

Загальна форма запису заголовка процедури:

procedure < ім'я процедури > (< список формальних параметрів >);

Список формальних параметрів може містити в собі параметри-значення, параметри-змінні, перед якими повинне стояти зарезервоване слово var, і деякі інші категорії формальних параметрів.

Виклик і виконання процедури здійснюється за допомогою оператора процедури:

< ім'я процедури > (< список фактичних параметрів >);

Між формальними і фактичними параметрами повинна бути повна відповідність, тобто формальних і фактичних параметрів повинна бути однакова кількість; порядок слідування фактичних і формальних параметрів повинний бути однаковий; тип кожного фактичного параметра повинний збігатися з типом відповідного йому формального параметра.

При виклику процедури спочатку передаються параметри, при цьому параметри-значення передаються за значенням, а параметри-змінні – за посиланням. Основна відмінність цих способів передачі параметрів полягає в наступному.

Присвоювання значень параметру-змінній усередині процедури одночасно виконується і для відповідного аргументу (фактичного параметра). Це зв'язано з тим, що параметр-змінна містить не саме передане значення, а його адресу, за якою віно розташовується в оперативній пам'яті комп'ютера. Маючи у своєму розпорядженні адресу, процедура може довільно змінювати дані, що знаходяться за цією адресою. Таким чином,

параметри, у які записуються результати роботи процедури, повинні передаватися тільки за посиланням.

Параметри-значення містять копії переданих даних. При завершенні роботи процедури, ці копії знищуються. Отже, параметри-значення можна використовувати тільки для передачі в процедуру вихідних даних.

Приклад 1.31.

Обчислити $u = \dfrac{\max(a, a+b) + \max(a, b+c)}{1 + \max(a+bc, ac)}$.

Рішення.

Для обчислення значення змінної u необхідно три рази знайти максимальне з двох чисел. Це означає, що в рішенні загальної задачі необхідно виділити підзадачу пошуку максимального з двох чисел і оформити алгоритм її рішення у виді процедури. Назвемо процедуру max. Вона буде мати три параметри: два вхідних – x, y і один вихідний – z. Отже, заголовок процедури буде мати вид:

```
procedure max(x,y:real; var z:real);
```

Склавши заголовок процедури, можна записати алгоритм обчислення результату:

```
procedure max(x,y:real; var z:real);
begin
  if x>y then z := x else z := y
end;
```

Таким чином, ми склали опис процедури. Для того щоб скористатися процедурою, її потрібно викликати з основної програми. За умовою задачі нам потрібно викликати процедуру три рази:

```
max(a,a+b,t1);
max(a,b+c,t2);
max(a+b*c,a*c,t3);
```

При виклику підпрограми указуються фактичні параметри. Помітимо, що як фактичний параметр, переданий за значенням, може використовуватися довільний вираз відповідного типу, а як фактичний параметр, переданий за посиланням – тільки змінна відповідного типу.

Програма.

```
program p1_31;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var a,b,c,t1,t2,t3,u:real;
procedure max(x,y:real; var z:real);
begin
  if x>y then z := x else z := y
end;
begin
  writeln('Enter a,b,c');
  readln(a,b,c);
  max(a,a+b,t1);
  max(a,b+c,t2);
  max(a+b*c,a*c,t3);
  u := (t1+t2)/(1+t3);
  writeln('u=',u:7:3);
  readln
end.
```

Результати роботи програми

```
Enter a,b,c
2 5 7
u=  0.500
```

1.4.2. Категорії формальних параметрів

Вище вже були названі дві категорії формальних параметрів – параметри-значення і параметри-змінні. Крім них у Delphi 6 є ще параметри-константи і так називані вихідні параметри.

Розглянемо відмінності між цими категоріями.

Як було вже сказано, параметр-значення передається за значенням. Це означає, що в момент звертання до підпрограми відбувається наступне:

1. Обчислюється значення відповідного фактичного параметра. Уживання слова «обчислюється» означає, що в якості відповідного фактичного параметра можуть бути не тільки константи і змінні, але і вирази.
2. Отримане значення копіюється в тимчасову пам'ять – так називаний стік. Відтіля його й одержує підпрограма.
3. У підпрограмі копія фактичного параметра присвоюється параметру-значенню. Зміна параметра-значення усередині підпрограми ніяк не позначається на значенні відповідного фактичного параметра. Саме тому параметри-значення варто використовувати для передачі вхідних даних у підпрограму.

Параметр-змінна передається за посиланням. Це означає, що в підпрограму передається адреса в оперативній пам'яті, за якою розташовується значення відповідного фактичного параметра. Звідси випливає:

1. Знаючи адресу, підпрограма може витягти значення фактичного параметра, змінити його і розмістити за зазначеною адресою своє значення.
2. Як фактичне значення для параметра-змінної може бути використана тільки змінна. Таким чином, параметри-змінні варто використовувати для повернення в визивну програму результатів роботи підпрограми.

При використанні параметра-константи перед ним у заголовку процедури чи функції повинне стояти зарезервоване слово const, наприклад:

```
procedure sum(const x:real; y:integer; var z:real);
```

Параметр-константа передається подібно параметру-змінній за посиланням, але між ними існує принципова відмінність: значення параметра-константи не можна змінювати усередині підпрограми. При спробі змінити значення параметра-константи компілятор видасть повідомлення про помилку. Отже, параметри-константи треба використовувати в тому випадку, коли ми хочемо бути упевненими, що фактичний параметр не буде змінений при виклику підпрограми.

При описі вихідного параметра перед ним у заголовку процедури варто ставити зарезервоване слово out, наприклад:

```
procedure abc(x,y:real; out z:real);
```

Вихідний параметр подібний параметру-змінній. Але, якщо параметр-змінна може бути використана як для передачі в процедуру вхідних даних, так і для повернення результату, то вихідний параметр варто використовувати тільки для повернення результату, оскільки пам'ять, займана відповідним фактичним параметром, у момент звертання до процедури очищається.

1.4.3. Опис функції. Вказівник функції.

Опис функції подібний до опису процедури, однак існують деякі відмінності. Найважливіше з них полягає в тім, що результатом роботи функції є одне значення довільного типу. Тип результату задається в заголовку функції, загальний вид якого:

function < ім'я функції > (< список формальних параметрів >): < тип результату >;

Серед вхідних у функцію операторів повинний обов'язково бути присутнім хоча б один оператор присвоювання, у лівій частині якого стоїть ім'я стандартної змінної result і яка трактується як результат, що повертається функцією. Цей оператор і визначає значення, вироблюване функцією.

Виклик і виконання функції здійснюються при обчисленні значення вказівника функції

< ім'я функції > (< список фактичних параметрів >),

який входить у деякий вираз у визивній програмі. При виклику функції передача фактичних параметрів відбувається так само, як і при виклику процедури.

Приклад 1.32

Для того щоб краще усвідомити різницю між процедурою і функцією, складемо програму з використанням функції за умовою приклада 1.31:

обчислити $u = \dfrac{\max(a, a+b) + \max(a, b+c)}{1 + \max(a+bc, ac)}$.

Рішення.

При рішенні даної задачі доцільно використовувати функцію для обчислення максимального з двох чисел. Використання функції є обґрунтованим, тому що результатом обчислення максимуму буде одне значення простого типу.

Програма.

```
program p1_32;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var a,b,c,u:real;
function max(x,y:real):real;
begin
  if x>y then result := x else result := y
end;
begin
  writeln('Enter a,b,c');
  readln(a,b,c);
  u := (max(a,a+b)+max(a,b+c))/(1+max(a+b*c,a*c));
  writeln('u=',u:7:3);
  readln
end.
```

Результати роботи програми

```
Enter a,b,c
2 5 7
u=  0.500
```

У даному прикладі вказівниками функції є max(a,a+b), max(a,b+c) і max(a+b*c,a*c).

1.4.4. Глобальні і локальні змінні

Підпрограми можуть повертати результат в основну програму не тільки за допомогою параметрів-змінних і вихідних параметрів, але і безпосередньо змінюючи глобальні змінні. Змінні, описані в основній програмі, є глобальними стосовно внутрішніх процедур і функцій. Це означає, що вони доступні в підпрограмах, описаних усередині основної програми. Змінні, описані усередині процедур і функцій, називаються локальними. Вони породжуються при кожнім вході в підпрограму і знищуються при виході з цієї підпрограми, тобто локальні змінні існують тільки при виконанні підпрограми і недоступні в основній програмі.

Приклад 1.33.

Обчислити $f = \sum_{i=1}^{10}(\sin^{2t}\alpha + \cos\alpha) + \sum_{i=5}^{20}(\sin^{2t}\beta + \cos\beta).$

Рішення.

З загальної задачі вичленуємо підзадачу обчислення суми $s(i0, ik, x) = \sum_{i=i0}^{ik}(\sin^{2t} x + \cos x)$, що залежить від трьох параметрів – *i*0, *ik* і *x*. Оформимо рішення цієї підзадачи у виді процедури. Якби ми вирішили скористатися механізмом параметрів для обміну даними між основною програмою і процедурою, то наша процедура мала б чотири параметри: три для передачі вхідних даних – *i*0, *ik*, *x* і один для повернення результату. У програмі, приведеній нижче, для обміну даними між основною програмою і процедурою використовується механізм глобальних параметрів. Тому процедура s узагалі не має параметрів.

Програма.

```
program p1_33;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var i0,ik:integer; alfa,beta,x,z,z1,z2,f:real;
procedure s;
var i:integer; sum:real;
begin
  sum := 0;
  for i := i0 to ik do
    sum := sum+exp(i*ln(sqr(sin(x))))+cos(x);
  z := sum
end;
begin
  writeln('Enter alfa,beta');
  readln(alfa,beta);
  i0 := 1; ik := 10; x := alfa;
  s;
  z1 := z;
  i0 := 5; ik := 20; x := beta;
  s;
  z2 := z;
  f := z1+z2;
  writeln('f=',f);
  readln
end.
```

Результати роботи програми

```
Enter alfa,beta
2 3
f=-1.59398644626015E+0001
```

У цьому прикладі глобальними є змінні: i0, ik; alfa ,beta ,x ,z ,z1 ,z2 ,f. Локальними є змінні: i, sum.

Помітимо, що якщо ім'я локальної змінної збігається з ім'ям глобальної змінний, то при виконанні підпрограми така глобальна змінна стає недоступної, тобто «закривається» локальною змінною.

PDF-XChange
Click to buy NOW!
www.docu-track.com

1.4.5. Параметри, що мають значення за замовчуванням

У Delphi 6 існують параметри, значення яких можна не вказувати при виклику процедури чи функції. Їх називають параметрами зі значеннями за замовчуванням, чи, коротше, замовчуваними параметрами. У списку параметрів у загальному виді вони описуються в такий спосіб

< ім'я параметра >:< тип > = < значення > .

Замовчувані параметри повинні знаходиться наприкінці списку формальних параметрів. Якщо при виклику процедури чи функції фактичний параметр, який відповідає замовчуваному параметру не зазначений, то в підпрограму передається задане для замов чуваного параметра < значення >. Якщо в підпрограмі є кілька замовчуваних параметрів і для частини з них задані фактичні значення, а для частини – ні, то останні повинні розташовуватися наприкінці списку формальних параметрів. Іншими словами, при виклику підпрограми фактичні параметри повинні розташовуватися підряд, без пропусків.

Приклад 1.34.

Скласти програму для знаходження максимального значення для позитивних чисел, кількість яких не перевищує 5.

Рішення.

У процедурі для пошуку максимального значення опишемо п'ять параметрів, що замовчуються. Як значення за замовчуванням приймемо 0.

Програма.

```
program p1_34;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var x,y,z,w,v,m:real;
procedure p(var max:real; a:real=0; b:real=0; c:real=0;
                    d:real=0; e:real=0);
begin
  max := a;
  if max<b then max := b;
  if max<c then max := c;
  if max<d then max := d;
  if max<e then max := e;
end;
begin
  writeln('Enter x,y,z,w,v');
  readln(x,y,z,w,v);
  p(m,x,y);
  writeln('max=',m:3:0);
  p(m,x,y,z);
  writeln('max=',m:3:0);
  p(m,x,y,z,w);
  writeln('max=',m:3:0);
  p(m,x,y,z,w,v);
  writeln('max=',m:3:0);
  readln
end.
```

Результати роботи програми

```
Enter x,y,z,w,v
2 4 6 8 10
max=  4
max=  6
max=  8
max= 10
```

1.4.6. Перезавантаження функцій

У Object Pascal мається можливість використовувати підпрограми з однаковими іменами, але які відрізняються кількістю і типом параметрів. У заголовках таких підпрограм повинно бути зазначене зарезервоване слово overload. При виклику таких підпрограм компілятор аналізує кількість і тип фактичних параметрів і викликає відповідну їм підпрограму.

Приклад 1.35.

Скласти програму для знаходження мінімального з двох чисел і мінімального з двох символів.

Рішення.

У програмі будемо використовувати дві функцій з ім'ям min. Перша для знаходження мінімального числа, друга – для визначення мінімального з двох символів.

Програма.

```
program p1_35;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var x,y:real; s,v:char;
function min(a,b:real):real;overload;
begin
  if a>b then result := b else result := a
end;
function min(a,b:char):char;overload;
begin
  result := a;
  if a>b then result := b;
end;
begin
  writeln('Enter two numbers');
  readln(x,y);
  writeln('min=',min(x,y):3:0);
  writeln('Enter two characters');
  readln(s,v);
  writeln('min=',min(s,v));
  readln
end.
```

Результати роботи програми

```
Enter two numbers
3 6
min=  3
Enter two characters
az
min=a
```

При звертанні до функції min(x,y) викликається функція , визначена для дійсних аргументів, у при звертанні min(s,v) – для символьних аргументів.

1.4.7. Процедурні типи

При вживанні слова «тип» у нас мимоволі виникає асоціація зі словом «дані» і маються на увазі дані цілих, дійсних і інших типів.

Виявляється, що процедури і функції теж є даними і, отже, у програмі можуть існувати змінні процедурних типів. Змінним процедурних типів як значення присвоюваються імена відповідних підпрограм.

Цей підхід широко використовується в Object Pascal, особливо при використанні класів (див. частину 2). При створенні консольних застосувань процедурний тип використовується для передачі процедур і функцій як параметрів з визивної програми у викликувану підпрограму.

Існують два процедурних типи: тип-процедура і тип-функція. Для того щоб оголосити процедурний тип, необхідно в розділі type вказати ім'я типу, знак «=» і заголовок процедури без імені:

type < ім'я процедурного типу > = < заголовок процедури чи функції без імені >;

Наприклад:

```
type
  func1 = function  (a,b:real):real;
  func2 = function  (const x:integer):char;
  p1 = procedure  (x,y:integer; var z:real);
  p2 = procedure;
```

Для ілюстрації сказаного розглянемо приклад програми.

Приклад 1.36.

Використовуючи метод трапецій знайти $\int_0^{\pi} \sin^2 x dx$ і $\int_0^{\pi} \cos 3x dx$.

Рішення.

Відрізок [0, π] разобъём на 10 рівних частин, тобто крок інтегрування буде h=0.1π . Відповідно до методу трапецій наближене значення інтеграла $\int_0^{\pi} f(x)dx$ буде обчислюватися за формулою

$$\int_0^{\pi} f(x)dx = h(\frac{f(x_0)+f(x_{10})}{2} + f(x_1)+f(x_2)+...+f(x_9)) ,$$

де x_0, x_1, x_2,…,x_{10} – вузли інтегрування, що обчислюються за формулою x_i = i·h, i = 0,1,2,…,10.

Програма.

```
program p1_36;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type
  myfunc = function (x:real):real;
var a,b,s1,s2:real;
function f1(x:real):real;
begin
  result := sin(x)*sin(x)
end;
function f2(x:real):real;
begin
```

```
  result := cos(3*x)
end;
procedure trap(const a,b:real; f:myfunc; var s:real);
var x,h:real; i:integer;
begin
  h := (b-a)/10;
  s := (f(a)+f(b))/2;
  for i := 1 to 9 do
    begin
    x := i*h;
    s := s+f(x)
    end;
  s := s*h
end;
begin
  a :=0;
  b := pi;
  trap(a,b,f1,s1);
  writeln('s1=',s1:7:3);
  trap(a,b,f2,s2);
  writeln('s2=',s2:7:3);
  readln
end.
```

Результати роботи програми

```
s1=  1.571
s2=  0.000
```

У розділі type програма містить опис типу-функції – myfunc. Цей тип використовується при описі параметра-функції f у процедурі trap. При першому звертанні до процедури trap як фактичний параметр для f використовується ім'я функції f1, при другому звертанні – ім'я функції f2.

1.5. Рядки

1.5.1. Рядкові типи

Рядок являє собою набір символів, узятий в апострофи (одинарні лапки). Наприклад,

'abcde' , 'Іванов И.И.' , '10.0' , 'S=' .

Константа типу char являє собою символьний рядок одиничної довжини.

Використовувані в Object Pascal типи рядків приведені в таблиці 1.13.

Типи рядків

Таблиця 1.12.

Тип рядка	Максимальна довжина	Чи є нульовий символ наприкінці
ShortString	255 байт	немає
AnsiString	2 Гб	є
String	255 байт / 2 Гб	немає / є
WideString	2 Гб	є

Значення типу ShortString – це так називані короткі рядки, довжина яких не перевищує 255 символів. Кожен символ займає один байт, найперший байт містить число, що вказує довжину рядка. Кожен байт має свій порядковий номер. Перший байт, що містить

довжину рядка, має номер, рівний 0. По номері символу можна одержати доступ до його значення.

Приклад1.37.

У рядку 'Ivanov I.' дописати повністю ім'я.

Рішення.

Вихідний рядок має довжину рівну 9. Збільшимо спочатку цю довжину до 12, а потім помістимо в рядок відсутні символи.

Програма.

```
program p1_37;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var s:shortstring;
begin
  s := 'Ivanov I.';
  s[0] := chr(12);
  s[9] := 'v';
  s[10] := 'a';
  s[11] := 'n';
  s[12] := '.';
  writeln('s=',s);
  readln
end.
```

Результати роботи програми

```
s=Ivanov Ivan.
```

Помітимо, що довжина рядка, що зберігається в нульовому байті, представлена в символьному виді. Тому для перетворення в число варто користатися стандартною функцією ord, а назад – функцією chr.

Короткий рядок розміщається компілятором у пам'яті комп'ютера до початку виконання програми, тобто статично.

Рядок типу AnsiString розташовується в пам'яті інакше. Сама змінна типу AnsiString займає в пам'яті 4 байти і є вказівником, тобто містить адресу тієї комірки пам'яті, починаючи з який буде фактично розташовуватися символьний рядок. Виділення місця в пам'яті відбувається на етапі виконання програми, тобто динамічно. Програма сама визначає необхідну довжину рядка по заданій кількості символів, і операційна система виділяє потрібну ділянку пам'яті. Наприкінці рядка розміщається термінальний (завершальний) нуль – символ #0 і так називаний лічильник посилань, що займає 4 байти. Нумерація символів у рядку типу AnsiString починається з одиниці, тобто перший символ такого рядка має індекс 1.

Лічильник посилань дозволяє заощаджувати пам'ять. Наприклад, якщо в програмі мається фрагмент:

```
……
var s1,s2:ansistring;
……
s1 := 'stroka';
s2 := s1;
……
```

то пам'ять для розміщення змінної s2 не виділяється, а в змінну s2 міститься значення вказівника зі змінної s1. Лічильник посилань в області пам'яті, зв'язаною зі змінною s1,

збільшує своє значення на 1 і стане рівним 2. У загальному випадку в програмі може бути декілька змінних, що посилаються на один і той самий рядок. Лічильник посилань дорівнює кількості змінних, що посилаються на рядок. Якщо одна з змінних, що посилаються на рядок типу AnsiString, змінить своє значення, то в пам'яті буде виділене місце для нового рядка. Число посилань у попередньому рядку зменшується на 1, а в новому рядку стає рівним 1. Якщо число посилань на рядок стане рівним 0, то рядок знищується і звільняє місце в пам'яті.

Приклад 1.38.

Складемо програму за умовою попереднього приклада, використовуючи рядок типу AnsiString.

Рішення.

Для установки довжини рядка будемо використовувати стандартну процедуру SetLength.

Програма.

```
program p1_38;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var s:ansistring;
begin
  s := 'Ivanov I.';
  setlength(s,12);
  s[9] := 'v';
  s[10] := 'a';
  s[11] := 'n';
  s[12] := '.';
  writeln('s=',s);
  readln
end.
```

Результати роботи програми

```
s=Ivanov Ivan.
```

Тип String інтерпретується компілятором Object Pascal по різному, у залежності від значення директиви компілятора $H. Якщо вона включена – {$H+} – то тип String інтерпретується як AnsiString, якщо немає – {$H-} – то як ShortString. За замовчуванням діє директива {$H+}.

Якщо в розділі type зазначено, наприклад, String[10], то незалежно від директив компілятора тип трактується як ShortString із зазначеним числом символів.

Тип WideString також являє собою динамічно розташовувані в пам'яті комп'ютера рядки, довжина яких обмежена тільки обсягом вільної пам'яті комп'ютера. Однак на відміну від рядка типу AnsiString кожен символ є Unicode-символом, тобто кодується 2 байтами. Тому в рядку типу WideString у порівнянні з рядком типу AnsiString, що має таку ж довжину, кількість символів удвічі менше.

1.5.2. Стандартні підпрограми для рядків.

Нижче приведені основні стандартні процедури і функції для роботи з рядками типу String.

```
function AnsiCompareStr(const S1, S2: string): Integer;
```

Порівнює два рядки S1 і S2 у кодуванні ANSI з урахуванням регістра. Повертає значення менше 0, якщо S1 < S2, 0, якщо S1 = S2, і більше 0, якщо S1 > S2. У русифікованих версіях Windows може бути застосована до рядків, що містять російські букви.

function AnsiCompareText(const S1, S2: string): Integer;
Функція AnsiCompareText цілком аналогічна попередньої функції AnsiCompareStr за винятком того, що порівняння символів здійснюється без урахування регістра.

function AnsiLowerCase(const S: string): string;
Повертає в кодуванні ANSI рядок S, перетворений до нижнього регістра. У русифікованих версіях Windows може бути застосована до рядків, що містять російські букви.

function AnsiPos(const Substr, S: string): Integer;
Повертає позицію (індекс) першого входження Substr у S. Якщо Substr немає в S, повертається 0. У русифікованих версіях Windows може бути застосована до рядків, що містять російські букви.

function AnsiUpperCase(const S: string): string;
Повертає рядок S у кодуванні ANSI, перетворений до верхнього регістра. У русифікованих версіях Windows може бути застосована до рядків, що містять російські букви.

function Concat(s1 [, s2,..., sn]: string): string;
Повертає рядок, що представляє собою зчеплення з рядків s1, ..., sn. Ідентична операції "+" для рядків.

function Copy(S; Index, Count: Integer): string;
Параметр S – це рядок типу string чи динамічний масив. Функція Copy повертає підрядок рядка S, що починається із символу S[Index] і вміщує Count символів.

procedure Delete(var S: string; Index, Count:Integer);
Видаляє з S підрядок, що починається із символу S[Index] і вміщує Count символів.

procedure Insert(Source: string; var S: string; Index: Integer);
Уставляє рядок Source у S, починаючи із символу за номером Index.

function Length(S): Integer;
Повертає число символів у рядку S.

function Pos(Substr: string; S: string): Integer;
Повертає позицію (індекс) першого входження підрядку Substr у рядок S. Якщо Substr немає в S, повертається 0.

procedure SetLength(var S; NewLength: Integer);
Параметр S є рядком чи динамічним масивом. Процедура SetLength установлює нову довжину NewLength рядка S. Якщо рядок має тип ShortString, то значення параметра NewLength повинне знаходиться в діапазоні 0..255 . Для довгих рядків значення параметра NewLength обмежено лише розмірами доступної пам'яті комп'ютера. При збільшенні довжини рядка старі значення, що знаходилися там, зберігаються, а в доданих позиціях знаходяться невизначені значення.

function StringOfChar(Ch: Char; Count: Integer): string;
Створює рядок, що складається з Count раз повторюваного символу Ch.

function Trim(const S: string): string;
Видаляє з рядка S початкові і завершальні пробіли і керуючі символи.

function TrimLeft(const S: string): string;
Видаляє з рядка S початкові пробіли і керуючі символи.

function TrimRight(const S: string): string;
Видаляє з рядка S завершальні пробіли і керуючі символи.

1.5.3. Рядкові вирази

Під рядковим будемо розуміти вираз, результатом якого є символьний рядок. Нагадаємо, що символьні значення можна трактувати як рядки одиничної довжини. Крім операцій відношення (див. пункт 1.2.4.1.), над рядками визначена операція зчеплення (конкатенації), що позначається знаком плюс – «+». Це бінарна операція. Її результатом є послідовність символів першого операнда, після якої розташовується послідовність символів другого операнда. Операндами можуть бути рядки, масиви типу char (див. пункт 1.6.1.) чи символи. Наприклад:

Вираз	Результат
'Object'+' Pascal'	'Object Pascal'
'Петров'+' П.П'+'.'	'Петров П.П.'

Рядки різних типів можуть змішуватися в одному виразі, змінним одного рядкового типу можна присвоювати значення іншого рядкового типу. Компілятор при цьому здійснює автоматичне приведення типів. Якщо змінній типу ShortString присвоюється як значення рядок, довжина якого перевищує довжину змінної, то рядок усікається до довжини змінної.

Дуже часто при обробці символьних рядків використовуються стандартні підпрограми (див. пункт 1.5.2.). Розглянемо приклад програми, у якому демонструється використання деяких з цих підпрограм.

Приклад 1.39.

Скласти програму, що виконує наступні дії:

1. Уводить із клавіатури рядок 'Bloomed apples and pears' (Розцвітали яблуні і груші).
2. Визначає довжину рядка.
3. Виділяє з вихідного рядка підрядки 'apples'(яблуні) і 'pears'(груші).
4. Видаляє з вихідного рядка підрядок 'apples and '(яблуні і).
5. Уставляє вилучений підрядок на колишнє місце.
6. Визначає номер позиції, у якій знаходиться буква r у вихідному рядку.

Рішення.

Для рішення цієї задачі нам будуть потрібні стандартні функції length, copy, pos і процедури delete і insert.

Програма.

```
program p1_39;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var a,b,c,d:string;
begin
  writeln('Enter string');
  readln(a);
  writeln('length(a) = ',length(a));
  b := copy(a,9,6);
  writeln('copy(a,9,6) = ',b);
  c := copy(a,20,5);
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
  writeln('copy(a,20,5) = ',c);
  d := copy(a,9,11);
  delete(a,9,11);
  writeln('after delete a = ',a);
  insert(d,a,9);
  writeln('after insert a = ',a);
  writeln('pos("r",a) = ',pos('r',a));
  readln
end.
```

Результати роботи програми.

```
Enter string
Bloomed apples and pears
length(a) = 24
copy(a,9,6) = apples
copy(a,20,5) = pears
after delete a = Bloomed pears
after insert a = Bloomed apples and pears
pos("r",a) = 23
```

Слід зазначити, що для введення символьних рядків необхідно використовувати процедуру readln, а не read. Це порозумівається тим, що наприкінці кожного символьного рядка, що вводиться з клавіатури, стоїть так називаний роздільник рядків EOLN – послідовність кодів #13 (CR) і #10 (LF). Роздільник рядків уставляється в текст, що вводиться, при натисканні на клавішу Enter. У результаті кожен символьний рядок відображається в окремому рядку дисплея. Процедура Read може зчитувати дані тільки до символу EOLN. Дані, розташовані в наступній символьному рядку, тобто за роздільником рядків EOLN, для неї не доступні. Процедура readln зчитує всі символи, розташовані до роздільника рядків, а потім і сам роздільник рядків. Оскільки символи #13 (переведення каретки) і #10 (перехід у початок рядка) є керуючими, то в результаті їхнього зчитування курсор дисплея переходить у початок наступної рядка. Розташований в новому рядку дисплея символьний рядок може бути зчитаний наступною процедурою readln. Зі сказаного випливає, зокрема, що, якщо змінна x має тип string, те процедура

```
readln(x);
```

еквівалентна послідовності процедур

```
read(x);
readln;
```

1.5.4. Стандартні підпрограми перетворення рядків у числові типи і назад

Нижче приведені функції і процедури для перетворення рядків у числові типи і назад. Помітимо, що для консольних застосувань ця задача не актуальна, оскільки процедури read, readln, write, writeln виконують ці перетворення автоматично (див. пункт 1.1.12.2). Зате при створенні віконних застосувань програміст повинний сам виконати перетворення типів, використовуючи стандартні підпрограми. Таким чином, основна область застосування підпрограм перетворення рядків у числові типи і назад – це віконні застосування, хоча ніщо не заважає використовувати ці підпрограми в консольних застосуваннях.

```
function IntToStr(Value: Integer): string; overload;
function IntToStr(Value: Int64): string; overload;
```

У Delphi 5 мається дві перезавантажувані функції IntToStr –для типу integer і для типу int64. Обидві функції повертають рядок, що містить перетворене ціле значення Value.

function FloatToStr(Value: Extended): string;
Перетворює дійсне значення Value у символьний рядок.

function FloatToStr(Value: Extended; Format: TFloatFormat; Precision, Digits: Integer): string;
Функція FloatToStr призначена для перетворення дійсного числа в його рядкове представлення і відрізняється від функції FloatToStr тим, що дозволяє одержувати зображення числа в різних форматах. Параметри функції означають наступне:

Value – число, рядкове представлення якого треба одержати;

Format – іменована константа, що визначає вид рядка;

Precision – параметр, що визначає точність представлення перетворюваного числа. Значення Precision повинне бути не більш 7 для типу Single, не більш 15 для типу Double і не більш 18 для типу Extended.

Digits – значення цього параметра залежить від використовуваного формату.

Значеннями параметра Format можуть бути наступні іменовані константи:

ffGeneral – загальний цифровий формат. Число зображується у форматі з десятковою крапкою (ffFixed), якщо кількість цифр ліворуч від крапки менше чи дорівнює значенню, заданому в параметрі Precision, або число більше чи дорівнює 0.00001. У противному випадку число зображується в науковому форматі.

ffExponent – науковий формат. Число зображується у виді ±d.ddd...E±ddd , де ± – знак «плюс» чи «мінус», d – десяткова цифра. Загальна кількість десяткових цифр мантиси, включаючи число перед десятковою крапкою, дорівнює значенню, що міститься в параметрі Precision. Параметр Digits задає мінімальну кількість цифр представлення експоненти (0..4) .

ffFixed – формат з десятковою крапкою. Число представляється у форматі з фіксованою крапкою: ±ddd.ddd... . Параметр Precision задає загальну кількість десяткових цифр у представленні числа. Перед десятковою крапкою завжди знаходиться як мінімум одна цифра. Параметр Digits задає кількість цифр після десяткової крапки (0..18) . Якщо кількість цифр цілої частини більше, ніж Precision, то число зображується в науковому форматі.

ffNumber – числовий формат. Числовий формат схожий на формат з десятковою крапкою (ffFixed), тільки в зображенні числа використовується роздільник груп розрядів. Наприклад, число 83574921,34 буде зображено в такий спосіб: 83 574 921,34 .

ffCurrency – грошовий формат. Подібний числовому формату, але наприкінці рядка ставиться символ грошової одиниці. Роздільник цілої і дробової частин, роздільник груп розрядів, а також зображення грошової одиниці, використовувані в зображенні числа в грошовому форматі, визначаються настройкою Windows. Для їхнього завдання в русифікованій версії Windows варто відкрити вікно *панели управления*, потім вікно *Свойства:Язык и стандарты* і перейти на вкладки *Числа* або *Денежная единица*.

procedure Str(X [: Width [: Decimals]]; var S);
Перетворює ціле чи дійсне значення X у рядок S. Не обов'язкові параметри: Width - ширина поля, що відводиться для виведення всього числа, Decimals - число цифр, розташованих після десяткової крапки. Параметр Decimals використовується тільки для дійсних чисел.

function StrToInt(const S: string): Integer;
Перетворює рядок S у ціле число. Рядок не повинний містити ведучих чи відомих пробілів.

function StrToFloat(const S: string): Extended;
Перетворює рядок S у дійсне число. Рядок не повинний містити ведучих чи відомих пробілів.

procedure Val(S; var V; var Code: Integer);
Перетворює рядок S у цілу чи дійсну змінну V, у залежності від типу цієї змінної. Параметр Code містить нуль, якщо перетворення пройшло успішно, у противному випадку він містить

номер позиції в рядку S, де виявлений помилковий символ. В останньому випадку перетворення не виконується. Рядок S може містити ведучі чи відомі пробіли.

Приклад 1.40.

Використовуючи функцію FloatToStr, вивести на екран дисплея значення змінної x = 123456789.12345678 у форматах ffExponent, ffFixed, ffGeneral, ffNumber і ffCurrency.

Рішення.

При використанні форматів ffNumber і ffCurrency у консольному застосуванні варто виконати настроювання Windows так, щоб символи, використовувані для поділу груп і позначення грошової одиниці, мали однакове кодування в таблицях кодів ANSI і ASCII. Наприклад, для позначення грошової одиниці краще використовувати не російську букву «р», а англійську «p», а як пробіл краще використовувати символ з кодом 32, а не 162, що використовується в Windows за замовчуванням. Для введення символу з кодом 32 потрібно натиснути клавішу Alt і на додатковій клавіатурі набрати число 32.

Програма.

```
program p1_40;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var x:real;
begin
  x := 123456789.12345678;
  writeln(FloatToStrF(x,ffExponent,6,2));
  writeln(FloatToStrF(x,ffFixed,17,8));
  writeln(FloatToStrF(x,ffGeneral,17,8));
  writeln(FloatToStrF(x,ffNumber,17,8));
  writeln(FloatToStrF(x,ffCurrency,17,8));
  readln
end.
```

Результати роботи програми.

```
1,23457E+08
123456789,12345678
123456789,12345678
123 456 789,12345678
123 456 789,12345678р.
```

1.6. Структуровані типи

До структурованих типів відносяться масиви, записи, множини і файли. Структуровані типи утворюються з інших типів, які, у свою чергу, можуть бути простими або структурованими. Це означає, що може мати місце вкладеність типів.

1.6.1. Масиви

У Delphi 6 масиви можуть бути статичними, тобто мати фіксоване число елементів, або динамічними, коли кількість елементів задається в процесі виконання програми.

1.6.1.1. Статичні масиви

Статичний тип-масив (далі просто – тип-масив) являє собою фіксовану кількість упорядкованих однотипних компонентів (елементів), які мають індекси. Він може бути одновимірним чи багатовимірним.

Щоб задати тип-масив, використовується зарезервоване слово array, після якого треба вказати в квадратних дужках тип індексу (індексів) компонентів, потім, після зарезервованого слова of, тип самих компонентів:

type

< ім'я типу > = array [< тип індексу (індексів) >] of < тип компонентів >;

Наприклад:

```
type
    vector = array [1..3] of real;
    table = array[1..4,1..5] of integer;
```

Тут vector – тип-масив, що складається з трьох дійсних чисел, table – тип-двовимірний масив, що складається з чотирьох рядків і п'яти стовпців.

Увівши тип-масив, можна потім задати змінні цього типу.

Розмірність масиву може бути будь-якою, компоненти масиву можуть бути також будь-якого типу, індекс (індекси) може бути будь-якого порядкового типу, крім типів LongWord і Int64.

Так, для описаних вище типів можна задати, наприклад, такі змінні:

```
var
    m1,m2:vector;
    matr:table;
```

Тип-масив можна вводити безпосередньо і при описі відповідних змінних. Наприклад,

```
var
    m1,m2:array [1..3] of real;
    matr:array [1..4,1..5] of integer;
```

Тут визначені ті ж масиви, що й у попередньому прикладі.

У Object Pascal одним оператором присвоювання можна передати всі елементи одного масиву іншому масиву того ж типу, наприклад:

```
m1 := m2;
```

З іншого боку, оголошення

```
var
    a:array [1..6] of single;
    b:array [1..6] of single;
```

створить різні типи масивів, тому оператор

```
a := b;
```

викликає повідомлення про помилку.

Доступ до компонентів масиву здійснюється зазначенням імені масиву, за яким у квадратних дужках міститься значення індексу (індексів) компонента. У загальному випадку кожен індекс компонента може бути заданий виразом відповідного типу, наприклад:

```
m[1],        matr[x,y],        m2[succ(i)] .
```

Приклад 1.41.

Дано вектор A = (a_1, a_2, a_3, a_4, a_5) . Знайти максимальний і мінімальний елементи вектора і їхні номера.

Рішення.

Алгоритм пошуку максимального елемента в масиві полягає в наступному. Нехай змінна max повинна містити результат пошуку максимального елемента. Як початкове значення змінній max присвоїмо значення будь-якого елемента масиву, наприклад a_1 . Далі будемо порівнювати кожен елемент масиву зі значенням змінної max . Якщо виявиться, що значення чергового елемента масиву перевищує поточне значення змінної max, то змінної max присвоїмо значення цього елемента.

Мінімальне значення шукається аналогічно.

Відзначимо, що введення і виведення масивів у Object Pascal здійснюється поэлементно. Тому для організації операцій введення-виведення масивів варто використовувати цикли.

Програма.

```
program p1_41;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const n = 5;
var a: array [1..n] of real;
    min,max:real;
    i,nmax,nmin:integer;
begin
  writeln('Enter array');
  for i := 1 to n do
    read(a[i]);
  readln;
  max := a[1]; nmax := 1;
  min := a[1]; nmin := 1;
  for i := 2 to n do
    begin
    if min > a[i] then
      begin
      min := a[i];
      nmin := i
      end;
    if max < a[i] then
      begin
      max := a[i];
      nmax := i
      end
    end;
  writeln('min=',min:3:0,'  max=',max:3:0);
  writeln('nmin=',nmin:2,'  nmax=',nmax:2);
  readln
end.
```

Результати роботи програми.

```
Enter array
23 16 45 62 24
min= 16  max= 62
nmin= 2  nmax= 4
```

Помітимо, що після введення масиву в програмі потрібно додати процедуру readln без параметрів для того, щоб вікно, що містить результати роботи програми, не закривалося передчасно до натискання на клавішу Enter.

Приклад 1.42.

Знайти суму позитивних і суму негативних елементів вектора $A = (a_1, a_2, a_3, a_4, a_5)$.

Рішення.

Уведемо позначення: spol – сума позитивних чисел, sort – сума негативних чисел.

Програма.

```
program p1_42;
{$APPTYPE CONSOLE}
uses
  SysUtils;
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
const n = 6;
var a: array [1..n] of real;
    spol,sotr:real;
    i:integer;
begin
  writeln('Enter array');
  for i := 1 to n do
    read(a[i]);
  readln;
  spol := 0;
  sotr := 0;
  for i := 1 to n do
    begin
    if a[i] > 0 then spol := spol+a[i];
    if a[i] < 0 then sotr := sotr+a[i];
    end;
  writeln('spol=',spol:8:4,'  sotr=',sotr:8:4);
  readln
end.
```

Результати роботи програми.

```
Enter array
2.33 -4.56 -23 -15.02 10 15
spol= 27.3300  sotr=-42.5800
```

Розглянемо приклади програм з використанням двовимірних масивів.

Приклад 1.43.

Знайти номер стовпця, у якому знаходиться мінімальний елемент матриці A(4x3) і вивести цей стовпець на екран дисплея.

Рішення.

Поняттю матриці, використовуваному в математиці, у програмуванні відповідає поняття двовимірний масив. Для елемента двовимірного масиву a[i,j] перший індекс – i – позначає номер рядка, а другий індекс – j – позначає номер стовпця.

Алгоритм пошуку максимального елемента в двовимірному масиві аналогічний алгоритму, розглянутому для одновимірного масиву в прикладі 1.41. Але якщо для перебору елементів одновимірного масиву досить було одного циклу, то для перебору елементів двовимірного масиву потрібно використовувати подвійний цикл.

Програма.

```
program p1_43;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var a: array [1..4,1..3] of integer;
    min,nmin,i,j:integer;
begin
  writeln('Enter array');
  for i := 1 to 4 do
    begin
    for j := 1 to 3 do
      read(a[i,j]);
    readln;
    end;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
  min := a[1,1];
  nmin := 1;
  for i := 1 to 4 do
    for j := 1 to 3 do
      if min > a[i,j] then
        begin
        min := a[i,j];
        nmin := j
        end;
  writeln('min=',min:2,'  nmin=',nmin:2);
  for i := 1 to 4 do
    writeln(a[i,nmin]);
  readln
end.
```

Результати роботи програми.

```
Enter array
55 77 10
23  2 54
 5 34 25
85 62 15
min= 2  nmin= 2
77
2
34
62
```

Приклад 1.44.

У матриці A(3x3) поміняти місцями елементи першого й останнього рядків.

Рішення.

В елементів, що стоять у першому рядку, незмінним є перший індекс і його значення дорівнює 1: a[1,j] . Аналогічно, елементи третього рядка мають вид: a[3,j]. Для того щоб переставити елементи першого і третього рядка, розташовані в одному стовпці, треба виконати послідовність операторів:

```
c := a[1,j];
a[1,j] := a[3,j];
a[3,j] := c;
```

де j = 1,2,3.

Програма.

```
program p1_44;
{$APPTYPE CONSOLE}
uses
  SysUtils;
const n=3;
type mas=array [1..n,1..n] of integer;
var a:mas; c,i,j:integer;
begin
  writeln('Enter array');
  for i := 1 to n do
    begin
    for j := 1 to n do
```

```
    read(a[i,j]);
  readln;
  end;
 for j := 1 to n do
  begin
  c := a[1,j];
  a[1,j] := a[3,j];
  a[3,j] := c;
  end;
 writeln('Result array');
 for i := 1 to n do
  begin
  for j := 1 to n do
    write(a[i,j]:2);
  writeln;
  end;
 readln
end.
```

Результати роботи програми.

```
Enter array
1 1 1
2 2 2
3 3 3
Result array
 3 3 3
 2 2 2
 1 1 1
```

1.6.1.2. Динамічні масиви

Динамічні масиви з'явилися в Delphi 4 і являють собою зручний засіб для збереження даних для тих задач, де заздалегідь невідома кількість елементів в оброблюваних масивах. А оскільки програми повинні складатися для довільних даних, то появу динамічних масивів можна вважати закономірною.

Оголосити одновимірний динамічний масив можна в такий спосіб:

```
var
    < ім'я масиву > : array of < тип >;
```

Наприклад:

```
var
    vect:array of real;
```

Длину масиву треба задати за допомогою процедури SetLength. Наприклад:

```
SetLength(vect,6);
```

Виділяє для масиву vect в оперативній пам'яті 6 елементів і присвоює цим елементам нульові значення. Індекс першого елемента масиву дорівнює 0, тому масив vect містить елементи vect[0], vect[1],…,vect[5].

За бажанні довжину динамічного масиву можна змінити за допомогою повторного використання процедури SetLength. Якщо нова довжина виявиться більше вихідної, то в масив додаються нові елементи з нульовими значеннями; якщо менше, то останні елементи у вихідному масиві будуть відкинуті.

Змінна, що має тип динамічного масиву, є вказівником, тобто містить адресу ділянки пам'яті, починаючи з який розташовуються елементи масиву. Тому видалення з пам'яті динамічного масиву може бути здійснено одним із трьох способів:

1. Привласнити змінної значення nil, наприклад:

vect := nil;

2. Використовувати процедуру finalize, наприклад:

finalize(vect);

3. Установити нульову довжину, наприклад:

SetLength(vect,0);

Приклад 1.45.

Скласти програму для сортування елементів масиву довільної довжини за зростанням.

Рішення.

Застосуємо для рішення задачі метод бульбашки. Алгоритм цього методу полягає в наступному. Нехай у нас є одновимірний масив A=(a_1,a_2,a_3,…a_n). Порівняємо елементи a_1 і a_2 . Якщо виявиться, що $a_1 > a_2$, то переставимо місцями елементи a_1 і a_2, у противному випадку елементи залишаються на своїх місцях. Далі, незалежно від результатів попереднього порівняння, порівнюються елементи, що стоять на другому і третім місцях у масиві А. Якщо елемент, що стоїть на другому місці виявиться більше елемента, що стоїть на третім місці, то, як і раніше, вони міняються місцями. Процес продовжується, поки не буде виконане порівняння для передостаннього й останнього елементів масиву. У результаті на останнім, n-тому місці в масиві А виявиться найбільший елемент цього масиву.

Для того, щоб на передостаннім місці в масиві виявився елемент, найбільший із всіх елементів, крім останнього, описану вище процедуру застосовують до елементів, що стоять на місцях з 1-го по (n-1)-ше і т.д.

Оскільки в динамічному масиві індекси елементів змінюються від 0 до n-1, де n – кількість елементів у масиві, що відрізняється від позначень використаних вище, то в програмі для зручності введене змінні: g1 – початкове значення індексу, g2 – кінцеве значення індексу.

Програма.

```
program p1_45;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var a:array of integer;
    g1,g2,n,c,i,j:integer;
begin
    writeln('Enter length of array');
    readln(n);
    SetLength(a,n);
    g1 := 0;
    g2 := n-1;
    writeln('Enter array');
    for i := g1 to g2 do
      read(a[i]);
    readln;
    for i := g2-1 downto g1 do
      for j := g1 to i do
        if a[j] > a[j+1] then
          begin
          c := a[j];
          a[j] := a[j+1];
          a[j+1] := c;
          end;
    writeln('Sorted array');
    for i := g1 to g2 do
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
    write(a[i]:2);
  readln;
end.
```

Результати роботи програми.

```
Enter length of array
6
Enter array
6 5 4 3 2 1
Sorted array
 1 2 3 4 5 6
```

Двовимірний динамічний масив визначається як динамічний масив динамічних масивів, наприклад:

```
var
    matr:array of array of integer;
```

За цим же правилом можна визначити і тривимірний масив, наприклад:

```
var
    cube:array of array of array of char;
```

Визначити довжину багатовимірного масиву можна за допомогою тієї ж процедури SetLength, наприклад:

```
SetLength(matr,4,5);
```

створює масив розміром 4 на 5.

Як і для одновимірних масивів, початкове значення індексу в багатовимірних масивах дорівнює 0.

Приклад 1.46.

Скласти програму для знаходження суми двох квадратних матриць довільного розміру.

Рішення.

Нагадаємо, що сумою матриць A і B називається матриця C, елементи якої обчислюються за формуліою:

```
c[i,j] = a[i,j] + b[i,j]    i,j = 1,2,…,n...
```

Програма.

```
program p1_46;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type mas=array of array of integer;
var a,b,c:mas;
    g1,g2,n,i,j:integer;
begin
  writeln('Enter n');
  readln(n);
  SetLength(a,n,n);
  SetLength(b,n,n);
  SetLength(c,n,n);
  g1 := 0;
  g2 := n-1;
  writeln('Enter array a');
  for i := g1 to g2 do
    begin
    for j := g1 to g2 do
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
      read(a[i,j]);
    readln;
    end;
  writeln('Enter array b');
  for i := g1 to g2 do
    begin
    for j := g1 to g2 do
      read(b[i,j]);
    readln;
    end;
  for i := g1 to g2 do
    for j := g1 to g2 do
      c[i,j] := a[i,j]+b[i,j];
  writeln('Result array');
  for i := g1 to g2 do
    begin
    for j := g1 to g2 do
      write(c[i,j]:2);
    writeln;
    end;
  readln
end.
```

Результати роботи програми.

```
Enter n
4
Enter array a
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
Enter array b
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
Result array
 3 3 3 3
 3 3 3 3
 3 3 3 3
 3 3 3 3
```

Цікавою особливістю динамічних масивів є те, що можна створювати непрямокутні масиви, тобто коли для кожного рядка кількість стовпців різна. Для цього потрібно спочатку установити довжину першого виміру, а потім, з огляду на те, що кожен елемент динамічного масиву сам може бути динамічним масивом, визначити довжину кожного рядка. Наприклад, послідовність операторів

```
…….
var r:array of array of single;
…….
SetLength(r,3);
SetLength(r[0],1);
```

```
SetLength(r[1],2);
SetLength(r[2],3);
```

створює трикутний масив.

1.6.1.3. Параметри-масиви

Типом будь-якого параметра в списку формальних параметрів процедур і функцій може бути тільки стандартний чи раніше оголошений тип. Тому не можна, наприклад, оголосити наступну процедуру:

```
procedure t(x:array[1..10] of real);
```

тому що в списку формальних параметрів фактично з'являється тип-діапазон, що вказує границі індексів масиву.

Якщо в підпрограму передається масив, то варто спочатку описати його тип, наприклад:

```
type
      Tmas = array [1..10] of real;
procedure t(x:Tmas);
……
```

Приклад 1.47.

Дано матриці A(3x3) і B(3x2). У матриці A поміняти місцями 2-й і 3-й рядки, а в матриці B – 1-й і 3-й. Отримані матриці вивести на екран.

Рішення.

Опишемо три процедури: для введення матриці, виведення матриці й обміну рядків у матриці. Уведемо наступні позначення:

1. Для процедури InputMatrix: m, n – відповідно кількість рядків і стовпців матриці (вхідні параметри), x – матриця, що вводиться, (вихідний параметр).
2. Для процедури OutputMatrix: m, n – відповідно кількість рядків і стовпців матриці (вхідні параметри), x – виведена матриця (вхідний параметр).
3. Для процедури Exchange: n – кількість стовпців матриці (вхідний параметр), x – матриця, у якій потрібно поміняти місцями рядка (вхідний параметр), l і k – номера рядків, які потрібно поміняти місцями (вхідні параметри).
4. Початкові матриці позначимо a і b.

Програма.

```
program p1_47;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type matrix=array [1..3,1..3] of integer;
var a,b:matrix;
procedure InputMatrix(m,n:integer; var x:matrix);
var i,j:integer;
begin
  writeln('Enter array ');
  for i := 1 to m do
    begin
    for j := 1 to n do
      read(x[i,j]);
    readln;
    end
end;
procedure OutputMatrix(m,n:integer; x:matrix);
var i,j:integer;
begin
```

```
  writeln('Array after exchange');
  for i := 1 to m do
    begin
    for j := 1 to n do
      write(x[i,j]:2);
    writeln;
    end
end;
procedure Exchange(n,l,k:integer; var x:matrix);
var c,j:integer;
begin
  for j := 1 to n do
    begin
    c := x[l,j];
    x[l,j] := x[k,j];
    x[k,j] := c
    end
end;
begin
  InputMatrix(3,3,a);
  InputMatrix(3,2,b);
  Exchange(3,2,3,a);
  Exchange(2,3,1,b);
  OutputMatrix(3,3,a);
  OutputMatrix(3,2,b);
  readln
end.
```

Результати роботи програми.

```
Enter array
1 1 1
2 2 2
3 3 3
Enter array
1 1
2 2
3 3
Array after exchange
 1 1 1
 3 3 3
 2 2 2
Array after exchange
 3 3
 2 2
 1 1
```

Динамічний масив може передаватися як параметр у ті підпрограми, в описі яких відповідний формальний параметр оголошений як **відкритий масив**, наприклад:

```
procedure x(t:array of real; var w:array of array of real);
```

Можна сказати, що відкритий масив, використовуваний для передачі параметрів, аналогічний динамічному масиву. Зокрема, початкове значення індексу у відкритому масиві також дорівнює 0.

У підпрограмі реальна довжина масиву може бути визначена або за допомогою функції Length – довжина масиву, або за допомогою функції High – найбільше значення індексу. Для цих функцій справедливо наступне співвідношення:

High(< масив >) = Length(< масив >) – 1 .

Як фактичний параметр для відкритого масиву може виступати і статичний масив. При цьому треба враховувати, що якщо мінімальний індекс у статичному масиві не дорівнює 0, то все рівно перший елемент статичного масиву буде відповідати нульовому елементу відкритого масиву. Наприклад, якщо є наступні описи:

```
……
var z:array [1..10] of integer;
……
procedure p1(x:array of integer);
begin
……
end;
begin
……
p1(z);
……
```

то елемент z[1] буде відповідати елементу x[0] у відкритому масиві.

Приклад 1.48.

Використовуючи динамічні і відкриті масиви скласти програму за умовою приклада 1.47.

Рішення.

Залишимо колишні позначення, прийняті в прикладі 1.47.

У програмі для визначення максимального номера рядка і стовпця використовується функція High. Для двовимірного масиву x функція High(x) поверне максимальний номер рядка, оскільки двовимірний масив трактується як масив масивів, тобто одновимірний масив, елементами якого також є одновимірні масиви, що утворять рядки масиву x. При звертанні до функції High(x[0]) буде повернутий максимальний номер стовпця в нульовому рядку масиву x. Оскільки в задачі розглядаються прямокутні матриці, то High(x[0]) означає одночасно і максимальний номер стовпця для масиву x.

Програма.

```
program p1_48;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type matrix=array of array of integer;
var a,b:matrix;
    n,m:integer;
procedure InputMatrix(var x:matrix);
var i,j:integer;
begin
  writeln('Enter array ');
  for i := 0 to High(x) do
    begin
    for j := 0 to High(x[0]) do
      read(x[i,j]);
    readln;
    end
end;
procedure OutputMatrix(x:matrix);
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
var i,j:integer;
begin
  writeln('Array after exchange');
  for i := 0 to High(x) do
    begin
    for j := 0 to High(x[0]) do
      write(x[i,j]:2);
    writeln;
    end
end;
procedure Exchange(l,k:integer; var x:matrix);
var c,j:integer;
begin
  for j := 0 to High(x[0]) do
    begin
    c := x[l-1,j];
    x[l-1,j] := x[k-1,j];
    x[k-1,j] := c
    end
end;
begin
  writeln('Enter n,m for a');
  readln(n,m);
  SetLength(a,n,m);
  writeln('Enter n,m for b');
  readln(n,m);
  SetLength(b,n,m);
  InputMatrix(a);
  InputMatrix(b);
  Exchange(2,3,a);
  Exchange(3,1,b);
  OutputMatrix(a);
  OutputMatrix(b);
  readln
end.
```

Результати роботи програми.

```
Enter n,m for a
3 3
Enter n,m for b
3 2
Enter array
1 1 1
2 2 2
3 3 3
Enter array
1 1
2 2
3 3
Array after exchange
 1 1 1
 3 3 3
 2 2 2
Array after exchange
 3 3
 2 2
 1 1
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

1.6.2. Множини

Множинний тип являє собою скінченний набір значень деякого базового типу. Як базовий тип може використовуватися будь-який порядковий тип, крім word, integer, longint, int64.

Опис множинного типу має вид:

< ім'я типу > = set of < базовий тип >;

Наприклад:

```
type
    mn1 = set of 'A'..'Z';
    mn2 = set of 1..5;
```

Значеннями змінних множинного типу є будь-які підмножини базової множини. Наприклад, базовими будуть описані вище множини mn1 і mn2.

Замість словосполучення «значення множинного типу» частіше, для стислості, використовують слово «множина». Для завдання множини використовується **конструктор** множини, що представляє собою список елементів базової множини, розділених комами й обрамлений квадратними дужками. Так, наприклад, якщо в нас маються змінні

```
var
    p1:mn1;
    p1:mn2;
```

те їм можна як значення присвоїти наступні множини:

```
p1 := ['A','B','C'];
p1 := ['Z','Y','X'];
p1 := ['A','K'..'T'];
p2 := [1,2,4];
p2 := [2..5];
```

Порядок елементів у множині несуттєвий і, наприклад, множини ['A','B'] і ['B','A'] є еквівалентними. Множина, що не містить елементів, називається порожньою і позначається як [ ]. Якщо елементи множини є послідовними значеннями базової множини, то можна вказати тільки перший і останній з них, розділивши їх двома крапками, аналогічно тому, як це робиться в типі-діапазоні. З іншого боку, множина ['Z'..'A'] буде порожньою, оскільки порядковий номер символу 'Z' більше, ніж у символу 'A'.

Елементи множин можуть задаватися за допомогою виразів відповідного базового типу:

```
p2 := [L+2..5];
```

Якщо L – ціла і дорівнює 1, то множина p2 має вид: [3,4,5].

Над множинами визначені такі операції:

+ – об'єднання;
* – перетин;
- – різниця.

Об'єднанням двох множин називається множина, що складається з елементів першої і другої множини. Наприклад:

['A','B'] + ['C'..'E'] містить ['A','B','C','D','E'] ,

['A','B'] + ['B','C'] містить ['A','B','C'],

оскільки кожен елемент входить у множину тільки один раз.

Перетином двох множин називається множина, що складається з елементів, які одночасно належать двом вихідним множинам. Наприклад:

['A','B'] * ['B','C'] містить ['B'],

а

['A','B'] * ['C'..'E'] є порожньою [ ] .

Різницею двох множин називається множина, що містить елементи першої множини, не приналежні другий, наприклад:

['A','B'] - ['B','C'] містить ['A'].

Для порівняння множин використовуються наступні операції:

= – вираз X = Y істинний, якщо X і Y містять ті самі елементи, тобто є еквівалентними;

<> – вираз X <> Y істинний, якщо одна з множин містить хоча б один елемент, що не міститься в другому, тобто множини не еквівалентні;

<= – вираз X <= Y істинний, якщо всі елементи множини X є одночасно елементами множини Y, тобто X підмножина Y;

>= – вираз X => Y істинний, якщо Y є підмножиною X;

in – операція, використовувана у виразах виду

< вираз базового типу > in < вираз множинного типу >;

Результат операції in дорівнює true, якщо значення < вираз базового типу > міститься в < виразі множинного типу >.

Істинними є наступні вирази:

['A'..'C'] = ['A','B','C'];
[1,2,3,4] >[2..4];
['A'..'Z'] >= ['Z'..'A'];
2 in [1..5];
'T' in ['A'..'Z'].

У порядку пріоритету перераховані операції можна розташувати в такий спосіб:

*
+, -
in, =, <>, <=, >=.

Використання множин дозволяє зробити програми більш ефективними за рахунок зменшення кількості різних перевірок.

Приклад 1.49.

Скласти програму, що підраховує загальну кількість цифр і знаків '+', '-', '*', що входять у рядок S.

Рішення.

Алгоритм рішення задачі складається в перевірці на приналежність кожного символу рядка S зазначеній множині.

Програма.

```
program p1_49;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var s:string; i,k:integer;
begin
  writeln('Enter s');
  readln(s);
  k := 0;
  for i := 1 to length(s) do
    if s[i] in ['0'..'9','+','-','*'] then
      k := k+1;
```

```
  writeln('k=',k:2);
  readln
end.
```

Результати роботи програми.

```
Enter s
abcd1243+*-
k= 7
```

1.6.3. Записи

Запис – це складна змінна, що складається з декількох компонентів, називаних полями. Кожне поле може мати свій тип. Завдяки цьому в запису може зберігатися досить різнорідна інформація. Наприклад, анкетні дані співробітника організації, що можуть включати: рік народження, освіту, родиний стан, кількість дітей і т.д.

1.6.3.1. Оголошення записів

Тип запису в загальному виді оголошується в такий спосіб:

```
type
    < ім'я типу > = record
        < ім'я_поля_1> : < тип >;
        < ім'я_поля_2> : < тип >;
        ……..
        < ім'я_поля_N> : < тип >;
    end;
```

Наприклад:

```
type
    TAuto = record
        model : string;
        year : integer;
        price : real;
        color : (white,blue,red,black);
    end;
```

У записах типу TAuto будуть міститися дані про автомобілі: у поле model – марка автомобіля, у year – рік випуску, у price – ціна автомобіля, у color – колір.

Змінні типу запису описуються в розділі var звичайним образом:

```
var
    x,y:TAuto;
```

Змінну-запис можна було б оголосити і безпосередньо в розділі var:

```
var
    x = record
        model : string;
        year : integer;
        price : real;
        color : (white,blue,red,black);
    end;
```

До кожного поля запису можна звернутися, використовуючи ім'я змінної типу запису й ім'я поля, розділених крапкою, наприклад:

```
x.model := 'opel';
x.year := 1998;
x.price := 4652.35;
x.color := blue;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

Приклад 1.50.

Скласти програму, що виконує наступні дії:

1. Уводить дані про студентів, що містять: прізвище, групу, оцінки по інформатиці, вищій математиці і фізиці.
2. Виводить на екран дисплея прізвища студентів, що починаються на букву 'S' і середній бал у який більше 4,5 .

Рішення.

Оскільки кількість студентів заздалегідь не обговорено, то для збереження даних про студентів будемо використовувати динамічний масив записів.

Програма.

```
program p1_50;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type
    student = record
      name:string;
      gruppa:string;
      marks:record
        inf,math,phiz:integer;
      end
    end;
var
    students: array of student;
    i,n:integer;s:real;
begin
  writeln('Enter number of students');
  readln(n);
  setlength(students,n);
  for i := 0 to n-1 do
    begin
    write('Enter name ');readln(students[i].name);
    write('Enter gruppa ');readln(students[i].gruppa);
    write('Enter inf ');readln(students[i].marks.inf);
    write('Enter math ');readln(students[i].marks.math);
    write('Enter phiz ');readln(students[i].marks.phiz);
    end;
  writeln('Result:');
  for i := 0 to n-1 do
    begin
    s := (students[i].marks.inf + students[i].marks.math +
         students[i].marks.phiz)/3;
    if ( students[i].name[1] = 'S') and (s > 4.5) then
      writeln(students[i].name)
    end;
  readln
end.
```

Результати роботи програми.

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
Enter number of students
4
Enter name Sidorov
Enter gruppa 1pp24
Enter inf 5
Enter math 5
Enter phiz 5
Enter name Ivanov
Enter gruppa 1pp25
Enter inf 5
Enter math 5
Enter phiz 5
Enter name Sidorchuk
Enter gruppa 1pp25
Enter inf 4
Enter math 5
Enter phiz 5
Enter name Serov
Enter gruppa 1pp24
Enter inf 3
Enter math 3
Enter phiz 4
Result:
Sidorov
Sidorchuk
```

Помітимо, що третє поле типу запису student також має тип запису. Тому до полів inf, math, phiz варто звертатися, указуючи три імені через крапку.

1.6.3.2. Оператор приєднання WITH

Щоб спростити доступ до полів, варто використовувати оператор приєднання with, що у загальному виді записується в такий спосіб:

with < список змінних, записів, полів > do < оператор >;

Наприклад, оператори

```
x.model := 'opel';
y.marks.inf := 4;
```

можна записати так:

```
with x do model := 'opel';
with a, marks do inf := 4;
```

Використовуючи оператор with, програму з попереднього пункту можна істотно спростити:

```
program p1_50a;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type
    student = record
      name:string;
      gruppa:string;
      marks:record
        inf,math,phiz:integer;
```

```
      end
    end;
var
    students: array of student;
    i,n:integer;s:real;
begin
  writeln('Enter number of students');
  readln(n);
  setlength(students,n);
  for i := 0 to n-1 do
    with students[i],marks do
    begin
    write('Enter name ');readln(name);
    write('Enter gruppa ');readln(gruppa);
    write('Enter inf ');readln(inf);
    write('Enter math ');readln(math);
    write('Enter phiz ');readln(phiz);
    end;
  writeln('Result:');
  for i := 0 to n-1 do
    with students[i],marks do
    begin
    s := (inf + math +phiz)/3;
    if ( name[1] = 'S') and (s > 4.5) then
      writeln(name)
    end;
  readln
end.
```

1.6.4. Файли

1.6.4.1. Файлові типи і файлові змінні

Файл – це набір інформації, що має ім'я. Файли призначені для збереження інформації на зовнішніх запам'ятовуючих пристроях – вінчестері, дискетах, лазерних компакт-дисках і т.д. Документ, створений у текстовому редакторі, малюнок, створений у графічному редакторі, консольне застосування – це всі приклади файлів. Файли різних типів, створювані й оброблювані різними застосуваннями, мають різний формат, тобто внутрішнє представлення даних. Незважаючи на всі розходження, загальним для усіх файлів є те, що в них є ім'я, а також те, що довжина файлу визначається тільки обсягом інформації, поміщеної у нього, і обмежується обсягом зовнішнього запам'ятовуючого пристрою.

Object Pascal має у своєму розпорядженні засоби створення й обробки файлів різних типів. Для того щоб одержати доступ до файлу, потрібно мати можливість зв'язати створене в Delphi застосування з деяким файлом для читання чи запису інформації. Цей зв'язок створюється за допомогою змінних файлового типу, чи, інакше, – файлових змінних.

У Object Pascal існує три файлових типи:

TextFile – текстовий файл, що представляє собою набір символьних рядків змінної довжини;

File of < тип > – типізований файл, що представляє собою набір даних зазначеного < типу >;

File – нетипізований файл, що представляє собою набір неструктурованих даних.

Приведемо приклад опису файлових змінних:

```
var
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
a1:textfile;
a2:file of integer;
a3:file of char;
a4:file;
```

тут a1 – текстовий файл, a2 і a3 – типізовані файли, a4 – нетипізований файл.

1.6.4.2. Стандартні підпрограми для доступу до файлів

Перед використанням файлової змінної вона повинна бути зв'язана з зовнішнім файлом за допомогою виклику процедури AssignFile:

AssignFile(< файлова змінна >, < ім'я файлу >);

тут < файлова змінна > – ім'я змінної, оголошеної в програмі як змінної файлового типу; < ім'я файлу > – символьний рядок, що містить ім'я файлу. Якщо файл знаходиться в одній папці з обробляючою його програмою, то досить указати тільки ім'я файлу, у противному випадку треба вказати повний шлях до файлу, наприклад:

'd:\programs\petja\myfile.txt' .

Коли зв'язок із зовнішнім файлом установлений, його можна відкрити для уведення чи виведення інформації. Існуючий файл можна відкрити за допомогою процедури Reset:

Reset(< файлова змінна >);

Процедура Reset відкриває існуючий зовнішній файл, ім'я якого було зв'язано з файловою змінною. Якщо зовнішній файл із зазначеним ім'ям відсутний, то виникає помилка періоду виконання програми (генерується виняток). Якщо файл уже відкритий, то він спочатку закривається, а потім відкривається знову. Поточна позиція у файлі встановлюється на початок файлу, тобто зв'язана з файлом змінна-вказівник буде вказувати на компонент із порядковим номером 0.

Якщо файлова змінна зв'язана з текстовим файлом, то він буде доступний тільки для читання. Для типізованих і нетипизованных файлів, відкритих процедурою Reset, допускається виконувати операції читання і запису у файл.

Новий файл можна створити і відкрити для запису за допомогою процедури Rewrite:

Rewrite(< файлова змінна >);

Процедура Rewrite створює новий зовнішній файл, ім'я якого зв'язане з файловою змінною. Якщо зовнішній файл із зазначеним ім'ям вже існує, то він видаляється і на його місці створюється новий порожній файл. Якщо файл уже відкритий, то він спочатку закривається, а потім відкривається знову. Поточна позиція у файлі встановлюється на початок файлу, тобто зв'язана з файлом змінна-вказівник буде вказувати на компонент із порядковим номером 0

Якщо процедура Rewrite відкриває текстовий файл, то він стає доступним тільки для запису. Для типізованих і нетипизованных файлів, відкритих процедурою Rewrite, допускається виконувати операції читання і запису у файл.

Текстовий файл може бути відкритий процедурою Append:

Append(< файлова змінна >);

Процедура Append відкриває існуючий зовнішній файл, зв'язаний з файловою змінною, для приєднання. Якщо зовнішнього файлу з зазначеним ім'ям не існує, то генерується виняток. Якщо файл уже відкритий, то він спочатку закривається, а потім відкривається заново. Змінна-вказівник буде вказувати на кінець файлу. Після звертання до Append текстовий файл стає доступним тільки для запису.

Коли програма завершує обробку файлу, він повинний закриватися за допомогою стандартної процедури CloseFile:

CloseFile(< файлова змінна >);

Процедура CloseFile закриває відкритий файл. При цьому забезпечується збереження у файлі всіх нових записів і реєстрація файлу в папці. Процедура CloseFile не розриває зв'язок файлу з файловою змінною, тому файл можна відкрити знову без повторного використання процедури AssignFile.

При відкритті файлу корисно використовувати стандартну функцію FileExists:

FileExists(< ім'я файлу >);

Тут < ім'я файлу > – це символьний рядок, що містить ім'я файлу на зовнішньому запам'ятовуючому пристрої. Якщо файл існує, то функція повертає true, у противному випадку – false.

Помітимо, що для нетипізованих файлів, синтаксис процедур AssignFile, Reset і Rewrite трохи відрізняється від приведеного вище. Ці відмінності будуть розглянуті при розгляді нетипізованих файлів.

1.6.4.3. Текстові файли

Текстовий файл являє собою послідовність символів, згрупованих у рядки довільної довжини, де кожен рядок закінчується символом кінця рядка – EOLN (end-of-line), що являє собою символ переведення каретки CR (код з номером 13 – #13), за яким, можливо, слідує символ переведення рядка LF (#10). Закінчується файл символом кінця файлу EOF (end-of-file, код – #26). Прикладом текстового файлу є текст програми, тобто файл із розширенням dpr .

Уведення даних з довільного текстового файлу можна здійснити процедурами read і readln з тією лише різницею, що в списку їхніх параметрів першою повинна стояти відповідна файлова змінна, наприклад:

read(f,x,y,z);

означає прочитати з файлу, зв'язаного з файловою змінною f, значення змінних x,y,z . Процедура

readln(f,a);

прочитає з файлу, зв'язаного з файловою змінною f, значення змінної a і перейде в цьому файлі до наступного рядку.

Процедури read і readln здійснюють послідовне читання з файлу символьних представлень змінних.

Виведення даних у довільний текстовий файл здійснюється процедурами write і writeln, у яких першою в списку параметрів зазначена відповідна файлова змінна, наприклад:

write(fi,'s=',s)

здійснює виведення у файл, зв'язаний з файловою змінною fi, символьного рядка 's=' і значення змінної s. Процедура

writeln(fi);

виводить у файл, зв'язаний з файловою змінною fi, порожній рядок.

Приклад 1.51.

Скласти програму для знаходження суми двох квадратних матриць довільного розміру. Уведення вихідних даних здійснити з текстового файлу inp.txt, результати вивести у файл result.txt .

Рішення.

Рішення цієї задачі без використання текстових файлів розглянуто в прикладі 1.46. Для створення текстового файлу inp.txt, що містить розмір матриць n, а також самі матриці a і b, можна використовувати будь-який текстовий редактор, наприклад, програму Блокнот. Можна зробити ще простіше – використовувати наявний у Delphi редактор коду, але замість тексту програми набрати вихідні дані. При збереженні файлу необхідно правильно вказати ім'я файлу, включаючи його розширення – txt, оскільки за замовчуванням редактор коду додає розширення dpr.

Програма.

```
program p1_51;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type mas=array of array of integer;
```

```
var a,b,c:mas;
    g1,g2,n,i,j:integer;
    f1,f2:TextFile;
begin
  AssignFile(f1,'inp.txt');
  AssignFile(f2,'result.txt');
  Reset(f1);
  Rewrite(f2);
  readln(f1,n);
  SetLength(a,n,n);
  SetLength(b,n,n);
  SetLength(c,n,n);
  g1 := 0;
  g2 := n-1;
  for i := g1 to g2 do
    begin
    for j := g1 to g2 do
      read(f1,a[i,j]);
    readln(f1)
    end;
  for i := g1 to g2 do
    begin
    for j := g1 to g2 do
      read(f1,b[i,j]);
    readln(f1)
    end;
  CloseFile(f1);
  for i := g1 to g2 do
    for j := g1 to g2 do
      c[i,j] := a[i,j]+b[i,j];
  writeln(f2,'Result array');
  for i := g1 to g2 do
    begin
    for j := g1 to g2 do
      write(f2,c[i,j]:2);
    writeln(f2)
    end;
  CloseFile(f2);
  writeln('Program terminated');
  readln
end.
```

Результати роботи програми.

Створений до запуску програми файл inp.txt містить наступні дані:

```
4
1 1 1 1
1 1 1 1
1 1 1 1
1 1 1 1
2 2 2 2
2 2 2 2
2 2 2 2
2 2 2 2
```

У результаті виконання програми буде створений файл result.txt. Нижче приведене його вміст:

```
Result array
 3 3 3 3
 3 3 3 3
 3 3 3 3
 3 3 3 3
```

Використовувані файли inp.txt і result.txt знаходяться в одній папці з виконуваним файлом програми. Тому в процедурах AssignFile зазначені тільки імена файлів без шляху доступу до них. Якщо файли будуть знаходитися в різних папках, то необхідно вказувати шлях.

Відзначимо також, що в приведеному приклада проілюстрована важлива властивість текстових файлів: вони можуть бути створені і переглянуті будь-яким текстовим редактором.

При роботі з текстовими файлами приведені нижче функції дозволяють контролювати положення поточної позиції у файлі:

function Eof [(var F: FileText)]: Boolean;
Функція повертає значення true, якщо досягнутий кінець текстового файлу, зв'язаного з файловою змінною F. У противному випадку повертається false.

function Eoln [(var F: FileText)]: Boolean;
Функція повертає значення true, якщо досягнутий кінець рядка в текстовому файлі, зв'язаного з файловою змінною F. У противному випадку повертається false.

function SeekEof [(var F: FileText)]: Boolean;
Діє аналогічно Eof, але пробіли, знаки табуляції і маркери кінця рядка EOLN пропускаються.

function SeekEoln [(var F: FileText)]: Boolean;
Діє аналогічно Eoln, але пробіли і знаки табуляції пропускаються.

Використовуючи функції Eof і Eoln можна було б організувати введення масивів із приклада 1.51 трохи інакше. Нехай, наприклад, fa – файлова змінна, зв'язана з текстовим файлом, що містить тільки масив a. Тоді цей масив можна було б ввести в такий спосіб:

```
……….
 i := 0;
 while not eof(fa)do
   begin
   j := 0;
   while not eoln(fa) do
     begin
     read(fa,a[i,j]);
     j := j+1
     end;
   readln(fa);
   i := i+1;
   end;
…….....
```

Якщо текстовий файл містить символьне представлення числових даних, то замість функцій eof і eoln переважніше використовувати SeekEof і SeekEoln.

1.6.4.4. Типізовані файли

Типізований файл містить компоненти одного типу. Тип компонентів може бути будь-яким, крім файлового. Створити і переглянути такий файл за допомогою текстового редактора як текстовий файл не можна. Тому обробка таких файлів повинна здійснюватися програмним шляхом.

Для читання даних з типізованого файлу застосовується процедура read. Список уведення процедури read повинний мати змінні того ж типу, що і компоненти файлу. Для запису в типізований файл використовується процедура write, список виведення якої також повинний мати вирази того ж типу, що і компоненти файлу. Процедури readln і writeln, використовувані для текстових файлів, для типізованих файлів не застосовуються.

Звичайно доступ до файлів організується послідовно, тобто, коли елемент зчитується за допомогою стандартної процедури read чи записується за допомогою стандартної процедури write, вказівник файлу зміщається до наступного за порядком елементу файлу. Однак до типізованих файлів можна організувати прямий доступ за допомогою стандартної процедури Seek, що зміщає вказівник файлу до заданого елемента. Для визначення поточної позиції у файлі і поточного розміру файлу можна використовувати стандартні функції Filepos і FileSize. Нижче приведений опис цих підпрограм.

procedure Seek(var F; N: Longint);
Процедура зміщає поточну позицію (вказівник) у типізованому файлі, зв'язаному з файловою змінною F до необхідного компонента з номером N. Нумерація компонентів у файлі починається з нуля.

function FilePos(var F): Longint;
Повертає номер поточного компонента у файлі, зв'язаному з файловою змінною F.

function FileSize(var F): Integer;
Повертає кількість компонентів у файлі, зв'язаному з файловою змінною F.

Приклад 1.52

Скласти програму, що виконує наступні дії:

1. Уводить із клавіатури елементи цілочисленої матриці A(3x3) і записує їх у типізований файл f1.dat .
2. Виводить на екран третій рядок матриці A з файлу f1.dat .

Рішення.

Нумерація елементів двовимірного масиву у файлі йде зліва направо, зверху вниз. Зважаючи на те, що нумерація починається з нуля, елемент матриці A[3,1] буде шостим компонентом у файлі.

Програма.

```
program p1_52;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var a:array [1..3,1..3] of integer;
    f:file of integer;
    i,j:integer;
begin
  AssignFile(f,'f1.dat');
  Rewrite(f);
```

```
writeln('Enter array');
for i := 1 to 3 do
  begin
  for j := 1 to 3 do
    read(a[i,j]);
  readln
  end;
for i := 1 to 3 do
  for j := 1 to 3 do
    write(f,a[i,j]);
seek(f,6);
writeln('Result:');
for j := 1 to 3 do
  begin
  read(f,a[3,j]);
  write(a[3,j]:2)
  end;
CloseFile(f);
readln
end.
```

Результати роботи програми.

```
Enter array
1 2 3
4 5 6
7 8 9
Result:
 7 8 9
```

1.6.4.5. Нетипізовані файли

З погляду Object Pascal, нетипізований файл являє собою послідовність байтів, що містять дані довільного типу і структури. Основне призначення нетипізованих файлів – забезпечення сумісності з будь-якими типами файлів, використовуваних в операційній системі Windows, і організація високошвидкісного обміну даними між зовнішніми запам'ятовуючими пристроями й оперативною пам'яттю.

Для зв'язку нетипізованого файлу з файловою змінною використовується описана раніше процедура AssignFile.

У процедурах Reset і Rewrite для нетипізованих файлів указується додатковий параметр RecSize, щоб задати розмір запису, що використовується при передачі файлу:

procedure Reset(var F: File [; RecSize: Word]);

procedure Rewrite(var F: File [; RecSize: Word]);

Якщо параметр RecSize не зазначений, то прийнята за замовчуванням довжина запису дорівнює 128 байтам. Довжина запису виміряється в байтах і може бути задана довільним цілим числом – від 1 байта до 2 Гбайт. Якщо задати довжину, рівною одному байту, то це дозволить точно відбити розмір будь-якого файлу (коли довжина запису дорівнює 1, то у файлі не можуть бути присутніми неповні записи, тобто записи з меншою довжиною). Якщо задати довжину запису, кратною 512 байт (512 байт – це розмір фізичного сектора на зовнішнім запам'ятовуючому пристрої), то це дозволить виконувати операції читання-запису для нетипізованого файлу з максимальною швидкістю.

За винятком процедур Read і Write для всіх нетипізованих файлів допускається використання будь-якої стандартної процедури, що допускається використовувати з

типізованими файлами. Замість процедур Read і Write тут використовуються відповідно процедури BlockRead і BlockWrite, що дозволяють пересилати дані з високою швидкістю:

procedure BlockRead(var F: File; var Buf; Count: Integer [; var AmtTransferred: Integer]);

procedure BlockWrite(var F: File; var Buf; Count: Integer [; var AmtTransferred: Integer]);

Тут F – ім'я файлової змінної, зв'язаної з нетипізованим файлом, Buf – змінна, у яку будуть міститися дані при читанні з файлу і з який будуть витягатися дані при запису у файл. Змінна Buf повинна мати довжину, рівну Count* RecSize байт. Count – параметр цілого типу, що вказує, яку кількість записів необхідно прочитати чи записати за одне звертання до файлу. Необов'язковий параметр AmtTransferred містить кількість реально прочитаних чи записаних записів.

Приклад 1.53.

Скласти програму для копіювання графічного файлу zz.bmp з поточного каталогу на дискету під ім'ям qq.bmp .

Рішення.

Як буфер будемо використовувати змінну b, що є масивом типу byte. Він містить 1024 елемента, що відповідає розміру одного кілобайта. Розмір одного запису в процедурах Reset і Rewrite також установимо рівним 1024 байт. Процедура FileSize, поверне кількість записів, що містяться у вхідному файлі. У циклі for будемо зчитувати по одному запису (по одному кілобайту) з вхідного файлу і записувати у вихідний файл.

Програма.

```
program p1_53;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var b:array[1..1024] of byte;
    f1,f2:file;
    s1,s2:string;
    count,i:integer;
begin
  writeln('Enter name of first file');
  readln(s1);
  writeln('Enter name of second file');
  readln(s2);
  AssignFile(f1,s1);
  AssignFile(f2,s2);
  if FileExists(s1) then
    begin
    Reset(f1,1024);
    Rewrite(f2,1024);
    writeln('Size of file '+s1+' = ',FileSize(f1));
    for i :=1 to  FileSize(f1) do
    begin
      BlockRead(f1,b,1);
      BlockWrite(f2,b,1);
    end;
    writeln('process terminated...');
    end
  else
    writeln('File not exists');
  readln
end.
```

Результати роботи програми.

PDF-XChange Click to buy NOW! www.docu-track.com

```
Enter name of first file
zz.bmp
Enter name of second file
a:qq.bmp
Size of file zz.bmp = 482
process terminated...
```

1.6.4.6. Деякі допоміжні процедури і функції для роботи з файлами

Нижче приведений опис процедур і функцій, що можуть використовуватися для файлів усіх типів:

procedure ChDir(S: string);
Змінює поточну папку на папку, зазначену в символьному рядку S.

function DeleteFile(const FileName: string): Boolean;
Видаляє файл, ім'я якого міститься в символьному рядку FileName. Якщо видалення пройшло успішно, функція повертає true, у противному випадку вона повертає false.

procedure GetDir(D: Byte; var S: string);
Повертає поточну папку на диску, визначеному параметром D:

Значення параметра D	Диск
0	поточний диск
1	A
2	B
3	C
і т.д.	

procedure MkDir(S: string);
Створює нову папку, ім'я якої зазначено в символьному рядку S.

procedure Rename(var F; Newname:string);
Перейменовує файл, зв'язаний з файловою змінною F. Нове ім'я файлу задається в змінній Newname.

procedure RmDir(S: string);
Видаляє папку, ім'я якої зазначено в змінній S. Папка, що видаляється, повинна бути порожньою, тобто не містити файлів чи інших папок.

1.7. Тип variant

Тип variant використовується в тих випадках, коли заздалегідь невідомо, значення яких типів будуть використовуватися в програмі, або коли стандартна підпрограма Object Pascal має параметри з таким типом.

Значеннями змінних типу variant можуть бути значення цілих, дійсних, логічних, рядкових, дата-час типів, а також динамічні масиви і так називані варіантні масиви. Наприклад:

```
var
        x: variant;
……
x := 25;
x := 1.15;
x := true
```

```
x := ‘example’;
……
```

Змінна типу variant займає 16 байт у пам'яті комп'ютера, де в перших двох байтах міститься інформація про тип розташованого в змінній значення. У момент створення змінної типу variant їй присвоюється спеціальне значення unassigned. Значення null означає, що змінній типу variant присвоєне помилкове чи невідоме значення.

До складу Delphi 6 входить модуль Variants, що містить підпрограми для роботи з даними типу variant. Так, наприклад, визначити тип значення, що зберігається в змінній типу variant можна, використовуючи функцію VarType, що повертає код типу, – деяке значення цілого типу. Якщо отриманий код скласти зі стандартною константою varTypeMask за допомогою логічної поразрядной операції and

VarType(x) and varTypeMask ,

то результатом буде одне з наступних значень:

Значення, що повертаються функцією VarType

Таблиця 1.13.

Значення	Тип аргументу
varEmpty	Аргумент не містить значення
varNull	Аргумент містить помилкове чи невідоме значення
varSmallint	тип Smallint
varInteger	тип Integer
varSingle	тип Single
varDouble	тип Double
varCurrency	тип Currency
varDate	тип TDateTime
varError	код помилки операційної системи
varBoolean	тип WordBool
varVariant	тип Variant
varByte	тип Byte
varString	тип AnsiString

Наприклад, вираз

VarType(x) and varTypeMask = varInteger

прийме значення true, якщо в змінній x, що має тип variant, зберігається ціле значення.

Якщо змінна типу variant використовується в виразі чи її значення присвоюється змінній іншого типу, то перетворення типів відбувається за наступними правилами:

Правила перетворення типів для значень, що зберігаються у варіанті

Таблиця 1.14.

Тип значення, що зберігається у варіанті	До якого типу даних приводиться			
	Цілі	Дійсні	Рядкові	Логічні
Цілі	Перетворення у відповідний цілий тип	Перетворення у відповідний дійсний тип	Число перетвориться у відповідний рядок символів	False для 0, інакше - True
Дійсні	Округлення до найближчого цілого	Перетворення у відповідний дійсний тип	Число перетворюється у відповідній рядок символів	False для 0, інакше - True
Рядкові	Рядок символів, що містить	Рядок символів, що містить	Перетворення у відповідний	False для 'False' і '0'; True для

	цифри, перетвориться в ціле число	цифри, перетвориться у дійсне число	рядковий тип	'True' і для інших цифр
Логічні	0 для False; -1 для True (255 для типу Byte).	0 для False; -1 для True.	'0'для False; '-1' для True.	False для False; True для True.
Unassigned	0	0.0	" -порожній рядок	False
Null	Збуджується виняток	Збуджується виняток	Збуджується виняток	Збуджується виняток

Варто також враховувати, що якщо один чи обидва операнди в операції дорівнюють null, то результат операції теж буде дорівнювати null. А якщо один чи обидва операнди мають значення Unassigned, те збуджується виняток. Якщо в бінарній операції один операнд є змінною типу variant, то інший операнд також перетвориться в тип variant.

Приклад 1.54.

Студенти, що вивчають інформатику, навчаються на різних спеціальностях і відповідно мають різні форми звітності: економісти – іспит, механіки – залік. Обчислити кількість студентів, які успішно освоїли дисципліну.

Рішення.

Уведемо позначення:

econom – масив з десятьма елементами цілого типу, значення яких лежать у діапазоні 2..5 – містить оцінки, отримані економістами на іспиті;

mechan – масив з десятьма елементами рядкового типу, значеннями яких можуть бути рядки ‘yes’, ’YES’, ’Yes’, ‘no’, ‘NO’, ‘No’, що позначають, зданий залік чині;

s – глобальна змінна, у якій накопичується кількість студентів, що успішно освоїли дисципліну.

Стандартна функція AnsiUpperCase перетворить усі символи рядка в заголовні. Її використання в програмі дозволяє спростити перевірку значень, що вводяться.

Процедура count, що має вхідний параметр x типу variant, визначає тип значення, що міститься в x, і збільшує значення глобальної змінний s, якщо в x міститься позитивна оцінка чи оцінка про здачу заліку.

Програма.

```
program p1_54;
{$APPTYPE CONSOLE}
uses
  SysUtils, Variants;
const n=10;
var
   econom:array[1..n] of integer;
   mechan:array[1..n] of string;
   s,i:integer;
procedure count(x:variant);
  begin
  if (VarType(x) and varTypeMask = varInteger) and (x > 2)
    then  s := s + 1;
  if (VarType(x) and varTypeMask = varString) and
   (AnsiUpperCase(x)='YES') then s := s + 1;
  end;
begin
  s := 0;
  writeln('Enter marks of economists');
  for i :=1 to n do read(econom[i]); readln;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
  writeln('Enter marks of mechanics');
  for i :=1 to n do readln(mechan[i]);
  for i := 1 to 10 do
    begin
    count(econom[i]);
    count(mechan[i]);
    end;
  writeln('Result:');
  writeln('s = ',s);
  readln
end.
```

Результати роботи програми.

```
Enter marks of economists
2 3 4 5 2 3 4 5 2 3
Enter marks of mechanics
yes
no
yes
no
yes
no
yes
no
yes
no
Result:
s = 12
```

Помітимо, що дані типу variant, подібно даним типу boolean, не можна вводити за допомогою процедур read і readln, але їхнє значення можна виводити на екран дисплея за допомогою процедур write і writeln.

У змінну типу variant можна помістити так називаний варіантний масив. Зробити це можна за допомогою функції VarArrayCreate, що визначається в модулі Variants у такий спосіб:

function VarArrayCreate(const Bounds: array of Integer; VarType: Integer): Variant;

Функція VarArrayCreate створює варіантний масив з елементів типу VarType з кількістю і границями вимірів, зазначеними у параметрі Bounds, наприклад:

```
var    y:variant;
……
y := VarArrayCreate([0,5],varDouble);
```

У змінній y створений масив із шістьма значеннями типу Double.

Можна створити варіантний масив, елементами якого будуть також варіанти:

```
var    z:variant;
……
z := VarArrayCreate([0,5],varVariant);
```

Елементи масиву z можуть приймати значення різних типів:

```
z[0] := 6;
z[1] := true;
z[2] := 7.77;
z[3] := 'Ivanov I.I.';
```

Крім функції VarArrayCreate, для роботи з варіантними масивами можуть бути використані наступні підпрограми, визначені в модулі Variants:

function VarArrayDimCount(const A: Variant): Integer;
Повертає число вимірів у варіантному масиві A. Результат, що повертається, буде дорівнювати 0, якщо A не є варіантним масивом.

function VarArrayHighBound(const A: Variant; Dim: Integer): Integer;
Повертає верхню границю індексу для зазначеного виміру Dim у варіантному масиві A.

function VarArrayLowBound(const A: Variant; Dim: Integer): Integer;
Повертає нижню границю індексу для зазначеного виміру Dim у варіантному масиві A.

function VarArrayOf(const Values:array of Variant): Variant;
Повертає одновимірний варіантний масив, елементи якого задані в параметрі Values. Нижня границя індексів створюваного варіантного масиву дорівнює 0.

function VarIsArray(const V: Variant): Boolean;
Повертає True якщо V є варіантним масивом, інакше повертає False.

1.8. Динамічна пам'ять і вказівники

Для змінних більшості розглянутих раніше типів місце в оперативній пам'яті визначається компілятором у процесі компіляції програми. Таке розміщення даних називається **статичним**. При цьому повинна бути заздалегідь відома кількість змінних і їхній тип.

Однак часто доводиться обробляти дані, кількість яких і їхній тип заздалегідь не відомі. У цьому випадку зручно використовувати динамічну пам'ять. **Динамічна пам'ять (купа)** – це та оперативна пам'ять комп'ютера, що виділяється програмі в процесі її виконання. Використання динамічної пам'яті дозволяє заощаджувати ресурси комп'ютера. Наприклад, на початку там можуть бути розміщені одні змінні. Потім вони можуть бути вилучені, а на їхньому місці розміщені інші.

Будь-яка комірка оперативної пам'яті – байт – характеризується власною адресою. Адреси змінних можна зберігати в змінних спеціального типу – **вказівниках**. Вказівники можуть бути типізованими і нетипізованими. Типізований вказівник може зберігати адресу змінної визначеного типу. Описується вказівник, як і будь-яка інша змінна, у розділі var:

```
var
    < ім'я вказівника >:^< тип >;
```

Значок ^ указує, що змінна є вказівником, наприклад:

```
var
    po1,po2:^integer;
    po3:^real;
```

Тут po1 і po2 – вказівники на змінні типу integer, а po3 – вказівник на змінну типу real.

Вказівнику може бути присвоєне порожнє значення nil, наприклад:

```
po1 := nil;
```

означаюче, що змінна po1 ні на що не вказує, тобто не містить адресу якої-небудь комірки пам'яті. Як правило, при запуску програми всі вказівники містять значення nil.

Якщо змінні є вказівниками на однаковий тип, то їм можна присвоювати значення один одного, наприклад,

```
po1 := po2;
```

Для того щоб одержати значення, на яке посилається вказівник, треба виконати операцію роз'їменування вказівника. Для цього треба помістити значок ^ праворуч від вказівника. Наприклад, якщо L – змінна типу integer, то припустимі наступні оператори:

```
po1^ := L;   // у po1 поміщена адреса L
L := po1^+6;   // еквівалентно L := L+6;
```

Адресу будь-який змінної можна одержати за допомогою операції узяття адреси @. Наприклад, приведені вище оператори можна було б замінити еквівалентними

```
po1 := @L;
L := po1^+6;
```

Приклад 1.55.

Скласти програму для знаходження максимального елемента в масивах A(6) і B(11).

Рішення.

У приведеній нижче програмі функція max призначена для пошуку максимального елемента в масиві. Параметр beginning є вказівником на дійсне число й у момент виклику функції повинний містити адресу початку масиву. У параметрі n повинне бути зазначено кількість елементів у масиві.

При пошуку максимального елемента у функції max використовується стандартна процедура inc, що має наступне опис:

procedure Inc(var X [; N: Longint]);

Збільшує змінну X порядкового типу. Наприклад, якщо X – цілого типу, то Inc(X) еквівалентно X := X+1, а Inc(X,N) еквівалентно X := X+N. Якщо X є типізованим вказівником, то Inc(X) збільшує X на величину, що дорівнює довжині одного значення відповідного типу. Inc(X,N) збільшує X на величину, що дорівнює довжині N значень відповідного типу.

Крім того, у програмі використовується процедура input, призначена для введення масиву і яка має такі ж параметри, як і функція max.

Програма.

```
program p1_55;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type preal=^real;
var a:array [1..6] of real;
    b:array [1..11] of real;
function max(beginning:preal; n:integer):real;
var i:integer;
begin
  result := beginning^;
  for i := 2 to n do
    begin
    inc(beginning);
    if beginning^>result then result := beginning^
    end
end;
procedure input(beginning:preal; n:integer);
var i:integer;
begin
  for i:=1 to n do
    begin
    read(beginning^);
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
    inc(beginning);
    end;
    readln
end;
begin
  writeln('Enter array A');
  input(@a,6);
  writeln('Enter array B');
  input(@b,11);
  writeln('Maximum  element of array A is ',max(@a,6):5:2);
  writeln('Maximum  element of array B is ',max(@b,11):5:2);
  readln
end.
```

Результати роботи програми.

```
Enter array A
4.4 5.3 6.7 7.4 6.4 4.7
Enter array B
5.6 7.4 9.5 6.87 4.54 6.54 44.54 74.45 53.2 44.7 23.26
Maximum  element of array A is  7.40
Maximum  element of array B is 74.45
```

У розглянутому прикладі використання вказівників не було зв'язано з виділенням динамічної пам'яті. Пам'ять під будь-яку динамічно розташовувану змінну виділяється процедурою New. Параметром процедури є типізований вказівник, наприклад:

```
New(po1);
```

Процедура виділяє пам'ять під змінну відповідного типу і присвоює вказівнику адресу (посилання), починаючи з якої буде розташовуватися значення змінної.

Процедура Dispose виконує зворотню операцію – звільняє пам'ять, динамічно виділену процедурою New. Параметром процедури Dispose є вказівник, у якому зберігається адреса області пам'яті, виділена раніше процедурою New, наприклад:

```
Dispose(po1);
```

У результаті виконання процедури Dispose звільняється ділянка динамічної пам'яті, але не міняється значення вказівника. Вказівнику, що звільнився, можна присвоїти значення nil, якщо планується його подальше використання.

Приклад 1.56.

Розглянемо використання динамічної пам'яті на простому прикладі обчислення суми трьох чисел.

Рішення.

У програмі виділяється динамічна пам'ять для збереження трьох дійсних значень, уводяться ці значення з клавіатури, обчислюється сума і, нарешті, звільняється динамічна пам'ять.

Програма.

```
program p1_56;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var x1,x2,x3:^real;
begin
  new(x1);
  new(x2);
  new(x3);
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
  writeln('Enter 3 numbers');
  readln(x1^,x2^,x3^);
  x1^ := x1^+x2^+x3^;
  writeln('Summa=',x1^:5:2);
  dispose(x1);
  dispose(x2);
  dispose(x3);
  readln
end.
```

Результати роботи програми.

```
Enter 3 numbers
1.1 3.3 5.5
Summa= 9.90
```

Нетипізований вказівник не зв'язаний з яким-небудь конкретним типом даних. Для опису нетипізованого вказівника служить зарезервоване слово Pointer, наприклад:

```
var  x:Pointer;
```

Нетипізовані вказівники використовують у тих випадках, коли в процесі роботи програми тип і структура даних змінюються. Для роботи з нетипізованими вказівниками варто використовувати наступні процедури:

procedure GetMem(var P: Pointer; Size: Integer);
Резервує за нетипізованим вказівником P фрагмент динамічної пам'яті необхідного розміру Size.

procedure FreeMem(var P: Pointer[; Size: Integer]);
Повертає в динамічну пам'ять фрагмент, що раніше був зарезервований за нетипізованим вказівником P.

Часто використовується також наступна функція:

function SizeOf(X): Integer;
Повертає кількість байтів, зайнятих змінною чи типом X.

Приклад 1.57.

В одновимірних масивах A(6) і B(4) знайти максимальні елементи. Це завдання аналогічне завданню, розглянутому в прикладі 1.55, але для розміщення масивів будемо використовувати динамічну пам'ять.

Рішення.

У програмі визначені два типи – mas і pmas.

Mas – це тип-масив, що містить один елемент дійсного типу. Виявляється, що цього цілком достатньо, щоб розміщати в динамічній пам'яті масиви довільної довжини, оскільки для використання масиву, розташованого в динамічній пам'яті, досить знати адресу комірки пам'яті, починаючи з якої розміщається масив, і тип елементів масиву. Доступ до довільного елемента масиву буде здійснюватися за допомогою індесів. Та обставина, що значення індексів будуть виходити за припустимі границі, для масивів, розташованих у динамічній пам'яті, ніяк не контролюється Delphi.

Pmas – це вказівник на тип mas. Цей тип потрібний, насамперед, для опису формального параметра x у підпрограмах max і input. Якби ми описали параметр x у такий спосіб:

```
x:^mas ,
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

те це було б синтаксичною помилкою.

У програмі використовуються дві підпрограми max і input.

Функція max шукає максимальний елемент у масиві, що складається з n елементів, адреса початку цього масиву міститься в вказівнику x типу pmas.

Процедура input уводить кількість елементів у масиві, самі елементи масиву, розміщає масив у динамічній пам'яті і повертає: x – вказівник на початок масиву, size – розмір динамічної пам'яті, відведеної під масив, n – кількість елементів у масиві.

Динамічна пам'ять виділяється за допомогою стандартної процедури getmem, першим параметром якої є нетипізований вказівник p. Поміщена у p адреса копіюється потім у вказівник x

x := p;

Раніше говорилося, що вказівникам можна присвоювати значення один одного, якщо вони вказують на той самий тип. Нетипізовані вказівники є виключенням з цього правила – вони сумісні з будь-якими типізованими вказівниками. Ця обставина використана в програмі при виділенні і звільненні динамічної пам'яті для масивів.

Програма.

```
program p1_57;
{$APPTYPE CONSOLE}
uses
  SysUtils;
type mas = array [1..1] of real;
     pmas = ^mas;
var a,b:pmas;
    size1,size2,n1,n2:integer;
    pr:pointer;
function max(x:pmas; n:integer):real;
var i:integer;
begin
  result := x^[1];
  for i := 2 to n do
    if x^[i]>result then result := x^[i]
end;
procedure input(var x:pmas;var size,n:integer);
var i:integer;
    p:pointer;
begin
  writeln('Enter size of array');
  readln(n);
  size := n*sizeof(real);
  getmem(p,size);
  x := p;
  writeln('Enter array');
  for i:=1 to n do
    read(x^[i]);
  readln
end;
begin
  input(a,size1,n1);
  input(b,size2,n2);
  writeln('Maximum  element of array A is ',max(a,n1):5:2);
  writeln('Maximum  element of array B is ',max(b,n2):5:2);
  pr := a;
```

```
  freemem(pr,size1);
  pr := b;
  freemem(pr,size2);
  readln
end.
```

Результати роботи програми.

```
Enter size of array
6
Enter array
3.22 5.86 7.44 87.05 4.76 68.22
Enter size of array
4
Enter array
7.89 63.77 94.07 12.33
Maximum  element of array A is 87.05
Maximum  element of array B is 94.07
```

При роботі з динамічною пам'яттю варто враховувати наступне:

1. Якщо динамічна пам'ять була виділена за допомогою процедури New, то звільнення пам'яті здійснюється тільки за допомогою процедури Dispose. Це ж правило поширюється на процедури GetMem і FreeMem.
2. Початкова адреса й обсяг динамічної пам'яті, що звільняється за допомогою процедури FreeMem, повинні бути точно такими ж, як і при виділенні динамічної пам'яті за допомогою процедури GetMem.

Вказівники і динамічна пам'ять використовуються в Delphi дуже інтенсивно. Так, наприклад, параметри-змінні в підпрограмах це не що інше, як вказівники. Вказівником є змінна динамічного масиву, що вказує на початок масиву.

Крім того, у мові Object Pascal є ряд визначених типів вказівників, наприклад, так називаний нультермінальний рядок типу PChar. З типом PChar сумісний масив символів з нульовою нижньою границею, що має нульовий символ (#0) наприкінці. Наприклад, ми можемо написати так:

```
const mas:array [0..5] of char = ‘ABCDE’#0;
var x:PChar;
……
x := @mas;
```

Але можна записати коротше:

```
var x:PChar;
……
x := ‘ABCDE’;
```

Відмінністю між символьним масивом і змінною типу PChar є те, що місце для масиву в пам'яті виділяється статично, тобто на етапі компіляції, а для нультермінального рядка місце в пам'яті виділяється під час роботи програми, тобто динамічно.

Тип PChar використовується в Delphi для сумісності з аналогічними типами, що маються в інших системах програмування й операційній системі Windows, а також при виклику стандартних підпрограм, що мають формальні параметри цього типу.

1.9. Налагодження консольних застосувань

Термін "налагодження" означає виправлення помилок у програмі і забезпечення її правильної роботи. Виникаючі в процесі створення програми помилки класифікуються в такий спосіб:

1. Синтаксичні помилки.

PDF-XChange
Click to buy NOW!
www.docu-track.com

2. Помилки періоду виконання програми.
3. Логічні помилки.

Delphi дозволяє легко знайти і виправити помилки, що виникають як під час компіляції (синтаксичні помилки), так і під час виконання. До складу інтегрованого середовища розроблювача Delphi входить могутній і гнучкий налагоджувач, що дозволяє вам порядково виконувати програму, аналізуючи при цьому вирази і модифікуючи значення змінних. Цей налагоджувач убудований в інтегроване середовище розроблювача Delphi, завдяки чому ви можете редагувати, компілювати і налагоджувати програму, не виходячи з Delphi.

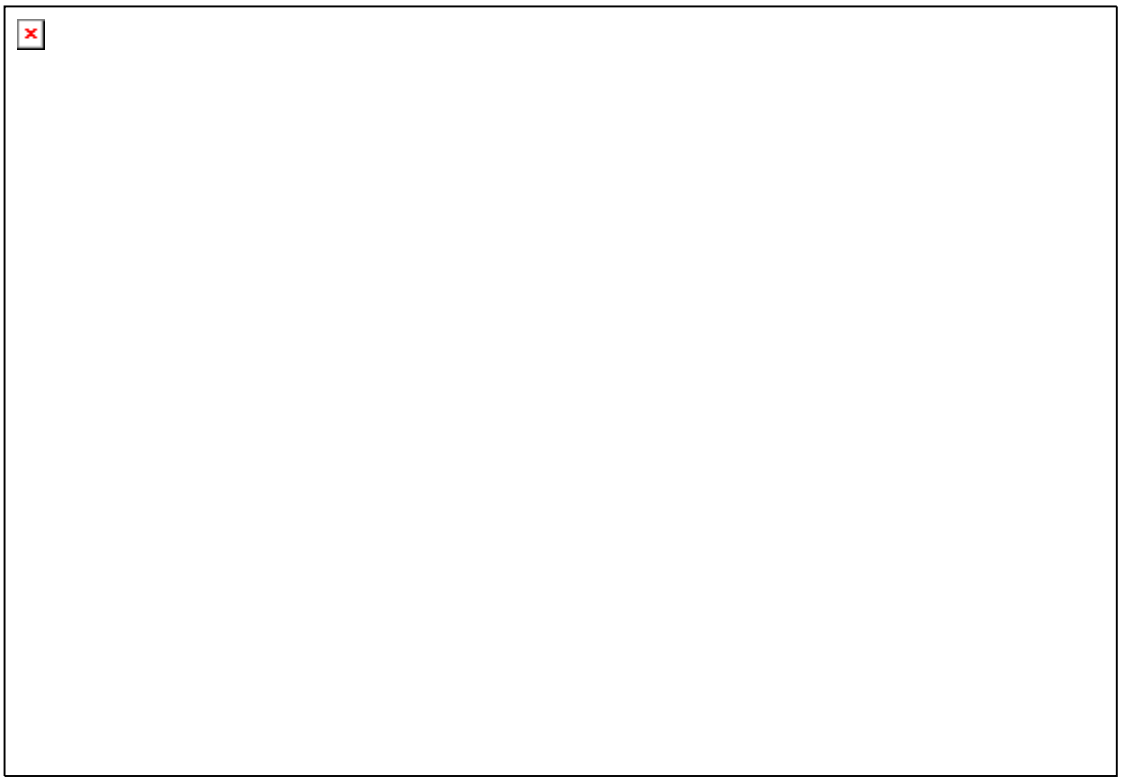
Розглянемо кожен тип помилок.

1.9.1. Синтаксичні помилки

Помилки на етапі компіляції (чи синтаксичні помилки) виникають у тому випадку, якщо ви забудете описати змінну, передасте неправильну кількість параметрів процедурі чи присвоїте дійсне значення цілочисленій змінній. Це означає, що написані вами оператори не задовольняють вимогам мови Object Pascal.

Object Pascal має строгі правила, особливо в порівнянні з іншими мовами, тому, виправивши синтаксичні помилки, ви виконаєте основну частину налагодження.

Delphi не буде виконувати компіляції вашої програми (генерувати машинний код) доти, поки не будуть усунуті всі синтаксичні помилки. Якщо Delphi знайде синтаксичну помилку при компіляції програми, він припинить компіляцію, знайде розташування помилки в програмі, виділить рядок, що містить помилку, коричневим кольором і у вікні редактора коду виведе повідомлення про помилку. Виправивши помилку, ви можете продовжити компіляцію.



Мал. 1.15. Повідомлення транслятора про синтаксичні помилки

На малюнку 1.15 зображене вікно редактора коду в той момент, коли програма була запущена на компіляцію і компілятор знайшов синтаксичні помилки. Нагадаємо, що запуск на компіляцію і виконання програми здійснюється командою Run|Run (функціональна клавіша F9), а запуск програми тільки на компіляцію можна здійснити за допомогою команди Project|Compile < ім'я файлу проекту > (комбінація клавіш Ctrl+F9). У нижній

PDF-XChange Click to buy NOW! www.docu-track.com

частині редактора коду містяться повідомлення про помилки. Повідомлення починається словом [Error] , узятим у квадратні дужки, далі йде ім'я файлу проекту, потім у круглих дужках указується номер рядка програми, де була допущена помилка і, на закінчення, після двокрапки йде текст, що пояснює зміст допущеної помилки. Наприклад, на малюнку 1.15 перше повідомлення інформує нас про те, що в п'ятому рядку програми використовується неописаний ідентифікатор n. Дійсно, n – це константа і повинна бути описана в розділі const. В другому повідомленні говориться про те, що в одинадцятому рядку програми для змінної і, що є параметром циклу, виконується неприпустиме присвоювання. І, справді, у розглянутому фрагменті програми є вкладені цикли, і в цих циклах використовується та сама керуюча змінна, що є грубою помилкою.

Виправлення помилок варто починати з першої, оскільки часто одна помилка є причиною появи інших помилок. Виправивши першу помилку, спробуйте знову запустити програму на компіляцію. У багатьох випадках виправлення тільки однієї помилки істотно зменшує загальну кількість повідомлень про помилки.

Крім повідомлень, що починаються словом [Error] (Помилка), компілятор може видати повідомлення, що починаються словами [Warning] (Попередження) і [Hint] (Зауваження). Попередження і зауваження не є помилками і, незважаючи на їхню наявність, компілятор створить виконуваний модуль. Проте, варто уважно вивчити зроблені компілятором зауваження і попередження, оскільки вони спрямовані на поліпшення вашої програми.

1.9.2. Помилки періоду виконання програми

Інший можливий тип помилок – це помилки етапу виконання (чи семантичні помилки). Це відбувається в тому випадку, якщо ви транслюєте коректну програму, але потім при її виконанні починається спроба виконати неприпустиму дію, наприклад, відкрити неіснуючий файл для уведення чи виконати ділення на 0. У цьому випадку Delphi генерує так називаний виняток (exception).

Консольне застосування, що містить помилку періоду виконання, має одну особливість – DOS-вікно застосування, що містить повідомлення про помилку, з'являється на екрані лише на якусь мить, після чого зникає. Тому прочитати повідомлення практично неможливе. Але, як було сказано вище, помилка періоду виконання генерує виняток, і в нас є можливість побачити повідомлення про цей виняток.

У випадку консольного застосування для того, щоб побачити повідомлення про сгенерований виняток, необхідно виконати нескладне настроювання середовища Delphi.

Виконаємо команду Tools|Debugger Options... . Після цього розкриється вікно Debugger Options, призначене для настроювання параметрів убудованого в середовище Delphi налагоджувача. Розкриємо вкладку OS Exceptions (мал. 1.16) . У вікні Exceptions приведені назви різних винятків, що може генерувати Dephi. Наприклад, виняток Float Divide By Zero генерується в тому випадку, якщо в програмі дійсне число ділиться на нуль.

Нижче вікна Exceptions знаходиться група перемикачів Handled by, що містить перемикачі Debugger і User program. За замовчуванням включеним є перемикач User program, що означає, що сгенеровані Delphi винятки повинні оброблятися програмою, тобто програміст повинний передбачити в програмі оператори для обробки можливих виняткових ситуацій. При створенні консольних застосувань варто зробити активним перемикач Debugger. Це приведе до того, що при запуску програми в середовищі Delphi, налагоджувач візьме задачу обробки виняткових ситуацій на себе й у випадку їхнього виникнення на екран дисплея буде видаватися повідомлення про сгенеровані винятки.

Для того щоб налагоджувач обробляв усі винятки, вони усі повинні бути виділені мишею чи за допомогою клавіатури у вікні Exceptions. І тільки потім варто активізувати перемикач Debugger. У результаті ліворуч від назв винятків з'являться червоні кружечки.

PDF-XChange
Click to buy NOW!
www.docu-track.com

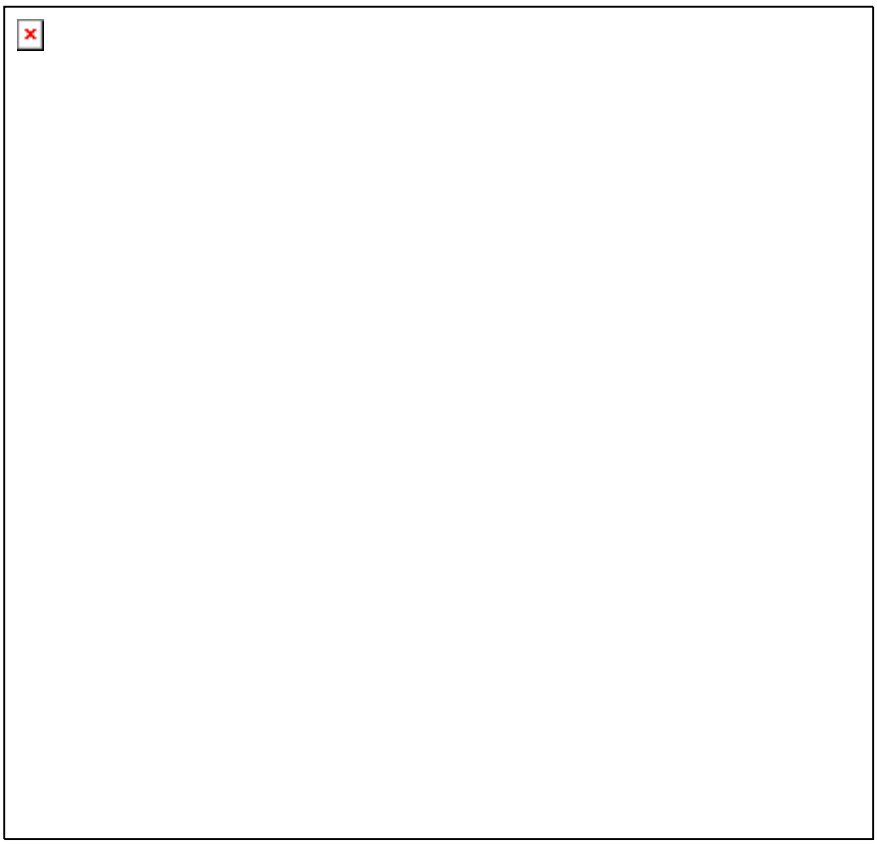


Рис 1.16. Вікно Debugger Options

Виконавши настроювання налагоджувача, ми можемо запустити на виконання, наприклад, таку програму:

```
program otladka;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var x,y:real;
begin
  writeln('Enter x');
  readln(x);
  y := sqr(1/sqrt(1-x*x));
  writeln('y=',y:7:3);
  readln
end.
```

Ця програма буде виконуватися для будь-яких значень x, крім x=1, тому що в цьому випадку відбудеться ділення на нуль. Дійсно, запустивши програму і ввівши значення 1 для x, ми спровокуємо появу виняткової ситуації. На екран буде видане повідомлення про сгенерований виняток (мал. 1.17).

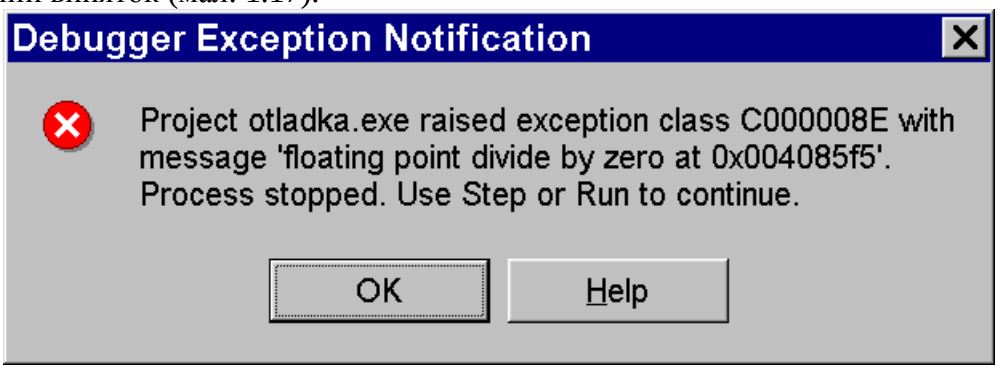
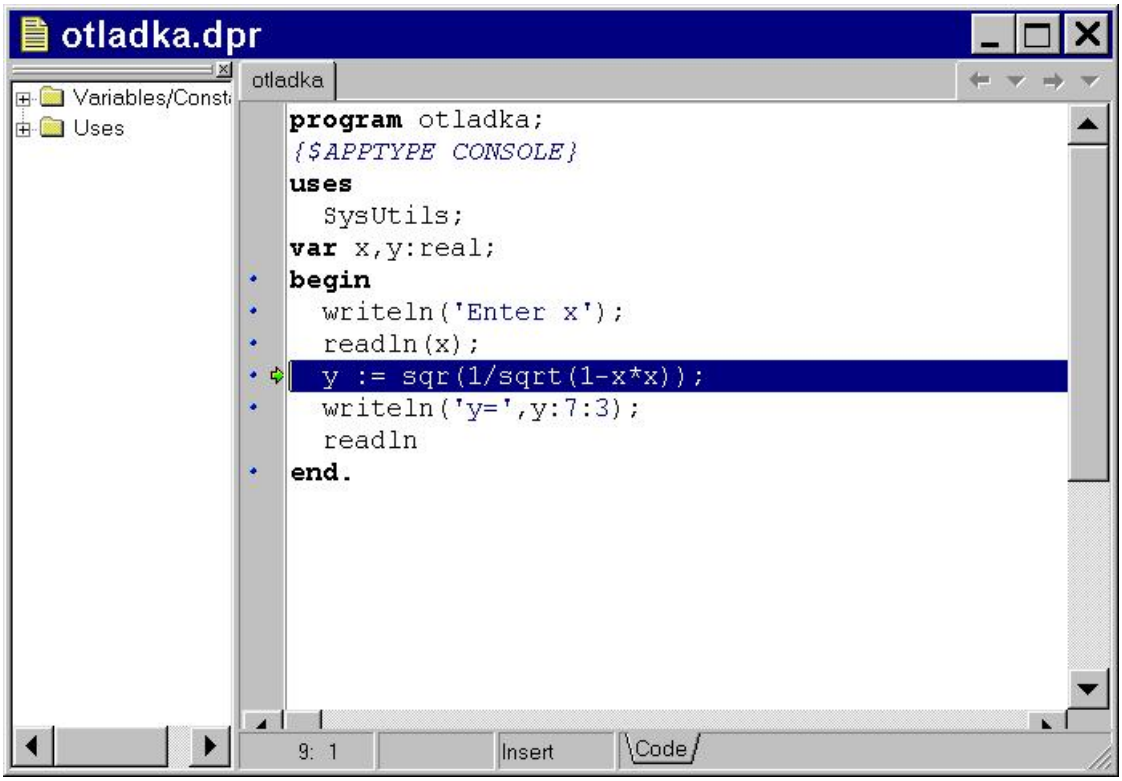


Рис 1.17. Повідомлення про сгенерований виняток.

Повідомлення, що міститься у вікні Debugger Exception Notification, інформує нас про те, що відбулося ділення дійсного числа на нуль і для завершення роботи завислого застосування варто виконати команди Run|Run або Run|Step Over . Додамо, що для переривання налагодження і виконання застосування можна також скористатися комбінацією клавіш Ctrl+F2.

Натиснувши на кнопку OK у вікні Debugger Exception Notification і перервавши виконання програми, ми потрапимо у вікно редактора коду, у якому рядок, що містить помилку, буде виділена синьою смугою (мал. 1.18).



Мал. 1.18. Рядок редактора коду, що містить помилку періоду виконання.

Якщо в цей момент підвести курсор миші до будь-якої змінної, що міститься в тексті програми, то з'явиться віконце, у якому буде зазначене поточне значення змінної. Цю можливість забезпечує наявний у середовищі Delphi 6 Майстер оцінки виразів (ToolTip Expression Evaluation).

Отже, ми з вами навчилися головному – виявляти помилки періоду виконання. Крім того, інформація, що міститься в повідомленні про сгенерований виняток, дозволяє з'ясувати характер помилки, а Майстер оцінки виразів дозволяє одержати додаткову інформацію про поточні значення змінних. Часто цього буває досить, щоб знайти і виправити помилку. Але якщо цього виявляється недостатньо, то варто використовувати більш могутні засоби налагодження, такі як покрокове налагодження і вікно спостереження. Ці засоби розглянуті в наступному пункті.

1.9.3. Логічні помилки

Програма, містить логічні помилки, якщо реалізований у ній алгоритм не є правильним. У цьому випадку програма працює, видає результат, але цей результат невірний. Таке відбувається якщо, наприклад, у програмі неправильно записані формула або при пошуку максимуму з трьох чисел порівняли між собою тільки два числа. Для пошуку й

PDF-XChange Click to buy NOW! www.docu-track.com

усунення логічних помилок необхідно використовувати тести, – вирішені яким-небудь образом, у тому числі і вручну, задачі, що мають правильну відповідь.

У великих і складних програмах логічні помилки і помилки періоду виконання досить важко відстежити і знайти. У цих випадках цілком природним є бажання виконати програму в інтерактивному режимі, спостерігаючи за змінами значень окремих змінних чи виразів. При цьому непогано було б мати можливість зупинятися у визначеному місці програми і дивитися, що там відбувається. Бажано також зупинятися і змінювати значення деяких змінних при виконанні програми. Це дозволить уплинути на її поводження і побачити, у який бік воно змінилося. Усе це хотілося б робити в режимі, що дозволяє швидко відредагувати, перекомпілювати і знову запустити програму.

Усі ці бажані можливості поряд з іншими засобами може вам надати налагоджувач Delphi, що є складовою частиною інтегрованого середовища розробки.

Розглянемо технологію використання налагоджувача Delphi стосовно до програми, у якій обчислюється n! :

```
program p1_58;
{$APPTYPE CONSOLE}
uses
  SysUtils;
var n,i,fact:integer;
begin
  writeln('Enter n');
  readln(n);
  fact := 1;
  for i:= 2 to n do
    fact := fact*i;
  writeln('fact= ',fact);
  readln
end.
```

Перш ніж використовувати налагоджувач, необхідно виконати нескладне настроювання компілятора. Справа в тім, що компілятор Delphi є оптимізуючим. Оптимізація спрямована на підвищення ефективності вашої програми. Наприклад, іноді оптимізація дозволяє скорочувати розмір виконуваного файлу. При цьому компілятор видаляє «зайві», на його думку, змінні чи навіть оператори з тексту програми. Побічним ефектом оптимізації є те, що вона утруднює процес налагодження. Тому перед налагодженням оптимізуючу дію компілятора необхідно відключати. Після завершення налагодження остаточна версія програми повинна бути відкомпільована з оптимізацією.

Для того щоб відключити оптимізацію, необхідно виконати команду Project|Options…, перейти на вкладку Compiler і на панелі Code generation зняти позначку з вимикача Optimization (мал. 1.19). Після цього можна приступати до налагодження.

Включення оптимізації відбувається при додаванні позначки у вимикачі Optimization.

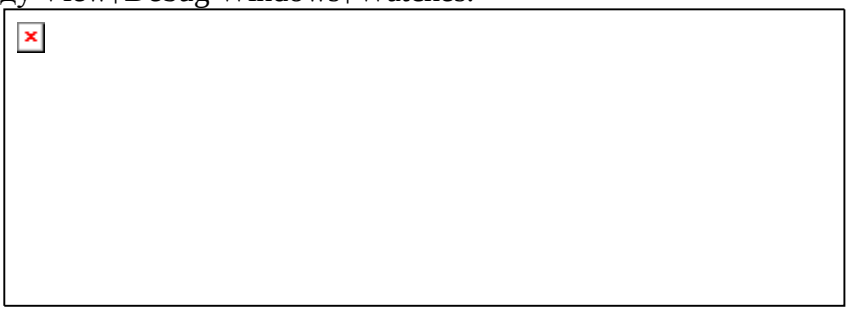
Мал. 1.19. Вікно настроювання властивостей компілятора.

Перейдемо до налагодження програми. Будемо виконувати програму по кроках, оператор за оператором. Якщо говорити ще точніше, будемо виконувати програму по рядках, де в кожнім рядку може розташовуватися один чи більш операторів. Таке виконання програми називається **трасуванням програми**.

У Delphi мається два режими трасування – без заходу в процедуру і з заходом у процедуру. Трасування без заходу в процедуру здійснюються по команді Run|Step Over або при натисканні на функціональну клавішу F8. При цьому трасування підпрограм не здійснюється, а з появою в тексті програми звертань до процедур або функцій відповідні підпрограми виконуються як один оператор.

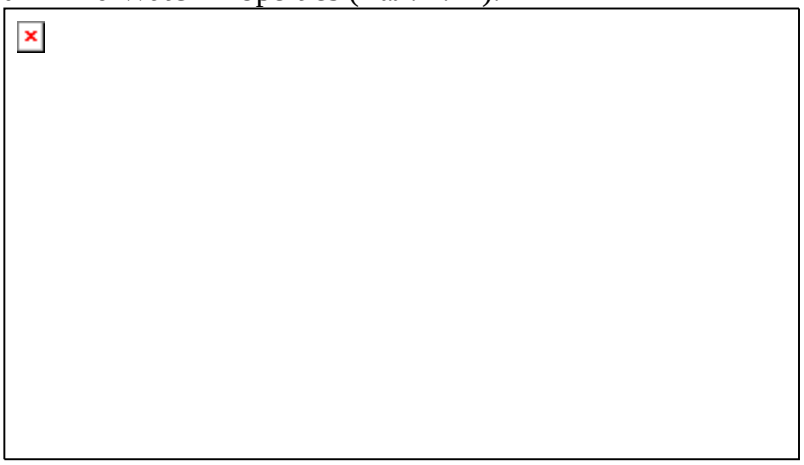
Трасування з заходом у процедуру виконується по команді Run|Trace into або при натисканні на функціональну клавішу F7. У цьому випадку по кроках буде виконуватися не тільки основна програма, але і всі підпрограми.

Для того щоб у процесі трасування спостерігати за зміною значень змінних, наявних у програмі, необхідно розкрити вікно спостереження Watch List (див. мал. 1.20) і внести в нього імена спостережуваних змінних чи виразів. Для відкриття вікна Watch List необхідно виконати команду View|Debug Windows|Watches.



Мал. 1.20. Вікно спостереження

Для додавання нового виразу чи змінної необхідно клацнути по вікну Watch List правою кнопкою миші й у з'явившемуся контекстному меню вибрати пункт Add Watch… . У результаті з'явиться вікно Watch Properties (мал. 1.21).



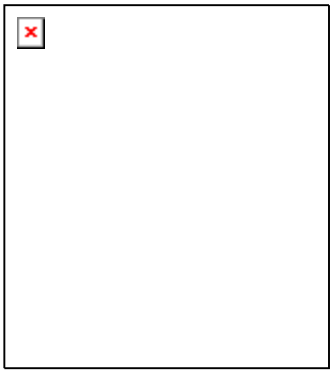
Мал. 1.21. Вікно додавання нового виразу.

Поле Expression вікна Watch Properties призначено для введення імен спостережуваних змінних чи виразів. Поле Repeat count використовується у випадку спостереження масивів і визначає кількість показуваних елементів масиву. Поле Digits служить для завдання кількості значущих цифр при відображенні дійсних значень. Вимикач Enabled дозволяє або забороняє виводити у вікно спостереження значення відповідної змінної чи виразу. Вимикач Allow Function Calls дозволяє або забороняє поміщати у вікно спостереження виразу, що містять виклики функцій. Перемикачі, що містяться в нижній частині вікна, призначені для визначення виду представлення значень різних типів.

Повернемося до нашого приклада і внесемо у вікно спостереження імена змінних n, i та fact. Для цього нам доведеться три рази відкривати вікно Watch Properties і послідовно вводити імена всіх трьох змінних.

Вікно Watch List є вбудовуваним. Це означає, наприклад, що його можна перетягнути на вікно Object Inspector, і вікно спостереження вмонтується. Цю обставину варто використовувати для зручного розміщення вікон на екрані дисплея.

Тепер усе готово для трасування нашої програми. Непоспішаючи натискаємо клавішу F7 чи F8 і стежимо, як змінюються значення змінних у вікні спостереження (див. мал. 1.22).



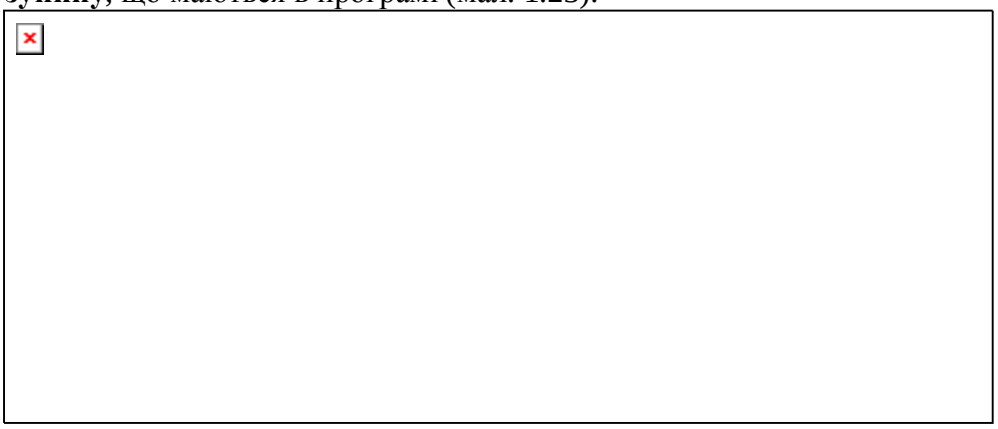
Мал. 1.22. Зміна значень у вікні спостереження.

Якщо необхідно виконати трасування частини програми, то треба установити курсор у редакторі коду на той оператор, з якого варто почати трасування і виконати команду Run|Run to Cursor (функціональна клавіша F4). Потім можна продовжити трасування, натискаючи клавіші F7 чи F8.

Крім цього, при налагодженні програми можна використовувати так називані **точки зупину програми (breakpoint)** . Точками **зупину** програми називаються деякі, спеціальним образом позначені рядки програми. При досягненні точки **зупину** програма припиняє свою

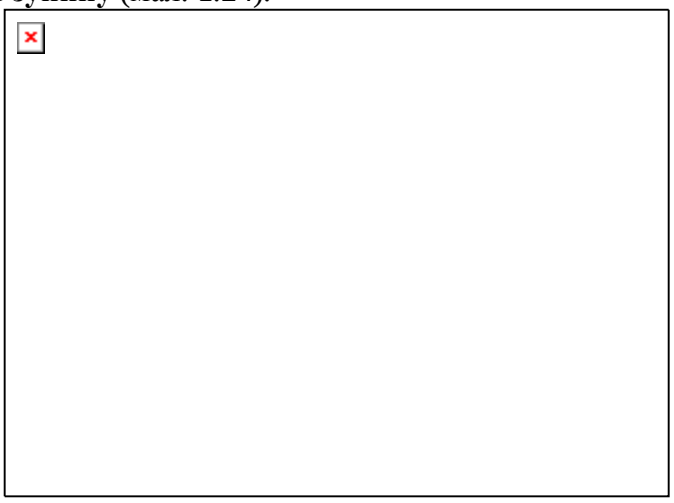
роботу. У цей момент програміст може переглянути значення спостережуваних змінних чи почати трасування програми.

Для того щоб додати точку **зупину**, можна виконати команду View|Debug Windows|Breakpoints. У результаті на екрані з'явиться вікно Breakpoint List, що містить опис усіх точок **зупину**, що маються в програмі (мал. 1.23).



Мал. 1.23. Вікно точок **зупину**.

Далі установимо мишу в межах цього вікна і натиснемо праву клавішу. З'явиться контекстне меню, з якого виберемо пункт Add, а потім у наступному розкривному меню – пункт Source Breakpoint. У результаті з'явиться вікно Add Source Breakpoint, призначене для додавання нової точки **зупину** (мал. 1.24).

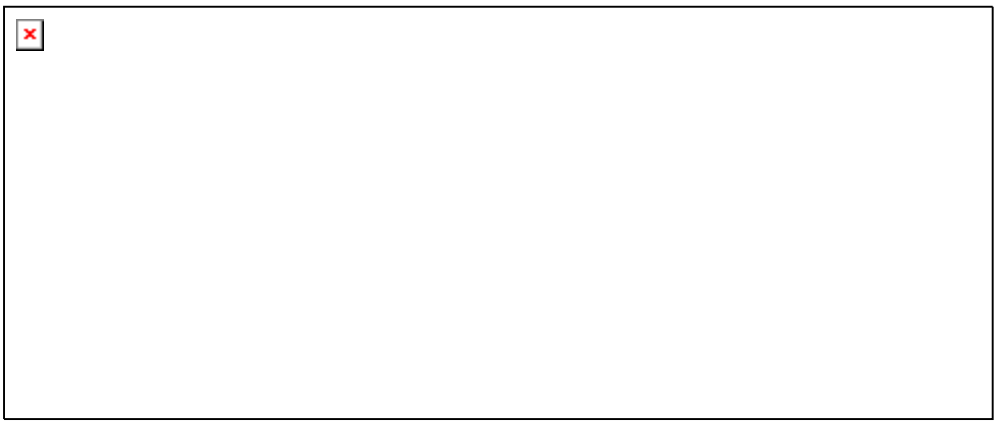


Мал. 1.24. Вікно додавання нової точки.

Поля в цьому вікні мають наступний сенс:
Filename – ім'я файлу, що містить текст програми;
Line number – номер рядка тексту програми, у яку додається точка **зупину**;
Condition – умова, при істинності якого програма буде припиняти свою роботу в точці **зупину**. У противному випадку точка **зупину** буде пропускатися;
Pass count – кількість проходів програми через точку **зупину** без припинення своєї роботи;
Group – ім'я групи, до якої відноситься точка **зупину**.

Об'єднання точок **зупину** в групи дозволяє одночасно дозволити чи заборонити дію всіх точок **зупину**, що відносяться до однієї групи. Для цього потрібно натиснути на кнопку Advanced і в додатковій панелі Actions, що з'явилася, у полях Enable group чи Disable group установити відповідні імена груп.

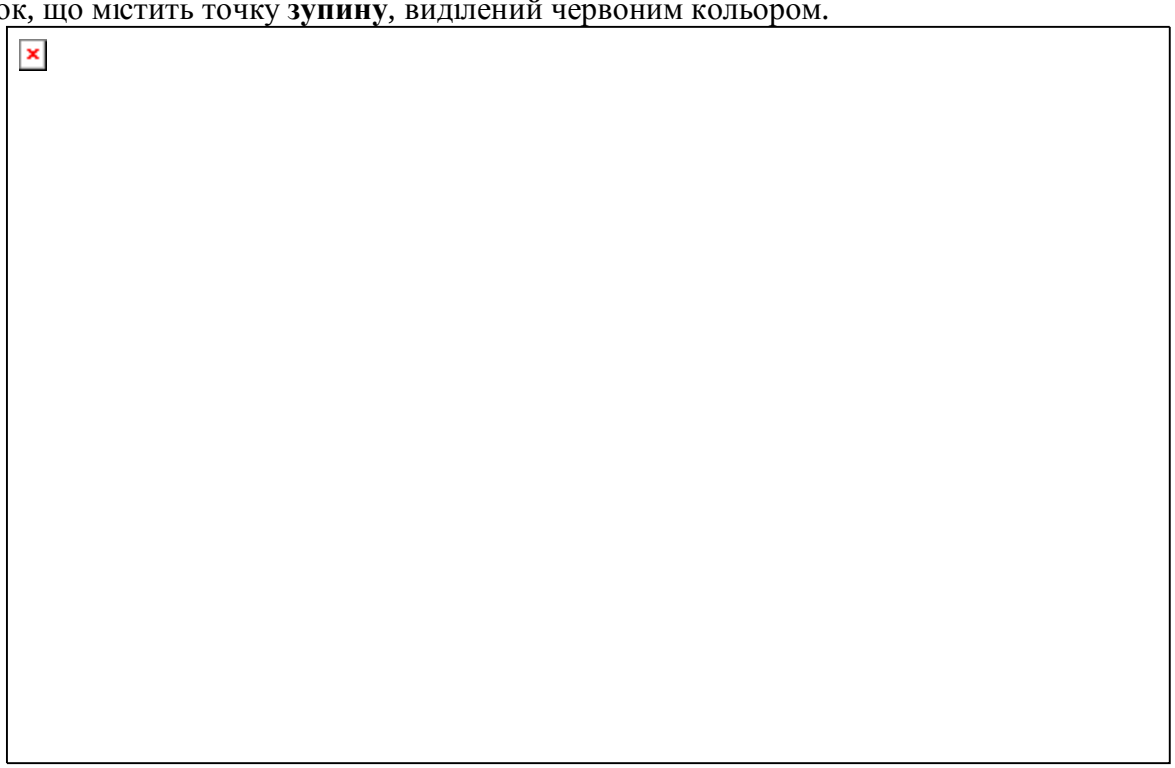
Після успішного заповнення всіх полів, введена інформація буде відбита у вікні точок **зупину** (мал. 1.25).



Мал. 1.25. Вікно Breakpoint List з інформацією про точку **зупину**.

Вікно точок **зупину**, як і вікно спостереження, є вбудовуваним і для зручності його можна помістити у вікно інспектора об'єктів.

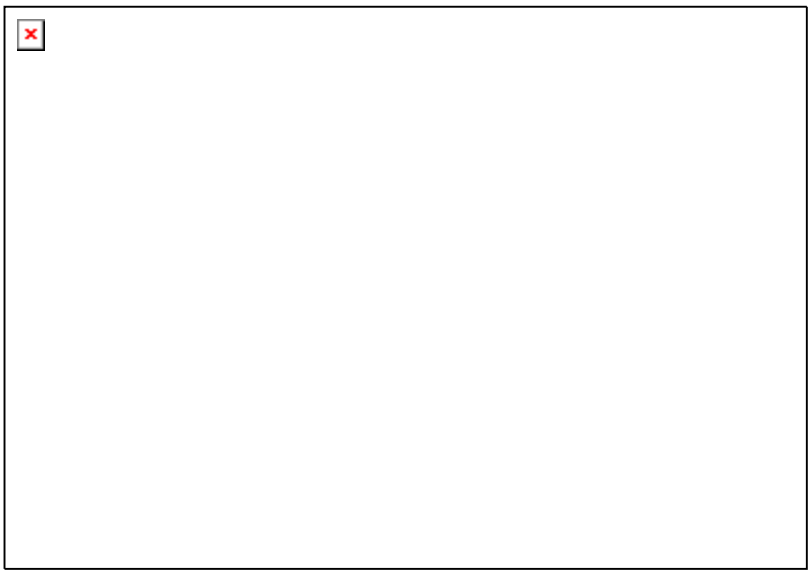
Нижче приведене вікно коду редактора після додавання точки **зупину** (мал. 1.26) . Рядок, що містить точку **зупину**, виділений червоним кольором.



Мал. 1.26. Вікно редактора коду після додавання точки **зупину**.

Установити чи зняти точку **зупину** можна і більш простим способом. Для цього досить клацнути лівою кнопкою миші по службовій зоні ліворуч від обраного рядка або установити текстовий курсор у потрібному рядку і натиснути функціональну клавішу F5.

При виконанні трасування програми, крім спостереження значень змінних програми, можна одночасно і змінювати ці значення. Це робиться за допомогою вікна Evaluate/Modify, що викликається командою Run|Evaluate/Modify… або за допомогою комбінації клавіш Ctrl+F7 (див. мал. 1.27) .



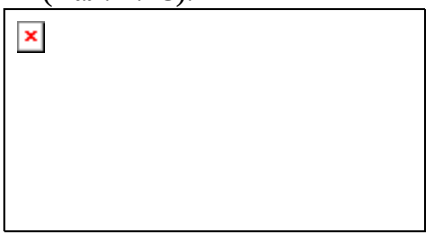
Мал. 1.27. Вікно Evaluate/Modify.

Якщо в процесі виконання трасування відкрити вікно Evaluate/Modify, ввести в поле Expression ім'я змінної чи виразу і натиснути на кнопку Evaluate, то в полях Result і New value відобразяться відповідні поточні значення.

Якщо в поле Expression введене ім'я змінної, то її значення можна змінити в поле New value і натиснути кнопку Modify. Після цього в програмі буде використовуватися нове значення змінної.

Кнопка Watch призначена для переносу виразу чи змінної з вікна Evaluate/Modify у вікно спостереження Watch List.

При натисканні на кнопку Inspect з'являється вікно Debug Inspector, що дозволяє робити дослідження різних даних (мал. 1.28).



Мал. 1.28. Вікно Debug Inspector.

PDF-XChange
Click to buy NOW!
www.docu-track.com

Частина 2. Object Pascal. Створення віконних застосувань

В другій частині підручника розглядається технологія створення віконних застосувань – основного виду застосувань, використовуваних в операційних системах сімейства Windows. Віконними застосуваннями є переважна більшість прикладних програм, створених для роботи в Windows, наприклад, такі програми, як Провідник, текстовий процесор Word, табличний процесор Excel, графічний редактор Paint і багато інших. Кожна така програма створена з урахуванням архітектури операційних систем сімейства Windows, і для її роботи виділяється вікно. На відміну від цього, консольне застосування не є віконним, незважаючи на те, що для його роботи також виділяється вікно, оскільки для виконання консольного застосування эмулюється режим операційної системи MS-DOS.

Розгляд консольних застосувань у першій частині підручника дозволив нам вивчити основи мови програмування Object Pascal. В другій частині вивчення Object Pascal буде продовжено і головна увага буде приділена таким поняттям, як клас, об'єкт і компонент, що безпосередньо зв'язані з процесом створення віконного застосування.

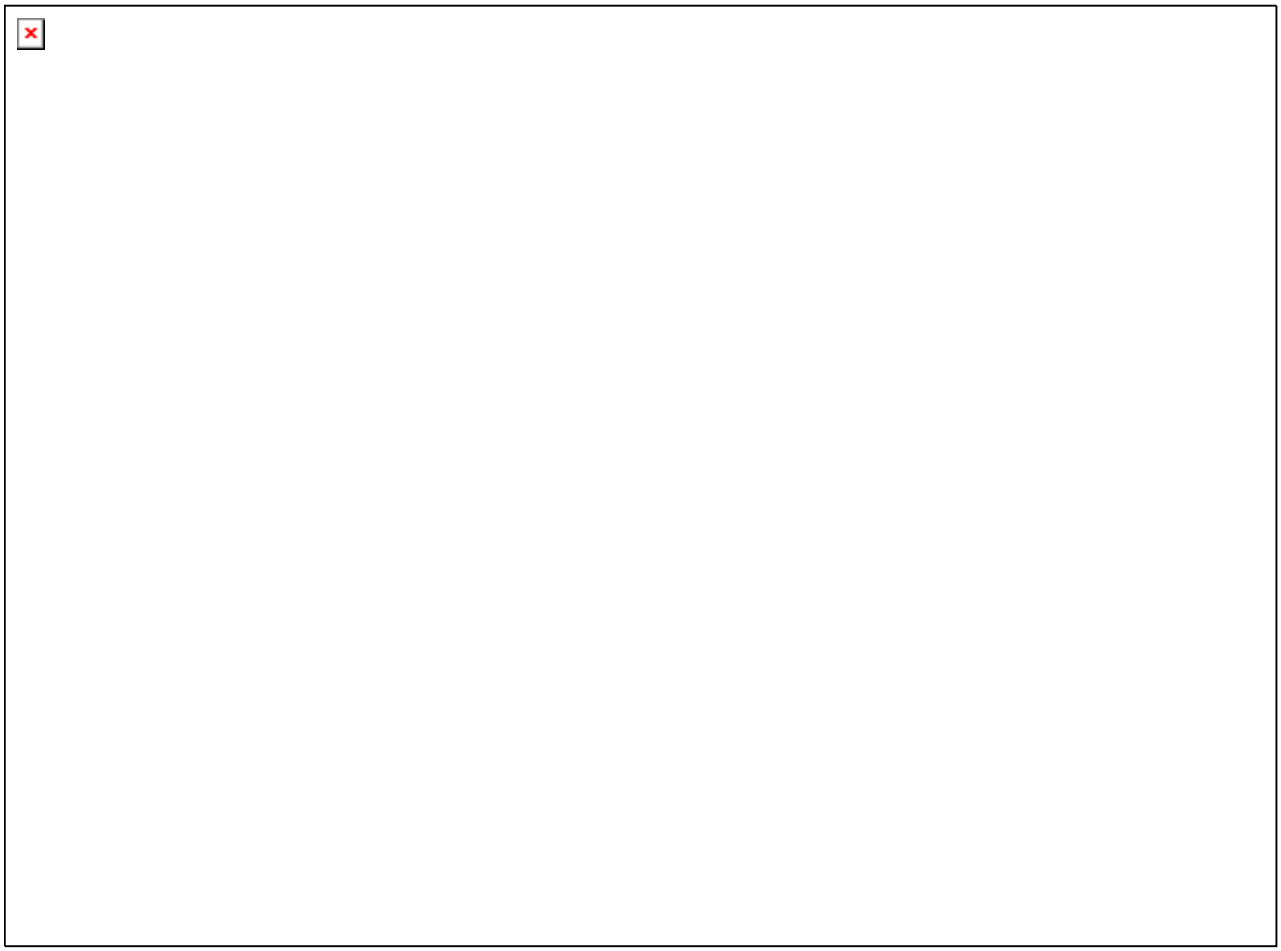
Практично всі знання, що ви одержали в першій частині, придадуться вам і в другій частині. Виключення складають процедури введення-виведення – read, readln, write, writeln, що не використовуються у віконних застосуваннях. Замість них використовуються спеціальні компоненти для введення і відображення текстової інформації.

2.1. Основні поняття

2.1.1. Етапи створення віконного застосування

У пункті 1.1.13 уже згадувалися основні вікна, що використовуються в середовищі програмування Delphi. Прийшов час познайомиться з ними більш докладно.

Після запуску Delphi 6 на екрані комп'ютера можна побачити (див. мал. 2.1):



Мал. 2.1. Вікна і панелі середовища Delphi 6

1 – у верхній частині екрана розташоване головне меню, що забезпечує доступ до команд середовища програмування; головне меню міститься в так називаному головному вікні, що має заголовок : Delphi 6 – Project1;

2 – у головному вікні містяться панелі інструментів, що дозволяють швидко виконувати часто використовувані команди головного меню;

3 – у правій нижній частині головного вікна розташовується Палітра Компонентів, що містить великий набір об'єктів, які можна додавати у форму; саме компоненти є засобом створення застосувань у середовищі Delphi;

4 – на середині екрана знаходиться вікно форми з заголовком Form1, що використовується для розміщення компонентів Delphi;

5 – вікно редактора коду, що містить код програми, за замовчуванням має заголовок Unit1.pas і призначено для створення і редагування коду програми;

6 – вікно Інспектора Об'єктів (Object Inspector) дозволяє змінювати властивості (характеристики) компонентів;

7 – вікно Дерева Об'єктів (Object TreeView) є нововведенням, що з'явилося в Delphi 6; у цьому вікні відображаються компоненти, розміщені на формі, модулі даних чи фреймі, а також логічні відносини, що існують між ними, наприклад відношення батьківський-дочірній (див. п. 2.3.5).

У термінології візуального програмування **об'єктами проекту (компонентами)** є діалогові вікна й елементи керування, що знаходяться в діалогових вікнах – командні кнопки, поля введення текстової інформації, перемикачі, меню і т.д.

Властивостями об'єкта є, наприклад, його розмір (висота і ширина), положення на екрані чи на формі, текст заголовка чи текст на командній кнопці.

Подія – це те, що відбувається під час роботи застосування. У Delphi у кожної події є ім'я. Наприклад, щиглик кнопкою миші – це подія OnClick, подвійний щиглик мишею – подія OnDblClick, при натисканні клавіші виникає подія OnKeyDown, а при відпусканні натиснутої клавіші – OnKeyUp і т.д. Реакцією на подію може бути яка-небудь дія, наприклад, реакцією на подію OnClick, що відбулася по щиглику кнопкою миші на командній кнопці в діалоговому вікні, може бути виконання обчислень по заданих формулах і видача результату. У Delphi реакція на подію реалізується як **процедура обробки події** (інакше – оброблювач події). Таким чином, завдання програміста полягає в написанні необхідних процедур обробки подій.

Укрупненно процес створення найпростішого віконного застосування можна розбити на два етапи: етап конструювання форми й етап програмування. Розглянемо ці етапи більш детально.

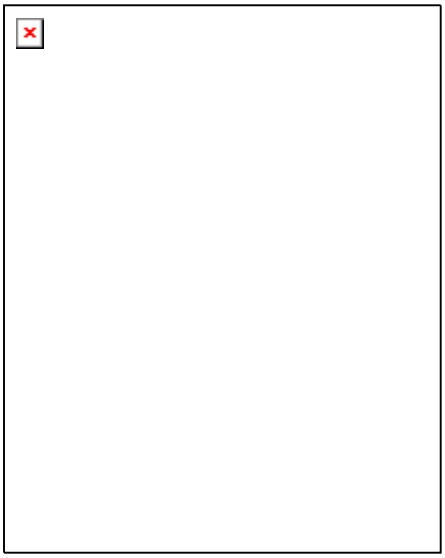
Перелічимо операції, що виконуються на етапі конструювання форми.

1. Насамперед треба вибрати необхідні компоненти з Палітри Компонентів і розмістити їх на формі. Для того щоб розмістити компонент на формі, необхідно виконати щиглик лівою кнопкою миші на компоненті, що знаходиться в Палітрі Компонентів, а потім клацнути лівою кнопкою миші в те місце форми, куди повинний бути поміщений компонент.
2. Поміщений на форму компонентів можна переміщати на формі за допомогою миші і змінювати його розміри, використовуючи маркери, що обрамляють виділений компонент.
3. Для того щоб додати компоненту потрібні властивості, треба використовувати сторінку Properties у вікні Інспектора Об'єктів. Ця сторінка складається з 2-х стовпчиків: лівий стовпчик містить назву властивості, а правий - конкретне значення властивості. Вікно Інспектора Об'єктів відображає інформацію для того компонента, що виділений щигликом миші. Рядки сторінки цього вікна вибираються щигликом миші і можуть відображати прості чи складні властивості. До простих відносяться властивості, що визначаються одним значенням – числом, рядком символів, значенням False чи True і т.д. Складні властивості визначаються сукупністю значень.

Ліворуч від імені таких властивостей указується символ «+». Подвійний щиглик по імені такої властивості приводить до розкриття списку значень складної властивості. Закривається розкритий список також подвійним щигликом миші по імені складної властивості. Цікавим нововведенням у Delphi 6 є додавання в Інспекторі Об'єктів так званих **розширених убудованих компонентних посилань** (expanded inline component references) чи, коротше, убудованих компонентів. Під цим терміном маються на увазі деякі властивості компонентів, значеннями яких є імена інших компонентів (тобто посилання на інші компоненти). Наприклад, у багатьох компонентів є властивість PopupMenu, яка містить ім'я компонента, що є контекстним меню. Властивості, що містять посилання на убудований компонент, відображаються в Інспекторі Об'єктів за замовчуванням червоним кольором. Коли такій властивості привласнюють значення, біля нєї з'являється символ «+». Якщо виконати подвійний щиглик по властивості, що містить ім'я убудованого компонента або просто клацнути по символу «+», то розкриється список властивостей убудованого компонента. Властивості убудованого компонента за замовчуванням відображаються зеленим кольором.

У результаті виконання зазначених операцій буде сформований зовнішній вигляд вікна майбутньої програми. Для того щоб програма виконувала якусь корисну роботу, необхідно перейти до другого етапу – етапу програмування. Розглянемо основні особливості цього етапу.

1. Кожен компонент може реагувати на визначений набір подій. Для того щоб довідатися, на які саме події відгукується компонент, необхідно виділити його щигликом миші на формі, після чого перейти на сторінку Events Інспектора Об'єктів (див. мал.2.2).



Мал. 2.2. Вікно Інспектора Об'єктів

Сторінка містить два стовпчики. У першому стовпчику перераховані імена подій, на які відгукується компонент, друга заповнюється програмістом і містить імена створених оброблювачів подій. Теоретично можна створити оброблювачі для всіх подій, для окремих подій або зовсім не створювати оброблювачів подій для даного компонента. В останньому випадку компонент може бути використаний в оброблювачах подій, створених для інших компонентів. Можна створити оброблювач якоїсь однієї події, а потім на сторінці Events Інспектора Об'єктів указати його ім'я для інших подій, що мають такий же тип (див. пункт 2.2.5). У цьому випадку одна і та ж сама процедура буде виконуватися при виникненні різних подій. Таким чином, на етапі програмування насамперед нам необхідно визначити, на які події повинен відгукуватися той чи інший компонент, і на сторінці Events Інспектора Об'єктів ввести ім'я оброблювача, що створюється.

PDF-XChange Click to buy NOW! www.docu-track.com

2. Оброблювач події являє собою процедуру і має ім'я, що складається з двох частин, розділених крапкою. Перша частина являє собою ім'я класу створюваної форми. Більш докладне поняття класу буде розглянуто в главі 2.2. Друга частина імені створюється або програмістом, як було сказано вище, або створюється середовищем Delphi, якщо програміст не увів своє ім'я. Якщо Delphi автоматично формує другу частину імені для оброблювача, то воно являє собою об'єднання імені компонента й імені події без префікса On, наприклад:

   procedure TForm1.Label1Click(Sender: TObject);

   Тут TForm1 – ім'я класу створюваної форми, Label1Click – друга частина імені оброблювача події, створена автоматично Delphi і означаюча, що процедура буде виконуватися, якщо по компоненті з ім'ям Label1 (мітка) клацнути лівою кнопкою миші. Параметр Sender типу TObject містить посилання на компонент, що створив подію OnClick (тобто на Label1).
3. Після того як ви вказали ім'я оброблювача події на сторінці Events Інспектора Об'єктів або не вказали його, тому що ім'я буде створено автоматично Delphi, необхідно створити заготовку оброблювача події. Для цього в Інспекторі Об'єктів виконаємо подвійний щиглик по полю, призначеному для імені створюваного оброблювача. У результаті у вікно редактора коду буде додана заготовка для оброблювача, наприклад:

```
procedure TForm1.Label1Click(Sender: TObject);
begin

end;
```

   Тіло процедури обмежене зарезервованими словами begin...end, між якими програміст повинний розміщати свої оператори.
4. Серед операторів, що вставляються програмістом, можуть бути і такі, котрі змінюють значення властивостей компонентів, розташованих на формі. Це означає, що властивості компонентів можна змінювати не тільки за допомогою Інспектора Об'єктів на етапі конструювання, але й у процесі виконання програми. Зміна властивостей компонентів у процесі виконання програми називається динамічною зміною властивостей.

Помітимо, що кожен компонент Delphi має одну так називану подію за замовчуванням. Як правило, ця подія, для якої найчастіше складаються оброблювачі. Наприклад, для багатьох компонентів Delphi такою подією є OnClick. Для того щоб створити заготовку оброблювача події за замовчуванням, досить виконати подвійний щиглик мишею по компоненті.

Між умістом вікна форми і вікна редактора коду існує нерозривний зв'язок, що строго контролюється Delphi. Наприклад, розміщення на формі компонента приводить до автоматичної зміни коду програми. Як уже було сказано вище, автоматично створюються також заготовки для оброблювачів подій. Програміст при цьому може наповняти заготовки конкретним змістом – вставляти оператори, додавати опис власних змінних, типів, констант і т.д. При цьому програміст повинний пам'ятати, що йому не можна видаляти з тексту програми ті рядки, що вставило туди середовище Delphi.

2.1.2. Структура проекту Delphi

Проект Delphi – це декілька зв'язаних між собою файлів. Так, будь-який проект завжди складається з уже знайомого нам файлу проекту (такий файл має розширення .dpr) і одного чи декількох модулів (файли з розширенням .pas). Файл проекту не призначений для редагування користувачем і створюється автоматично самою системою програмування Delphi. Для того щоб побачити уміст файлу проекту, необхідно виконати команду Project|View Source. Уміст файлу проекту може бути, наприклад, таким:

```
program Project1;
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.res}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Файл проекту (головний модуль) починається словом program, за яким слідує ім'я програми, що збігається з ім'ям проекту. Ім'я проекту задається програмістом у момент збереження файлу проекту, і воно визначає ім'я створюваного середовищем Delphi виконуваного файлу (файлу з розширенням .exe). Далі за словом uses йдуть імена використовуваних модулів: стандартного модуля Forms і модуля форми Unit1. Схожа на коментар директива {$R *.res} указує компілятору, що потрібно використовувати файл ресурсів, що містить опис ресурсів застосування, наприклад, піктограми. Зірочка вказує, що ім'я файлу ресурсів таке ж, як і у файлу проекту, але з розширенням.res.

Частина головного модуля, що виконується, знаходиться між операторними дужками begin…end. Оператори частини, що виконується, забезпечують ініціалізацію застосування і виведення на екран стартового вікна.

Крім головного модуля кожна програма включає як мінімум один модуль форми, що містить опис стартової форми застосування і підтримуючих її роботу процедур. У Delphi кожній формі відповідає свій модуль. Для переключення між формою і вікном редактора коду, що містить відповідний модуль, треба виконати команду головного меню View|Toggle Form/Unit, або натиснути функціональну клавішу F12, або на панелі інструментів View клацнути мишею по кнопці  .

Модулі – це програмні одиниці, що служать для розміщення фрагментів програм. За допомогою текстів програм, що містяться в них, (програмних кодів) і реалізується розв'язувана користувачем задача.

Модулі мають стандартну конструкцію (послідовність і перелік розділів), передбачену мовою програмування Object Pascal. Приведемо структуру модуля в загальному виді:

```
unit < ім'я модуля >;
interface
……..
implementation
……..
initialization
……..
finalization
……..
end.
```

Як приклад приведемо вміст модуля в тому виді, у якому він знаходиться відразу після запуску середовища Delphi:

```
unit Unit1;

interface

uses
```

```
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

end.
```

Починається модуль словом unit, за яким слідує ім'я модуля. Саме це ім'я згадується в списку використовуваних модулів в операторі uses головного модуля застосування.

Модуль може складатися з чотирьох розділів: інтерфейсу, реалізації, ініціалізації і завершальної частині.

Розділ інтерфейсу (починається словом interface) повідомляє компілятору, які дані, що розташовуються в модулі, є доступними для інших модулів програми. У цьому розділі перераховані (після слова uses) стандартні модулі, використовувані даним модулем, а також знаходиться сформований Delphi опис типу форми, який слідує за словом type (див. главу 2.2).

Розділ реалізації починається словом implementation і містить оголошення локальних змінних, процедур і функцій, що підтримують роботу форми. На початку розділу реалізації розташовується директива {$R *.dfm}, що вказує компілятору, що в розділ реалізації треба вставити команди установки значень властивостей форми, що знаходяться у файлі з розширенням .dfm, ім'я якого збігається з ім'ям модуля. Файл у форматі dfm генерується Delphi на основі зовнішнього вигляду форми.

За директивою {$R *.dfm} розташовуються описи процедур обробки подій форми. Сюди ж програміст може помістити опис своїх процедур і функцій, що можуть викликатися з процедур обробки подій.

Ініціююча і завершальна частини є необов'язковими.

Ініціююча частина починається словом initialization або береться в операторні дужки begin...end . Оператори з цієї частини виконуються до передачі керування основній програмі і звичайно використовуються для підготовки її роботи.

Завершальна частина починається словом finalization і містить операторы, що виконуються в момент закінчення програми.

У приведеному вище прикладі модуля ініціююча і завершальна частини відсутні.

На відміну від файлу проекту, створюваного автоматично Delphi, модуль може змінюватися (редагуватися) програмістом. При створенні користувачем нової форми, автоматично буде створюватися і новий модуль. Програма може містити до декількох десятків форм. Текст модуля при цьому буде доступний і користувачу, і самому середовищу Delphi, що автоматично буде вставляти в текст модуля опис будь-якого доданого до форми компонента, а також створювати заготовки (рядки коду) для оброблювачів подій. Програміст при цьому може додавати свої методи в раніше оголошені класи, наповняти оброблювачі подій конкретним змістом, уставляти власні змінні, типи, константи і т.д. Але, як уже було

сказано раніше, програмісту не можна видаляти рядки, вставлені в текст модуля інтегрованим середовищем Delphi.

При компіляції програми Delphi створює файли з розширеннями .dcu для кожного модуля.

Таким чином, pas-файл містить програмний код модуля, що був сформований у вікні редактора коду спільними зусиллями програміста і середовища Delphi, у файлі з расширением .dfm зберігається опис умісту вікна форми, а в dcu-файлі знаходиться результат перетворення тексту з обох файлів у машинні інструкції. Компоновщик, що входить в інтегроване середовище Delphi, перетворить dcu-файли в єдиний завантажувальний (виконуваний) exe-файл. Виконуваний файл дозволяє запускати програму як автономне застосування.

2.1.3. Приклад створення простого віконного застосування

Проілюструємо все сказане вище на прикладі створення простого віконного застосування.

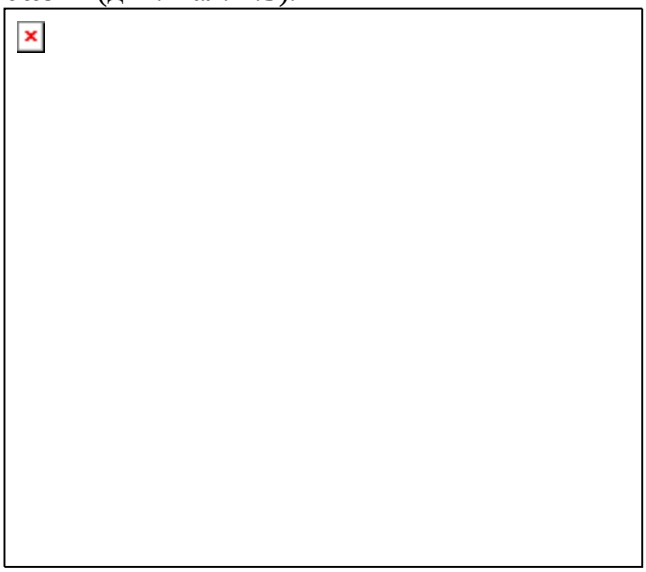
Приклад 2.1.

Скласти програму для обчислення площі кругу довільного радіуса.

Рішення.

Приведемо покроковий опис процесу створення віконного застосування.

1. Насамперед потрібно створити нову папку, у яку ви помістите усі файли, що входять у проект. Надалі завжди будемо створювати окрему папку для кожного нового проекту. Помітимо, що створити нову папку можна також і в момент збереження файлів, що входять у проект, використовуючи кнопку **Створення нової папки**, що присутня в діалоговому вікні збереження файлів.
2. Для створення нового проекту можна скористатися командою головного меню File|New|Application або, якщо ви тільки що завантажили середовище Delphi, можна відразу приступати до створення нового проекту.
3. Зі сторінки Standard Палітри Компонентів помістимо на форму Form1 чотири компоненти: Label1, Label2, Edit1 і Button1 (див. мал. 2.3).



Мал. 2.3. Розташування компонентів Label1, Label2, Edit1 і Button1 на формі

При поміщенні компонентів на форму можна відразу ж задавати їхні розміри. Для цього послу вибору компонента в Палітрі Компонентів треба вказати на формі прямокутну область, що займе компонент. Лівий верхній кут області визначається щигликом лівої клавіші миші, потім, не відпускаючи клавіші миші, потрібно перемістити вказівник миші в правий нижній кут області.

Більш докладно використовувані компоненти будуть розглянуті в наступних главах, а поки обмежимося короткою інформацією про них.

Label1, Label2 – мітки – компоненти, призначені для відображення текстової інформації. Відображуваний ними текст може бути розміщений у них як на етапі конструювання форми, так і на етапі виконання програми. У мітку Label1 ми помістимо текст «Уведіть радіус окружності», а в мітку Label2 на етапі виконання програми будемо поміщати обчислену площу кругу.

Edit1 – рядок уведення – компонент, призначений для введення символьного рядка. За допомогою цього компонента, ми будемо вводити значення радіуса. Оскільки дані будуть вводиться у виді символьного рядка, у програмі необхідно передбачити перетворення рядка в дійсне число.

Button1 – кнопка – компонент, призначений для формування події при натисканні на нього. У нашій програмі при натисканні на кнопку буде відбуватися обчислення площі кругу.

4. Перейдемо до формування зовнішнього вигляду нашої майбутньої програми. Для цього, використовуючи Інспектор Об'єктів, установимо властивості форми і розташованих на ній компонентів. Почнемо з форми.

Зробимо форму активною, клацнувши мишею по будь-якому її місцеві, не зайнятому іншими компонентами. Надалі завжди будемо мати на увазі, що, змінюючи властивості того чи іншого компонента в Інспекторі Об'єктів, треба попередньо виділили його мишею.

Виберемо в Інспекторі Об'єктів властивість Caption (Заголовок) і замість тексту «Form1» уведемо текст «Обчислення площі кругу». У такий спосіб ми змінимо заголовок нашої форми.

Помітимо, що значеннями властивості Caption є символьні рядки, тобто дані типу string. Тому надалі значення властивості Caption, а також інших властивостей типу string, ми будемо брати в одинарні прямі лапки, як це прийнято в Object Pascal. Наприклад: 'Form1', 'Обчислення площі кругу'.

Змінимо розміри форми і, отже, вікна майбутньої програми. Для цього можна підвести курсор миші до будь-якого краю форми (курсор при цьому прийме вид двунаправленної стрілки) і, не відпускаючи лівої клавіші миші, змінити розміри форми. При цьому автоматично будуть змінюватися властивості Height (Висота) і Width (Ширина), що знаходяться в Інспекторі Об'єктів. Це означає, що ми могли б змінити розміри форми, указавши значення властивостей Height і Width безпосередньо в Інспекторі Об'єктів. Але, з іншого боку, перевага візуального програмування саме і полягає в тім, щоб безпосередньо спостерігати за процесом проектування форми.

Використовуючи любий з зазначених прийомів, покладемо властивості Height і Width форми рівними відповідно 350 і 400 (пікселей).

Властивості Left і Top задають відстань від лівого верхнього кута екрана дисплея до лівого верхнього кута форми по горизонталі і вертикалі відповідно. Установимо їхні значення рівними 300 і 200 (пікселей) відповідно. Для цього ми можемо скористатися Інспектором Об'єктів або переміщати форму, схопивши мишею за її заголовок. У результаті наша форма буде знаходитися приблизно на середині екрана.

5. Установимо властивості компонентів Label1 і Label2. Компоненти Label1 і Label2 також мають властивості Height, Width, Left і Top. Але, у відмінності від форми, властивості Left і Top указують відстань від лівого кута форми, на якій розташовується компонент, до лівого верхнього кута самого компонента. Приведемо значення цих властивостей для компонентів Label1 і Label2:

	Label1	Label2
Height	57	будь-яке число
Width	129	будь-яке число
Left	131	100
Top	34	218

Значення властивостей Height і Width для компонента Label2 можна вибрати довільними, і далі ми роз'яснимо чому.

Як і для форми, ці значення можна установити, переміщаючи компоненти по формі і змінюючи їхні розміри за допомогою миші або удавшись до допомоги Інспектора Об'єктів. Крім того, можна скористатися панеллю інструментів Align (Вирівняти), зображену на малюнку 2.4.

Мал. 2.4. Панель інструментів Align

Цю панель можна викликати на екран дисплея, виконавши команду View|Alignment Palette головного меню Delphi. Кнопки, зображені в цій панелі, дозволяють вирівнювати виділені компоненти усередині форми. Наприклад, кнопка  центрує виділений компонент щодо вертикальної осі симетрії форми, а кнопка  центрує виділений компонент щодо горизонтальної осі симетрії форми. Вирівнювати можна не тільки один компонент, але кілька компонентів одночасно. Для того щоб виділити групу компонентів, необхідно при виділенні утримувати натиснутої клавішу Shift. Вирівнювання компонентів автоматично змінює значення властивостей Left і Top в Інспекторі Об'єктів.

Властивість Caption є основною для мітки і містить відображуваний нею текст. Для мітки Label1 установимо властивість Caption рівниною 'Уведіть радіус круга і натисніть кнопку Лічба'. Для мітки Label2 властивість Caption буде визначатися на етапі виконання програми.

Властивість AutoSize мітки визначає, чи буде розмір мітки встановлюватися автоматично, у залежності від довжини символьного рядка, поміщеного туди. Якщо так, то властивість AutoSize варто установити рівною True, у противному випадку – False.

Властивість WordWrap (Перенос слів) дозволяє або забороняє перенос слів, якщо рядок не вміщується в мітку і властивість AutoSize дорівнює False. У першому випадку властивість WordWrap повинна бути встановлена рівною True, у другому – False.

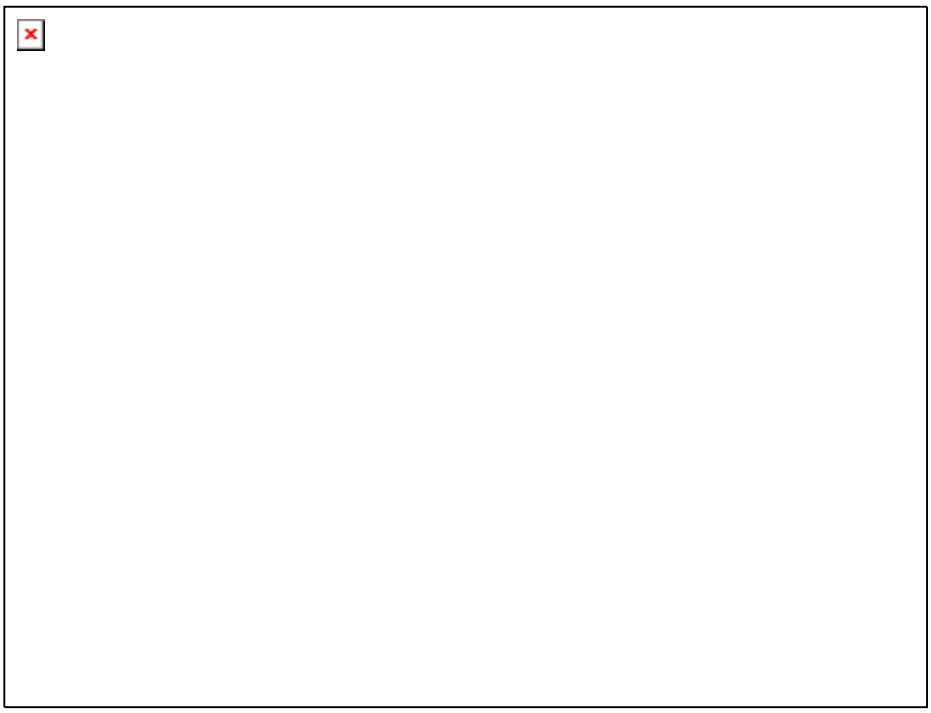
Властивість Alignment (Вирівнювання) визначає, як буде вирівняний текст усередині мітки: по лівому краю, по центрі чи по правому краю.

Приведемо значення цих властивостей для міток Label1 і Label2:

	Label1	Label2
AutoSize	False	True
WordWrap	True	False
Alignment	taCenter	taLeftJustify

Помітимо, що як тільки ми для мітки Label2 поклали значення властивості AutoSize рівною True, розміри компонента, тобто значення властивостей Height і Width, будуть визначатися розмірами поміщеного в нього символьного рядка. Як було сказано вище, символьний рядок буде привласнюватися властивості Caption мітки Label2 на етапі виконання програми, а зараз для поліпшення зовнішнього вигляду майбутньої програми можна для властивості Caption видалити значення 'Label2' і як значення привласнити символьний рядок, що містить кілька пробілів – ' ', щоб компонент Label2 був краще видний на формі.

Властивість Font є складною властивістю і визначає різні характеристики, що описують шрифт, використовуваний при відображенні текстів, наприклад висоту шрифту, його ім'я, накреслення і т.д. Для того щоб установити характеристики шрифту, виберемо щигликом миші властивість Font в Інспекторі Об'єктів і клацнемо по кнопці з трьома крапками, що з'явилася в правому стовпчику –  . На екрані з'явиться вікно Вибір шрифту (див. мал. 2.5).



Мал. 2.5. Діалогове вікно «Вибір шрифту»

За допомогою цього вікна установимо наступні характеристики шрифтів для міток Label1 і Label2:

	Label1	Label2
Шрифт	Times New Roman	Arial
Накреслення	Напівжирний	Курсив
Розмір	10	11

Помітимо, що після зміни характеристик шрифтів для міток Label1 і Label2 наявна в них властивість ParentFont поміняла своє значення з True на False. Властивість ParentFont визначає, чи буде для даного компонента використовуватися шрифт (властивість Font) батьківського компонента-контейнера. Для розглянутих компонентів батьківським компонентом є форма, що містить їх. Якщо властивість ParentFont компонента дорівнює True, то в компоненті використовується шрифт, характеристики якого задані в компоненті-батькові. У противному випадку батьківський і дочірній компоненти мають різні характеристики своїх шрифтів.

6. Властивості Height, Width, Left і Top для компонентів Edit1 і Button1 мають той же зміст, що і для компонентів Label1 і Label2. Приведемо значення цих властивостей для компонентів Edit1 і Button1:

	Edit1	Button1
Height	21	25
Width	193	75
Left	31	285
Top	146	146

Властивість AutoSize, яка є в компонента Edit1, визначає, чи буде рядок уведення збільшувати свою висоту, якщо зміниться висота шрифту в тексту, розміщеного в компоненті. Якщо ми хочемо, щоб рядок введення автоматично збільшувався, значення властивості варто покласти рівним True, у противному випадку – False. У нашому прикладі залишимо для визначеності значення, задане за замовчуванням – True.

Властивість Text є основною для компонента Edit1 і призначено для введення (чи рідше – для виведення) символьних рядків. За замовчуванням значенням цієї властивості є ім'я компонента – 'Edit1', але для поліпшення зовнішнього вигляду програми задамо як значення цієї властивості порожній рядок – ' '.

Для компонента Button1 як значення властивості Caption покладемо символьний рядок 'Лічба'.

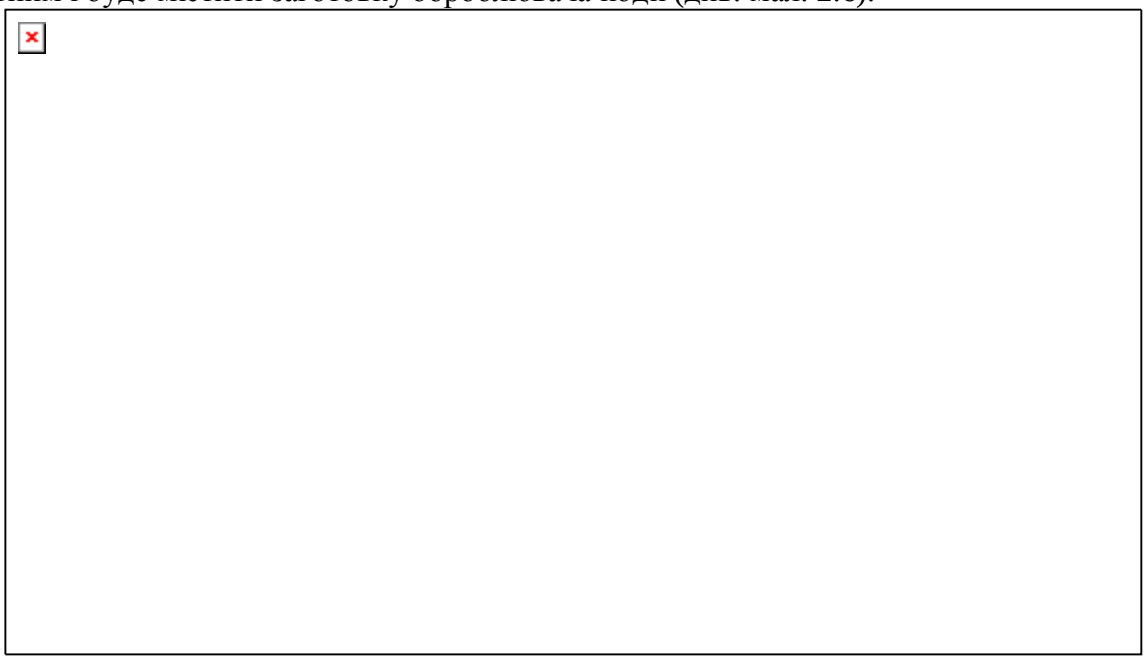
7. Усі використовувані в прикладі компонента – форма, мітки, рядок введення і кнопка – мають імена, що задаються в наявної в них властивості Name (Ім'я): Form1, Label1, Label2, Edit1, Button1. Імена компонентів генеруються автоматично інтегрованим середовищем Delphi при створенні компонента, тобто, наприклад, при розміщенні компонента на формі. Для утворення імені компонента використовується ім'я класу (про класи поговоримо в главі 2.2) з відкинутою першою буквою Т. Наприкінці імені додається цифра, що вказує, під яким порядковим номером у своєму класі з'явився на світ компонент. Наприклад, компонент Edit1 є екземпляром класу TEdit, а компоненти Label1 і Label2 є екземплярами класу TLabel.

Програміст за своїм розсудом може залишити імена, сгенеровані Delphi, а може дати компонентам свої імена. Для великих застосувань, очевидно, доцільно придумувати компонентам якісь осмислені імена. Для розглянутих у цій частині підручника прикладів на нашу думку, з пізнавальної точки зору, зручніше залишити імена сгенеровані Delphi.

На цьому етап конструювання форми можна вважати завершеним.

8. Переходимо до етапу написання коду програми.

Наша майбутня програма повинна обчислювати площу кругу після того, як у рядку уведення набрано значення радіуса і виконаний щиглик мишею по кнопці Лічба. Це означає, що ми повинні написати оброблювач події OnClick для кнопки Button1. Подія OnClick для кнопки є подією за замовчуванням, тому, щоб створити заготовку оброблювача події, досить виконати подвійний щиглик по кнопці Button1. У результаті вікно редактора коду стане активним і буде містити заготовку оброблювача події (див. мал. 2.6).



Мал. 2.6. Вікно редактора коду, що містить заготовку оброблювача події

Додамо в заготовку код для обчислення площі кругу:

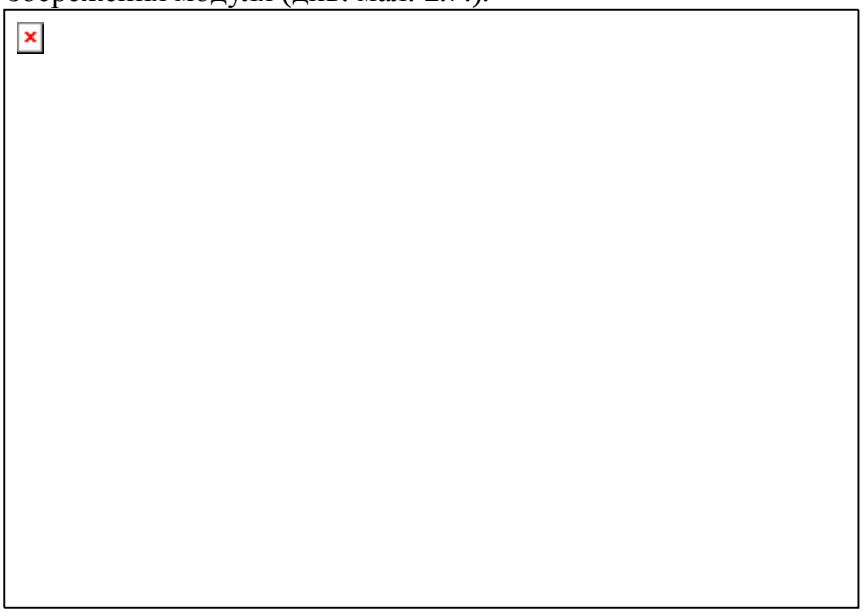
```
procedure TForm1.Button1Click(Sender: TObject);
var r,s:real;                                    //*
begin
  r := StrToFloat(Edit1.Text);                   //*
  s := pi*sqr(r);                                //*
  label2.Caption := 'Площа кругу дорівнює '+     //*
    FloatToStr(s,ffGeneral,7,2)                  //*
end;
```

Зірочка як коментар додана в рядки, уставлені програмістом. Зверніть увагу, як використовуються властивості компонентів у тексті програми. Для того щоб привласнити

властивості або витягти з властивості яке-небудь значення, варто вказати ім'я компонента, поставити крапку і потім указати властивість.

На цьому етап програмування можна вважати завершеним.

9. Перед запуском програми її завжди необхідно зберегти. У випадку зависання неналагодженої програми це допоможе зберегти ваш час. Для збереження проекту необхідно виконати команду головного меню File|Save All. У результаті на екрані дисплея з'явиться діалогове вікно збереження модуля (див. мал. 2.7.).



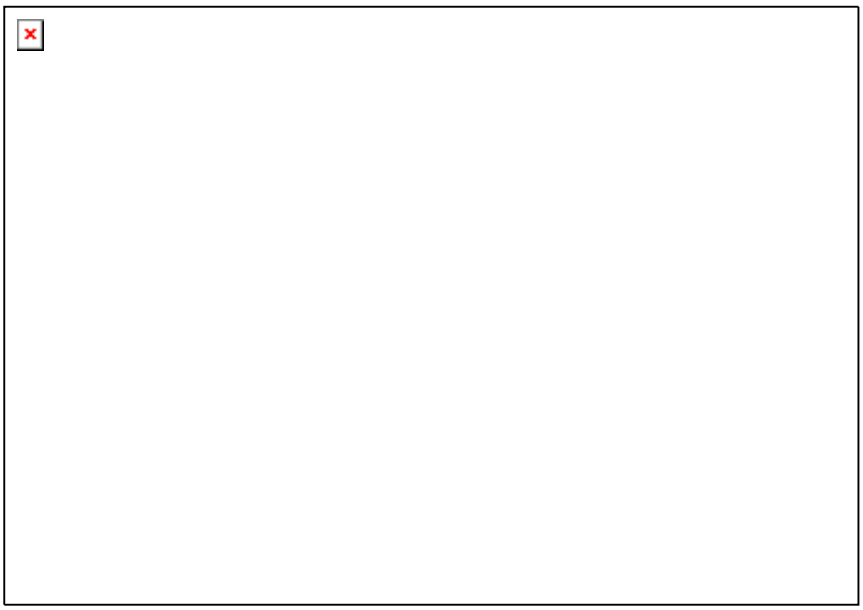
Мал. 2.7. Діалогове вікно збереження модуля

Далі за допомогою списку **Папка**, що розкривається, потрібно відшукати папку, створену заздалегідь для розроблювального проекту, і відкрити її. Якщо папка не створена, її можна створити за допомогою кнопки  – створення нової папки. Після того як потрібна папка відкрита, у поле **Ім'я файлу** варто вказати ім'я модуля, що зберігається. За замовчуванням модуль має ім'я Unit1. При бажанні можна замінити його більш осмисленим, але в цьому прикладі ми залишимо його без зміни. Завершимо збереження модуля щигликом по кнопці **Зберегти**.

Після цього з'явиться діалогове вікно збереження файлу проекту, цілком аналогічне попередньому (див. мал. 2.8.). За замовчуванням файлу проекту дається ім'я Project1. Оскільки це ж ім'я буде згодом привласнено виконуваному файлу, то тут бажано придумати що-небудь більш оригінальне, наприклад, area_of_circle чи, на худий кінець, p2_1.

Помітимо, що імена файлу проекту і модуля повинні бути різними.

PDF-XChange
Click to buy NOW!
www.docu-track.com



Мал.2.8. Діалогове вікно збереження файлу проекту

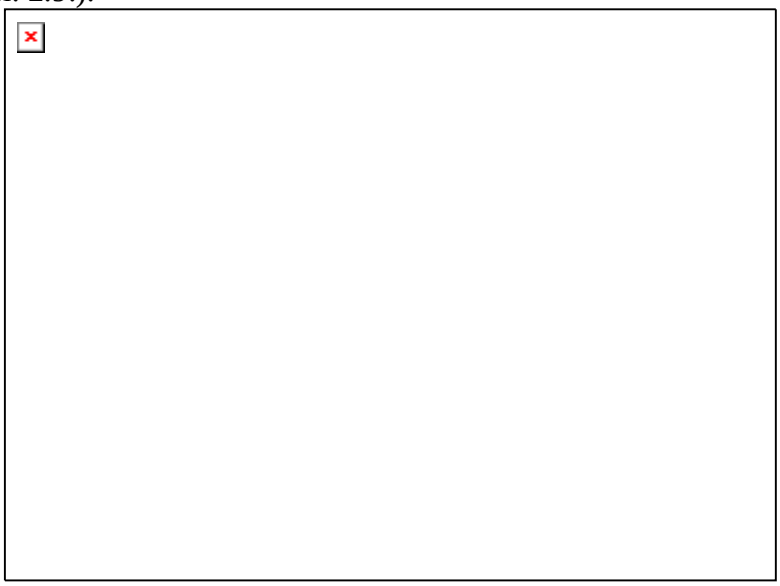
У головному меню Delphi є також інші команди, призначені для збереження:

Save – зберегти поточний модуль;

Save As – зберегти поточний модуль під новим ім'ям;

Save Project As – зберегти поточний проект під новим ім'ям.

Зберігши проект, запустимо його на виконання. Якщо будуть виникати помилки, наприклад, синтаксичні, то виправимо їх точно так само, як це ми робили для консольного застосування. У випадку відсутності помилок на екрані дисплея з'явиться вікно нашої програми (див. мал. 2.9.).



Мал. 2.9. Вікно програми обчислення площі кругу

Уведемо довільний радіус у рядок введення і натиснемо кнопку Лічба – у нижній частині вікна з'явиться відповідна площа кругу.

Приведемо текст модуля Unit1.pas, сформованого спільно нами і середовищем Delphi.

Текст модуля Unit1.pas.

```
unit Unit1;

interface

uses
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

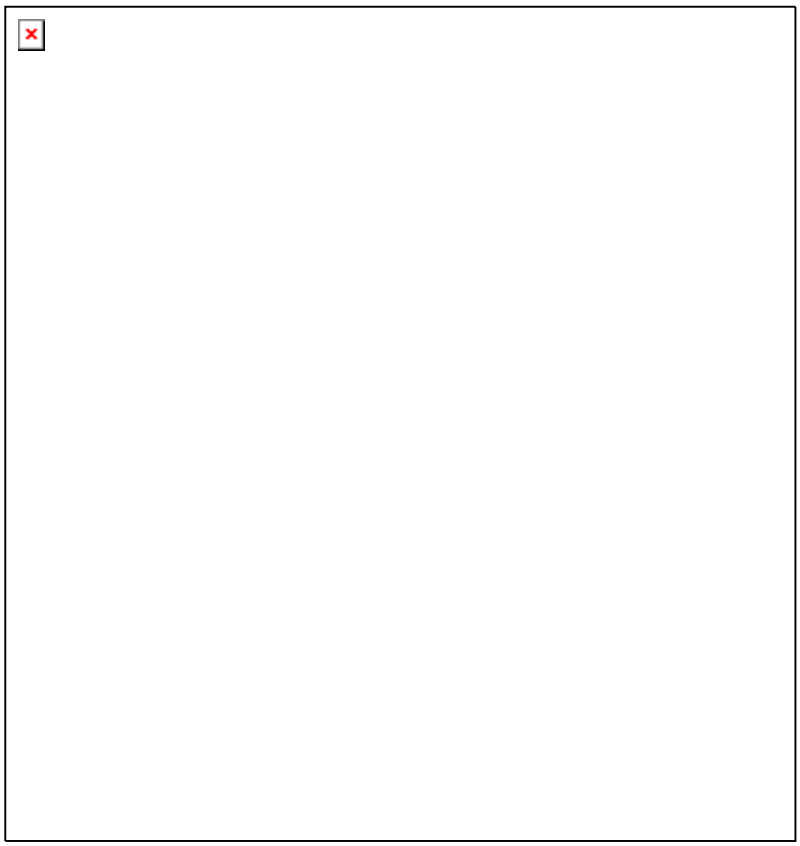
```
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Button1: TButton;
    Edit1: TEdit;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var r,s:real;
begin
  r := StrToFloat(Edit1.Text);
  s := pi*sqr(r);
  label2.Caption := 'Площа кругу дорівнює '+
    FloatToStrF(s,ffGeneral,7,2)
end;

end.
```

Рядки модуля, уставлені програмістом, виділені **напівжирним** шрифтом, уставлені середовищем Delphi на етапі конструювання форми – *курсивним* шрифтом і, нарешті, уставлені середовищем Delphi при створенні оброблювача події OnClick – підкресленим шрифтом. Як видно з цього приклада, значна кількість програмного коду модуля сформована Delphi автоматично.

Надалі при розгляді прикладів програмного коду рядки, уставлені програмістом, будемо виділяти напівжирним шрифтом.

Якщо заглянути в папку, у якій ми зберегли файл проекту і модуль, то знайдемо в ній целую групу файлів (див. мал. 2.10), частина з яких нам уже відома по створенню консольних застосувань, а про інші мова йшла в пункті 2.1.2 . Серед файлів, що входять у проект, новими для нас будуть:

Unit1.pas – файл модуля, використовуваний для збереження програмного коду;
Unit1.dfm – файл, що містить форму;
Unit1.dcu – відкомпільований файл модуля;
Unit1.~pas – резервна копія модуля;
Unit1.~dfm – резервна копія форми;
p2_1.res – файл, що містить використовувану проектом піктограму та інші ресурси.



Мал. 2.10. Файли, що входять у проект.

2.2. Класи й об'єкти

2.2.1. Основні принципи об'єктно-орієнтованого програмування

Object Pascal є об'єктно-орієнтованою мовою програмування. Зібрані в мові об'єктно-орієнтовані можливості інакше ще називають **об'єктною моделлю** мови програмування. Практичним результатом використання об'єктної моделі в Object Pascal є створення і підтримка компонентів. У цій главі ми розглянемо теоретичні основи об'єктної моделі Object Pascal, в основі якої лежать поняття класу й об'єкта.

Класами в Object Pascal називаються спеціальні типи, що містять поля, методи і властивості.

Об'єкт – це конкретний екземпляр класу, і, подібно іншим змінним, він описується в розділі var програми.

В основі класів лежать три фундаментальних принципи – інкапсуляція, спадкування і поліморфізм.

Інкапсуляцією називається об'єднання в класі даних і підпрограм для їхньої обробки. Дані містяться в полях класу, а процедури і функції для їхньої обробки називаються методами. Відповідно до правил об'єктно-орієнтованого програмування прямий доступ до полів класу небажаний. У зв'язку з цим у Object Pascal передбачені спеціальні конструкції, називані властивостями, що здійснюють читання чи записування у поля за допомогою виклику відповідних методів.

Інкапсуляція дозволяє створювати клас як щось цілісне, що має визначену функціональність. Як приклади можна привести стандартні класи, визначені в Delphi:

TEdit – дозволяє створювати і керувати роботою рядка введення;

TLabel – забезпечує функціонування мітки;

TButton – містить усе необхідне для роботи кнопки.

Загальноприйнятим правилом є давати назви класам, що починаються з букви T.

Спадкування означає, що будь-який клас може бути породжений іншим класом. Створити новий клас від деякого класу-батька можна за допомогою наступного програмного коду:

```
TNewClass = class(TOldClass)
```

Чудовою особливістю принципу спадкування є те, що породжений клас автоматично успадковує поля, методи і властивості свого батька і, більш того, може доповнювати їх новими. У результаті програміст, використовуючи стандартні класи, що маються в Delphi, може створювати власні класи і навіть цілі бібліотеки класів.

У Object Pascal усі класи є нащадками класу TObject. Цей клас не містить у собі полів і властивостей, зате його методи дозволяють створювати, підтримувати життєдіяльність і видаляти об'єкти. Якщо програміст хоче створити клас, що є безпосереднім нащадком класу TObject, то ім'я класу-батька в цьому випадку можна не вказувати, тобто наступні рядки є еквівалентними:

```
TSecondClass = class(TObject)
TSecondClass = class
```

Використання принципу спадкування в Delphi привело до створення розгалуженого дерева класів. У верхній частині цього дерева знаходяться так називані абстрактні класи, для яких не можна створити повноцінні працюючі об'єкти. Але разом з тим абстрактні класи є родоначальниками великих груп класів, для яких уже створюються реальні об'єкти.

Поліморфізм дозволяє використовувати однакові імена для методів, що входять у різні класи. Принцип поліморфізму забезпечує у випадку звертання до однойменних методів виконання того з них, що відповідає класу об'єкта.

Нехай, наприклад, ми вирішили створити новий клас, що відрізняється від батьківського класу тим, що в якомусь з його методів змінений алгоритм. У цьому випадку нам необхідно перекрити в класі-нащадку відповідний метод, тобто оголосити в класі-нащадку однойменний метод і записати в ньому потрібний алгоритм. У результаті ми одержимо два класи, що мають однойменні методи, що виконуються по-різному. Отримані класи будуть поліморфними.

У загальному виді клас оголошується в розділі type у такий спосіб:

```
type
    < ім'я класу > = class(< ім'я класу-батька >)
        public
          < опис загальнодоступних елементів >
        published
          < опис елементів, доступних в Інспекторі Об'єктів >
        protected
          < опис елементів, доступних у класах-нащадках >
        private
          < опис елементів, доступних тільки в модулі >
    end;
```

Секції public, published, protected, private можуть містити опис полів, методів, властивостей і подій. У наступних пунктах розглянемо більш докладно елементи, що входять у клас.

2.2.2. Поля

Полями називаються інкапсульовані в класі дані. Поля класу подібні полям запису, але на відміну від них можуть бути будь-як типу, у тому числі класами, наприклад:

```
type
    TChildClass = class
        FOne : Integer;
        FTwo : String;
        FThree : TObject;
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
        end;
```

Виходячи з принципу інкапсуляції, звертання до полів повинне здійснюватися за допомогою методів і властивостей класу. Разом з тим, у Object Pascal допускається звертатися до полів і безпосередньо. Для того щоб звернутися до поля, необхідно записати складене ім'я, що складається з імені класу й імені поля, розділених крапкою, наприклад:

```
var
        MyObject : TChildClass;
begin
        MyObject.FOne := 16;
        MyObject.FTwo := 'Деяке строкове значення';
end;
```

Помітимо, що звичайно ім'я поля таке ж, як і ім'я відповідної властивості, але до імені поля як першу букву додають букву F.

2.2.3. Методи

Методами називаються інкапсульовані в класі процедури і функції. Наприклад:

```
type
        TChildClass = class
                FOne : Integer;
                FTwo : String;
                FThree : TObject;
                function FirstFunc(x:real):real;
                procedure SecondProc;
        end;
```

Для того щоб звернутися до методів, як і для полів, необхідно використовувати складені імена:

```
var
        MyObject : TChildClass;
        y : real;
begin
……….
        MyObject.SecondProc;
        y := MyObject.FirstFunc(3.14);
……….
end;
```

Методи, визначені в класі, можуть бути статичними, віртуальними, динамічними чи абстрактними. Тип методу визначається механізмом перекриття його в нащадках.

Для статичних методів перекриття здійснюється компілятором. Наприклад, нехай у нас є опис батьківського класу TBase і його нащадка TDescedant, що містять однойменний метод MyJoy:

```
type
        TBase = class
                procedure MyJoy;
        end;
        TDescedant = class(TBase)
                procedure MyJoy;
        end;
var
        FirstObject : TBase;
        SecondObject : TDescedant;
begin
……..
```

```
                FirstObject.MyJoy;
                SecondObject.MyJoy;
        ……
        end;
```

Відповідно до принципу поліморфізму в операторі

```
                FirstObject.MyJoy;
```

викликається метод, описаний у класі TBase, а в операторі

```
                SecondObject.MyJoy;
```

викликається метод, описаний у класі TDescedant.

За замовчуванням усі методи, описані в класі, є статичними.

Динамічні і віртуальні методи відрізняються від статичних тим, що заміщення батьківських методів методами нащадків відбувається на етапі виконання програми. Для оголошення віртуального методу в батьківському класі необхідно використовувати зарезервоване слово virtual, а для оголошення динамічного методу – зарезервоване слово dynamic. У класі-нащадку в заголовку методу, що заміщає, повинне бути зазначене зарезервоване слово override. Наприклад:

```
type
        TBase = class
                procedure MyJoy; virtual;
        end;
        TDescedant = class(TBase)
                procedure MyJoy; override;
        end;
var
        FirstObject : TBase;
        SecondObject : TDescedant;
begin
        ……
        FirstObject.MyJoy;
        SecondObject.MyJoy;
        ……
end;
```

Якби ми захотіли, щоб метод MyJoy у класі TBase був динамічним, слово virtual у заголовку процедури варто замінити словом dynamic. Розходження між віртуальними і динамічними методами невелике і зв'язано з особливостями реалізації їхніх викликів. Можна сказати, що віртуальні методи більш ефективні з погляду витрат часу, а динамічні методи дозволяють більш раціонально використовувати оперативну пам'ять.

Абстрактними називаються віртуальні чи динамічні методи, що визначені в класі, але не містять ніяких дій, ніколи не викликаються й обов'язково повинні бути перевизначені в класах-нащадках. Оголошується абстрактний метод за допомогою зарезервованого слова abstract, розташованого після слів virtual чи dynamic, наприклад:

```
                procedure MyMetod; virtual; abstract;
```

Основне призначення абстрактних методів – бути родоначальником ієрархії конкретних методів у класах-нащадках.

У будь-якому класі містяться два спеціальних методи – конструктор і деструктор. Ці методи містяться в класі-родоначальнику всіх інших класів – TObject і, отже, успадковуються нащадками. Як і інші методи, вони можуть бути змінені в класах-нащадках, тобто перекриті. У класі TObject і в більшості його нащадків конструктор і деструктор називаються Create і Destroy відповідно.

Конструктори призначені для створення й ініціалізації об'єкта. Справа в тім, що об'єкт у мові Object Pascal є динамічною структурою і змінна-об'єкт містить не самі дані, а посилання на них. Конструктор розподіляє об'єкт у динамічній пам'яті і привласнює полям

об'єкта початкові значення. При цьому поля порядкових типів як початкове значення одержують 0, строкового – порожній рядок, поля-вказівники – значення nil, поля-варіанти – Unassigned. Крім того, конструктор поміщає посилання на створений об'єкт у змінну Self, що автоматично оголошується в класі. Зі сказаного випливає, що звертання до полів, властивостей і методів об'єкта повинне здійснюватися тільки після виклику конструктора.

Деструктор звільняє динамічну пам'ять і руйнує об'єкт.

Для оголошення конструктора і деструктора використовуються зарезервовані слова constructor і destructor відповідно. Наприклад:

```
type
    TSample = class
        Text:string;
        constructor Create;
        destructor Destroy;
    end;
```

Для того щоб створити об'єкт, необхідно застосувати метод-конструктор до класу об'єкта:

```
var
    MyObject : TSample;
begin
……..
    MyObject := TSample.Create;
……..
end;
```

Якщо створюється клас-нащадок і при його створенні планується здійснити деякі додаткові дії, відсутні в класі-батькові, то в конструкторі класу-нащадка варто спочатку викликати конструктор свого батька, а вже потім здійснювати додаткові дії. Викликати будь-який перекритий метод батьківського класу можна за допомогою зарезервованого слова inherited (успадкований). Наприклад, якщо в класі TDescedant є свій власний конструктор

```
type
    TDescedant = class(TBase)
        FMark : Boolean;
        constructor Create(Mark:Boolean);
    end;
```

те його реалізація могла б бути такою:

```
constructor TDescedant.Create(Mark:Boolean);
begin
    inherited Create;
    FMark := Mark;
end;
```

де виклик батьківського конструктора здійснюється оператором

```
    inherited Create;
```

а оператор

```
    FMark := Mark;
```

здійснює додаткові дії.

Крім деструктора Destroy, у базовому класі TObject визначений метод Free, що перевіряє, чи був об'єкт насправді створений і тільки потім викликає метод Destroy. Якщо об'єкт не був створений конструктором, то звертання до деструктора, приведе до генерації виняткової ситуації. Отже, для знищення непотрібного об'єкта зручніше використовувати метод Free, наприклад:

```
    MyObject.Free;
```

У класі можуть бути визначені методи, що викликаються без створення й ініціації об'єкта. Ці методи називаються методами класу, і для їхнього оголошення використовується зарезервоване слово class. Наприклад:

```
type
        TChildClass = class(TObject)
                class function ChildClassInfo:string;
        end;
var
        y : string;
begin
……….
        y := ChildClassInfo;
……….
end;
```

Як правило, методи класу призначені для одержання довідкової інформації про клас – імені класу, предка класу, розміру класу і т.д.

2.2.4. Властивості

Властивості зовні нагадують поля класу, але насправді являють собою механізм, що регулює доступ до полів. Як правило, властивість зв'язана з деяким полем класу і вказує ті методи класу, що повинні бути використані при читанні з цього поля або при записуванні в нього. Якщо визначений метод доступу для читання, то він повинний бути функцією без параметрів, що повертає значення того ж самого типу, що і властивість. Ім'я функції, призначеної для читання, прийнято починати з префіксу Get, після якого йде ім'я властивості. Метод, використовуваний для записування, повинний бути процедурою, що має один параметр. Цей параметр повинний бути того ж типу, що і властивість. Ім'я процедури, призначеної для записування, прийнято починати з префіксу Set, після якого йде ім'я властивості.

Для оголошення властивості використовуються зарезервовані слова property, read і write. Слова read і write позначають початок розділів, що містять імена методів, призначених для читання і записування відповідно. Наприклад

```
type
        TStudent = class
                FAge : integer;
                function GetAge : integer;
                procedure SetAge(Value : integer);
                property Age : integer read GetAge write SetAge;
        end;
```

Тут Age – властивість, зв'язана з полем FAge, GetAge і SetAge – методи, призначені відповідно для читання і записування в поле FAge.

Для звертання до властивості в тексті програми, так само, як для полів і методів, необхідно використовувати складені імена, що складаються з імені об'єкта, крапки й імені властивості, наприклад:

```
var
        GoodStudent : TStudent;
        HisAge : integer;
begin
        GoodStudent := TStudent.Create;
        GoodStudent.Age := 19;
……….
        HisAge := GoodStudent.Age;
……….
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
        GoodStudent.Free;
    end;
```

Використання властивостей, на відміну від безпосереднього використання полів, дозволяє здійснювати різні додаткові дії. Наприклад, ми могли б безпосередньо помістити в поле FAge потрібне значення

```
        GoodStudent.FAge := 19;
```

Але, використовуючи властивість Age, ми можемо реалізувати в методах, призначених для читання і записування – GetAge і SetAge – різні додаткові дії: перевірку значень, що вводяться, на приналежність до заданого діапазону, видачу повідомлень на екран, зміну зовнішнього вигляду об'єкта на екрані і т.д.

У тих випадках коли додаткові дії при читанні чи записуванні в поле не потрібні, замість імені відповідного методу можна вказати ім'я поля:

```
    type
        TStudent = class
            FAge : integer;
            procedure SetAge(Value : integer);
            property Age : integer read FAge write SetAge;
        end;
```

Поля можуть бути доступні тільки для чи читання тільки для записування. У цьому випадку при описі властивості опускаються розділи read чи write. Наприклад, якщо ми хочемо, щоб властивість Age була доступною тільки для читання, нам необхідно забрати розділ write:

```
    type
        TStudent = class
            FAge : integer;
            function GetAge : integer;
            property Age : integer read GetAge ;
        end;
```

Помітимо, що властивість може бути і не зв'язана з конкретним полем. У цьому випадку у властивості просто визначаються два методи, що виконують деякі дії з даними того ж типу, що і властивість.

2.2.5. Події

У пункті 2.1.1 уже говорилося, що подія – це те, що відбувається в процесі роботи програми. У Delphi визначено кілька десятків типових подій. Розглянемо, що являють собою події з погляду мови Object Pascal.

Подія – це властивість процедурного типу, і її значенням є вказівник на деякий метод. Привласнити такій властивості значення означає вказати адресу методу, що буде виконуватися в момент настання події. Такі методи, як говорилося раніше, називаються оброблювачами подій. Як приклад розглянемо, як описані стандартні події OnDblClick (виникає при подвійному щиглику лівою кнопкою миші), OnMouseDown (виникає при натисканні кнопки миші) і OnMouseMove (виникає при переміщенні миші) у класі TControl:

```
TControl = class(TComponent)
  private
……….
    FOnDblClick: TNotifyEvent;
    FOnMouseDown: TMouseEvent;
    FOnMouseMove: TMouseMoveEvent;
……….
  protected
……….
    property OnDblClick: TNotifyEvent read FOnDblClick write FOnDblClick;
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
    property OnMouseDown: TMouseEvent read FOnMouseDown write FOnMouseDown;
    property OnMouseMove: TMouseMoveEvent read FOnMouseMove write FOnMouseMove;
……
end;
```

Призначення зарезервованих слів private і protected ми розглянемо пізніше, а поки відзначимо наступне.

Поля FOnDblClick, FOnMouseDown і FOnMouseMove, використовувані при описі властивостей-подій, призначені для збереження вказівника на метод, що є оброблювачем відповідної події. Використовувані при описі полів процедурні типи визначаються в такий спосіб:

```
type
    TNotifyEvent = procedure (Sender: TObject) of object;
    TMouseEvent = procedure(Sender: TObject; Button: TMouseButton;
      Shift: TShiftState; X, Y: Integer) of object;
    TMouseMoveEvent = procedure(Sender: TObject; Shift: TShiftState;
      X, Y: Integer) of object;
```

Загальним для всіх процедурних типів є параметр Sender, що містить вказівник на об'єкт – джерело події. Параметр Button містить інформацію, яка клавіша миші була натиснута. Параметр Shift містить код клавіші, натиснутої на клавіатурі. X і Y – поточні координати миші в межах компонента.

Метод відрізняється від звичайної процедури тим, що крім явно описаних параметрів методу завжди неявно передається ще і вказівник на об'єкт, який викликав цей метод. Вказівник поміщується в автоматично визначену в класі змінну Self. Для того щоб підкреслити, що описані вище процедурні типи призначені для роботи саме з методами, тобто є типами методів, використовуються зарезервовані слова of object.

На сторінці Events в Інспекторі Об'єктів відображаються тільки ті властивості компонента, що мають тип методу, тобто події. Тип оброблювача події, що ставиться у відповідність якій-небудь події, повинний мати той же тип, що і подія, тому тип методу інакше ще називається типом оброблювача події.

2.2.6. Області видимості елементів класу

Кожен клас може містити чотири секції, що визначаються зарезервованими словами published (декларовані), private (особисті), protected (захищені) і public (доступні). Секції визначають області видимості елементів, що входять у клас. Усередині кожної секції спочатку описуються поля, а далі – методи, властивості і події.

Секція public містить елементи, видимі в будь-якій програмі чи модулі, що мають доступ до даного модуля, який містить оголошуваний клас.

Найбільш строгою є секція private, що обмежує видимість тим модулем, у якому розташований клас. Секція private повинна містити тільки ті поля і методи, що дійсно залежать від класу і повинні бути сховані від усіх похідних класів.

Елементи, що знаходяться в секції protected, усередині модуля підкоряються тим же правилам, що і для секцій public і private. Але за межами модуля доступ до захищених елементів мають только методи класів-нащадків.

У секції published правила видимості ідентичні правилам секції public. Відмінність полягає в тім, що секція published спеціально розроблена для властивостей, які будуть доступні на етапі конструювання програми у вікні Інспектора Об'єктів. Поля, оголошені в секції published, повинні мати класовий тип. Розташована на початку оголошення класу форми секція, що не має назви, за замовчуванням є секцією published.

Будь-яка секція при описуванні класу може створюватися довільну кількість разів. Порядок розташування секцій також може бути довільним. Крім того, будь-яка секція може бути порожньою.

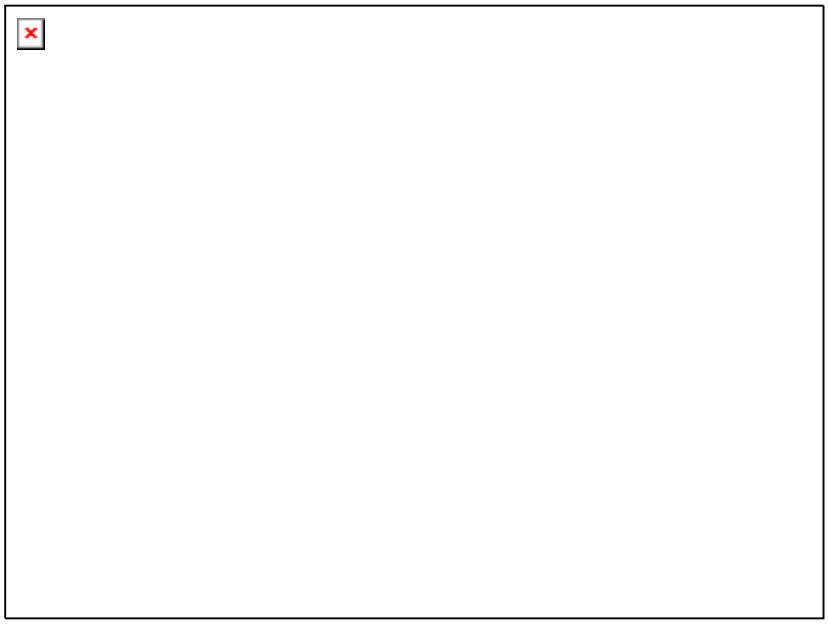
Повертаючись до приклада 2.1 з пункту 2.1.3 і використовуючи термінологію об'єктно-орієнтованого програмування, можна сказати наступне.

У процесі конструювання форми був створений клас TForm1, що є нащадком стандартного класу TForm, визначеного в Delphi. У класі TForm1 визначені поля Label1, Label2, Button1, Edit1. Усі ці поля мають класовий тип. Крім того, у класі TForm1 описаний метод Button1Click, що є оброблювачем події OnClick для кнопки Button1. Описи полів і методу класу TForm1 розташовані в секції published і доступні у вікні Інспектора Об'єктів.

2.3. Компоненти

2.3.1. Бібліотека візуальних компонентів

Класи, створені розроблювачами Delphi, утворюють складну ієрархічну структуру, називану Бібліотекою візуальних компонентів (Visual Component Library – VCL). Кількість вхідних у VCL класів складає кілька сотень. На малюнку 2.11 зображені базові класи, що є родоначальниками всіх інших класів.



Мал. 2.11. Базові класи ієрархії класів Delphi

Компонентами називаються екземпляри класів, що є нащадками класу TComponent. Екземпляри всіх інших класів називаються **об'єктами**. Різниця між компонентами і просто об'єктами полягає в тім, що компонентами можна маніпулювати на формі, а об'єктами – не можна.

Характерним прикладом класу, визначеного в VCL, але не є компонентом, є клас TFont. Ми не можемо безпосередньо помістити на форму об'єкт класу TFont. З іншого боку, при роботі, наприклад, з такими компонентами, як Label чи Edit ми будемо використовувати властивість Font класового типу TFont.

Помітимо також, що не всі компоненти-нащадки класу TComponent є візуальними. Наприклад, компонент Timer, призначений для відліку інтервалів реального часу, є невізуальним.

Сказане вище трохи суперечить назві VCL – Бібліотека візуальних компонентів, але, з іншого боку, візуальні компоненти є головним досягненням розроблювачів Delphi, тими будівельними елементами, за допомогою яких створюється каркас будь-якого застосування. Інші класи VCL є базою для створення візуальних компонентів або носять допоміжний характер.

Як уже говорилося раніше, відповідно до принципу спадкування компоненти Delphi успадковують дані і методи для їхньої обробки від своїх батьків. Тому, перш ніж перейти до знайомства з конкретними компонентами, буде корисно познайомитися з базовими класами, приведеними на малюнку 2.11.

PDF-XChange
Click to buy NOW!
www.docu-track.com

2.3.2. Клас TObject

Клас TObject є предком усіх класів, що входять у VCL і забезпечує можливість створення, керування і руйнування об'єктів. Для цього в класі визначені наступні методи:

constructor Create;
Конструктор Create виконує роботу з виділення під об'єкт необхідної динамічної пам'яті. Ініціалізацію даних не здійснює, оскільки перевантажується в класах-нащадках.

destructor Destroy; virtual;
Деструктор Destroy звільняє виділену під об'єкт, що видаляється, динамічну пам'ять.

procedure Free;
Знищує об'єкт і звільняє виділену під нього динамічну пам'ять, якщо об'єкт був створений, тобто вказівник на нього не дорівнює nil. При видаленні викликає деструктор Destroy. Виконання перевірки існування об'єкта перед його видаленням робить метод Free переважніше деструктора Destroy.

У класі TObject містяться ряд методів класу, що дозволяють одержати загальні характеристики класу:

class function ClassName: ShortString;
Функція класу ClassName повертає рядок, що містить ім'я класу для даного об'єкта, наприклад: 'TEdit', 'TButton', 'TLabel' і т.д.

class function ClassNameIs(const Name: string): Boolean;
Повертає значення true, якщо значення, що міститься в параметрі Name, збігається з ім'ям даного класу.

class function ClassParent: TClass;
Повертає клас безпосереднього предка даного класу.
Тип TClass називається вказівником на клас чи метаклассом. Визначається він у такий спосіб:

type TClass = class of TObject;

Якщо в програмі оголошена змінна типу TClass,

var AnyObj: TClass;

то значенням змінної AnyObj може бути посилання на будь-який клас, що є нащадком класу TObject.

class function InheritsFrom(AClass: TClass): Boolean;
Перевіряє, чи є клас, ім'я якого міститься в параметрі AClass, предком даного класу чи об'єкта.

class function InstanceSize: Longint;
Повертає розмір класу чи об'єкта в байтах.

class function MethodAddress(const Name: ShortString): Pointer;
Повертає адрес опублікованого методу, ім'я якого міститься в параметрі Name.

class function MethodName(Address: Pointer): ShortString;
Повертає ім'я опублікованого методу, адреса якого міститься в параметрі Address.

Більшість методів TObject не використовується безпосередньо в компонентах, задіяних при конструюванні форми застосування. Звичайно методи TObject перевантажені в спадкоємцях чи замінені іншими, побудованими на їхній основі.

Відзначимо також, що хоча формально TObject не є абстрактним класом, тобто не містить абстрактних методів, але об'єкти цього класу створювати не можна.

2.3.3. Клас TPersistent

Клас TPersistent походить безпосередньо від класу TObject і містить методи, необхідні для створення потокових об'єктів. **Потоковий об'єкт** – це об'єкт, що може запам'ятовуватися в потоці. У свою чергу, **потік** являє собою об'єкт, що інкапсулює деякий носій інформації, наприклад пам'ять чи дискові файли. Іншими словами, нащадки класу TPersistent можуть бути, зокрема, поміщені в оперативну пам'ять або у файл форми і витягнуті відтіля. Для реалізації сказаного до методів, наслідуваних від класу TObject, додані наступні:

procedure Assign(Source: TPersistent);
Привласнює даному об'єкту дані, що містяться в об'єкті, ім'я якого зазначено в параметрі Source.

procedure AssignTo(Dest: TPersistent); virtual;
Метод аналогічний Assign, але на відміну від його є захищеним і віртуальним.

procedure DefineProperties(Filer: TFiler); virtual;
Дозволяє помістити неопубліковані дані об'єкта в потік (файл форми). Клас TFiler є абстрактним базовим класом, призначеним для здійснення операцій читання і записування при завантаженні і збереженні компонентів, а також їхніх властивостей.

function GetNamePath: string; dynamic;
Повертає рядок, що містить ім'я об'єкта і використовується в Інспекторі Об'єктів.

function GetOwner: TPersistent; dynamic;
Захищений метод, що повертає вказівник на власника об'єкта.

Клас TPersistent не призначений для створення об'єктів і використовується тільки для створення похідних класів.

Істотною особливістю класу TPersistent і, отже, усіх його нащадків, є те, що цей клас скомпільований з директивою {$M+}. Це означає, що клас TPersistent має опубліковану секцію. Поміщені в цю секцію властивості в класах-нащадках будуть відображатися в Інспекторі Об'єктів.

2.3.4. Клас TComponent

Клас TComponent являє собою вершину в ієрархії компонентів. Від нього породжені усі компоненти, використовувані в застосуванні, включаючи саме застосування (клас TApplication) і форму (клас TForm). Прямими нащадками класу TComponent є невізуальні компоненти. До невізуальних відносяться компоненти, зовнішній вигляд яких на стадії проектування відрізняється від зовнішнього вигляду на стадії виконання програми. Зокрема, під час виконання програми вони можуть бути взагалі не видні. Прикладами таких компонентів є різні види меню, стандартні діалогові вікна, таймер і т.д. Клас TComponent є також базою для створення візуальних компонентів, про які буде сказано пізніше.

У класі TComponent визначені властивості і методи, загальні для всіх компонентів. Розглянемо основні з них.

type TComponentName: string;

PDF-XChange
Click to buy NOW!
www.docu-track.com

property Name: TComponentName;
Ім'я компонента. При приміщенні компонента на форму Delphi привласнює йому стандартне ім'я, наприклад Label1 чи Edit2, яке можна змінити на більш осмислене за своїм розсудом. Ім'я компонента повинне бути правильним ідентифікатором.

property Tag: Longint;
Властивість, призначена для розроблювачів. У цій властивості розроблювач може зберігати деяке число типу Longint і розпоряджатися їм за своїм розсудом.

У класі TComponent уводиться концепція **приналежності**, що поширюється на всю VCL. Зміст її полягає в тім, що будь-який компонент Delphi є власністю іншого компонента і, у свою чергу, може бути власником одного чи декількох компонентів. Найбільш важливим результатом цієї концепції є те, що при руйнуванні компонента-власника автоматично будуть зруйновані приналежні йому компоненти. Наприклад, при руйнуванні форми усі компоненти, що належать їй, також будуть зруйновані, і відповідна пам'ять буде звільнена.

Перелічимо властивості і методи, що реалізують концепцію приналежності.

property ComponentCount: Integer;
Задає кількість компонентів, власником яких є даний компонент. Властивість доступна тільки під час виконання програми і тільки для читання.

property ComponentIndex: Integer;
Визначає положення компонента в списку компонентів власника. Нумерація компонентів починається з нуля. Властивість доступна тільки під час виконання програми і тільки для читання.

property Components[Index: Integer]: TComponent;
Властивість-масив містить список компонентів, для яких даний компонент є власником. Властивість доступна тільки під час виконання програми і тільки для читання.

property Owner: TComponent;
Містить вказівник на компонент-власник для поточного компонента. Задається автоматично при приміщенні компонента в форму чи форми в застосування. Властивість доступна тільки під час виконання програми і тільки для читання.

procedure DestroyComponents;
Видаляє з динамічної пам'яті компоненти даного компонента-власника.

function FindComponent(const AName: string): TComponent;
Повертає вказівник на компонент, ім'я якого міститься в параметрі AName. Якщо компонент із таким ім'ям не виявлений, повертається значення nil.

procedure InsertComponent(AComponent: TComponent);
Уставляє компонент, ім'я якого зазначено в параметрі AComponent, у кінець списку компонентів, що знаходиться в масиві Components.

procedure RemoveComponent(AComponent: TComponent);
Видаляє компонент AComponent зі списку компонентів, що знаходиться в масиві Components.

У класі TComponent перевизначений конструктор:

PDF-XChange Click to buy NOW! www.docu-track.com

constructor Create(AOwner: TComponent); virtual;
Створює об'єкт даного класу, поміщає посилання на себе в масив Components свого власника, ім'я якого зазначено в параметрі AOwner. Значення параметра AOwner привласнюється властивості Owner створюваного компонента.

Як і попередні класи, клас TPersistent не призначений для створення об'єктів і використовується тільки для створення похідних класів.

2.3.5. Клас TControl

Клас TControl забезпечує велику частину властивостей, методів і подій візуальних компонентів, за допомогою яких виводиться інформація на екран і за допомогою яких можна вводити інформацію в програму, використовуючи клавіатуру і мишу. Для нащадків класу TControl використовується загальна назва – елементи керування.

Розглянемо основні характеристики цього класу.

У класі TControl уводиться поняття **батьківського елемента керування** (parent controls). Суть цього поняття полягає в наступному.

Кожен елемент керування може бути або безпосередньо поміщений у форму, або поміщений у деякий додатковий компонент, що групує, наприклад у панель (клас TPanel) чи в прокручувану область з лінійками скролінга (клас TScrollBox). У першому випадку батьківським елементом керування буде форма, а в другому – елемент, що групує. У якості батьківського може виступати тільки віконний елемент керування, тобто нащадок класу TWinControl, що буде розглянутий у пункті 2.3.4. Зв'язані з батьківським дочірні елементи керування можуть бути як віконними, так і невіконними. Використання зв'язку батьківський-дочірній дозволяє використовувати дочірньому елементу керування тих чи інших характеристик батьківського елемента. Це дозволяє створити однакове зображення батьківських і дочірніх елементів, що додає зображенню на екрані гарний стиль. Дочірні елементи не можуть виходити за границі свого батька. Якщо батьківський елемент переміщається по екрані, то разом з ним переміщаються і всі дочірні елементи. Таким чином, батьківський і дочірній елементи розглядаються як єдине ціле.

Помітимо, що не слід плутати властивість Owner, визначену в класі TComponent, з поняттям батьківського елемента керування. Властивість Owner указує на компонент, у який поміщений даний компонент (компоненти можуть бути невидимими), а зв'язок батьківський-дочірній дозволяє керувати видимими компонентами.

Для забезпечення відносини батьківський-дочірній призначені наступні властивості:

property Parent: TWinControl;
Задає компонент, що є батьком стосовно поточного елементу керування.

property ParentBiDiMode: Boolean;
Указує, чи використовує компонент значення властивості BiDiMode (див. далі) свого батька. У противному випадку використовується власна властивість BiDiMode.

property ParentColor: Boolean;
Указує, чи використовує компонент значення властивості Color (див. далі) свого батька. Інакше використовується власна властивість Color.

property ParentFont: Boolean;
Указує, чи використовує компонент значення властивості Font (див. далі) свого батька. Інакше використовується власна властивість Font.

property ParentShowHint: Boolean;

PDF-XChange Click to buy NOW! www.docu-track.com

Указує, чи використовує компонент значення властивості ShowHint (див. далі) свого батька. У противному випадку використовується власна властивість ShowHint.

Оскільки батьківський елемент керування повинний бути нащадком класу TWinControl, то саме в цьому класі визначені інші характеристики відносини батьківський-дочірній.

Для відображення елементів керування на екрані монітора використовуються властивості позиціювання і вирівнювання:

property Left: Integer;
Задає горизонтальну координату лівого верхнього кута елемента керування в пікселях щодо компонента-батька.

property Top: Integer;
Задає вертикальну координату лівого верхнього кута елемента керування в пікселях щодо компонента-батька.

property Width: Integer;
Задає ширину елемента керування в пікселях.

property Height: Integer;
Задає висоту елемента керування в пікселях.

type TAlign = (alNone, alTop, alBottom, alLeft, alRight, alClient);
property Align: TAlign;
Властивість Align визначає спосіб вирівнювання елемента керування в батьківському компоненті (формі чи груповому компоненті). Воно може приймати наступні значення:

alNone – вирівнювання не відбувається (використовується за замовчуванням);

alTop – вирівнюється по верхньому краї батьківського компонента, розширюється по всій його ширині, висота не змінюється;

alBottom – вирівнюється по нижньому краї батьківського компонента, розширюється по всій його ширині, висота не змінюється;

alLeft – вирівнюється по лівому краї батьківського компонента, збільшується по всій його висоті, ширина не змінюється;

alRight – вирівнюється по правому краї батьківського компонента, збільшується по всій його висоті, ширина не змінюється;

alClient – елемент керування розширюється до всієї клієнтської області батьківського компонента.

type TAnchors = set of TAnchorKind;
type TAnchorKind = (akTop, akLeft, akRight, akBottom);
property Anchors: TAnchors;
Властивість визначає прив'язку даного елемента керування до країв елемента-батька при зміні розмірів останнього. Властивість є множиною, що може містити наступні значення:

akTop – елемент прив'язаний до верхнього краю батьківського;

akLeft – елемент прив'язаний до лівого краю батьківського;

akRight – елемент прив'язаний до правого краю батьківського;

akBottom – прив'язаний до нижнього краю батьківського.

Якщо властивість Anchors містить прив'язки до протилежних сторін батьківського компонента, наприклад [akLeft,akRight], то при зміні розмірів батьківського компонента відбувається розтягання чи стиск даного компонента, оскільки відстані від сторін

батьківського компонента витримуються. При стиску може відбуватися повне знищення зображення даного елемента керування.

property AutoSize: Boolean;
Якщо властивість приймає значення True,то буде відбуватися автоматичне підстроювання розміру компонента під розмір його вмісту, наприклад, висота компонента Edit буде підбудовуватися під висоту шрифту тексту, що міститься в ньому.

property ClientHeight: Integer;
Задає висоту клієнтської області елемента керування в пікселях. Значення цієї властивості для більшості елементів керування збігається зі значенням властивості Height. Однак для деяких компонентів, наприклад форми, у клієнтську область не входить її обрамлення – заголовок, головне меню, рамка.

property ClientWidth: Integer;
Задає ширину клієнтської області елемента керування в пікселях.

У класі TControl визначені методи, що дозволяють перетворювати локальні координати елемента керування, що визначають його положення в елементі-батькові, у глобальні координати, що визначають його положення на екрані дисплея:

```
type TPoint = record
  X: Longint;
  Y: Longint;
end;
```
function ClientToScreen(const Point: TPoint): TPoint;
Повертає глобальні координати крапки, заданої локальними координатами, що містяться в параметрі Point. X – горизонтальна координата в пікселях, Y – вертикальна координата в пікселях.

function ScreenToClient(const Point: TPoint): TPoint;
Повертає локальні координати крапки, заданої глобальними координатами, що містяться в параметрі Point.

У класі TControl є цілий ряд властивостей, що визначають зовнішній вигляд елемента керування:

type TBiDiMode = (bdLeftToRight, bdRightToLeft, bdRightToLeftNoAlign, bdRightToLeftReadingOnly);
property BiDiMode: TBiDiMode;
Визначає спосіб поводження об'єктів з урахуванням національної специфіки: за замовчуванням дорівнює bdLeftToRight, що забезпечує введення тексту і прокручування інформації зліва направо і характерно для європейських мов. Інші значення використовуються для країн Сходу.

type TCaption = string;
property Caption: TCaption;
Містить заголовок елемента керування.

type TColor = $00000000..$02FFFFFF;
property Color: TColor;

Властивість Color визначає колір тла елемента керування. Значення кольору може задаватися як чотирьохбайтне шістнадцятирічне число, що визначає використовувану палітру кольорів, а також інтенсивність червоного, зеленого і синього кольорів у форматі RGB. Для найбільше часто використовуваних кольорів визначені відповідні константи:

Константи, використовувані як значення властивості Color.

Таблиця 2.1

Константа	Колір
clBlack	Чорний
clMaroon	Темно-бордовий
clGreen	Зелений
clOlive	Маслиново-зелений
clNavy	Темно-синій
clPurple	Пурпурний
clTeal	Морської води
clGray	Сірий
clSilver	Срібний
clRed	Червоний
clLime	Лимонно-зелений
clBlue	Синій
clYellow	Жовтий
clFuchsia	Бузковий
clAqua	Голубой
clWhite	Білий
clBackground	Поточний колір тла столу Windows
clScrollBar	Поточний колір смуг прокручування
clActiveCaption	Поточний колір тла смуги заголовка в активному вікні
clInactiveCaption	Поточний колір тла смуги заголовка в неактивному вікні
clMenu	Поточний колір тла меню
clWindow	Поточний колір тла вікон
clWindowFrame	Поточний колір рамок вікон
clMenuText	Поточний колір тексту меню
clWindowText	Поточний колір тексту вікон
clCaptionText	Поточний колір тексту заголовка в активному вікні
clActiveBorder	Поточний колір бордюру активного вікна
clInactiveBorder	Поточний колір бордюру неактивного вікна
clAppWorkSpace	Поточний колір робочої області застосувань
clHighlight	Поточний колір тла виділеного тексту
clHightlightText	Поточний колір виділеного тексту
clBtnFace	Поточний колір поверхні кнопок
clBtnShadow	Поточний колір тіней, що відкидаються кнопками
clGrayText	Поточний колір тексту недоступних елементів
clBtnText	Поточний колір тексту кнопок
clInactiveCaptionText	Поточний колір тексту заголовка в неактивному вікні
clBtnHighlight	Поточний колір виділеної кнопки
cl3DDkShadow	Колір темних тіней тривимірних елементів; тільки для Windows 95 чи NT 4.0
cl3DLight	Світлий колір на краях освітлених тривимірних елементів; тільки для Windows 95 чи NT 4.0
clInfoText	Колір тексту порад; тільки для Windows 95 чи NT 4.0
clInfoBk	Колір тла порад; тільки для Windows 95 чи NT 4.0

property Enabled: Boolean;

PDF-XChange
Click to buy NOW!
www.docu-track.com

Визначає, чи реагує компонент на події, зв'язані з мишею, клавіатурою і таймером.

property Font: TFont;
Властивість класового типу TFont, що визначає характеристики шрифту, використовуваного для відображення текстової інформації. Клас TFont буде розглянутий у главі 2.5.

property Hint: string;
Містить текст, відображуваний у вікні чи підказки в рядку стану.

property ShowHint: Boolean;
Дозволяє чи забороняє показувати вікно підказки.

property Text: TCaption;
Містить текст, асоційований з даним елементом керування, наприклад – текст, що міститься в рядку введення Edit.

property Visible: Boolean;
Визначає, чи відображається елемент керування на екрані.

Існує ряд методів, що дозволяють змінювати зовнішній вигляд елемента керування на стадії виконання програми:

procedure BringToFront;
Поміщає елемент керування поверх інших елементів в елементі-батькові.

procedure Hide;
Робить поточний елемент керування і всі його дочірні елементи невидимими. Властивість Visible приймає значення False.

procedure Refresh;
Стирає, а потім малює заново елемент керування.

procedure SendToBack;
Поміщає елемент керування нижче всіх інших елементів в елементі-батькові.

procedure Show;
Робить поточний елемент керування видимим. Видимими стають також і всі його дочірні елементи, якщо властивість Visible у них має значення True.

У класі TControl визначені властивості, що дозволяють змінювати форму вказівника миші при переміщенні по екрані, а також забезпечують перетаскування (Drag & Drop) елементів керування:

type TCursor = -32768..32767;
property Cursor: TCursor;
Визначає вид курсору миші при перебуванні його в області компонента.

property DragCursor: TCursor;
Визначає вид курсору миші під час перетаскування елемента керування..

type TDragMode = (dmManual, dmAutomatic);
property DragMode: TDragMode;

PDF-XChange Click to buy NOW! www.docu-track.com

Задає режим перетаскування елемента керування: dmAutomatic – елемент керування можна перетаскувати безпосередньо мишею, dmManual – безпосереднє перетаскування мишею неможливо.

Розглянемо основні події, визначені в класі TControl.
Події, що виникають при натисканні на ліву кнопку миші:

type TNotifyEvent = procedure (Sender: TObject) of object;
property OnClick: TNotifyEvent;
Виникає при виборі елемента керування мишею, тобто була натиснута і відпущена ліва клавіша миші, коли курсор миші знаходився над елементом керування.

property OnDblClick: TNotifyEvent;
Виникає при подвійному щиглику лівою клавішею миші по елементі керування.

Загальні події, що виникають при маніпулюванні мишею:

type TMouseEvent = procedure (Sender: TObject; Button: TMouseButton; Shift: TShiftState; X, Y: Integer) of object;
property OnMouseDown: TMouseEvent;
Подія виникає при натисканні клавіші миші, коли курсор знаходиться над елементом керування. Значення параметра Button визначають, яка кнопка миші натиснута: mbLeft – ліва, mbRight – права, mbMiddle – середня. Параметр Shift являє собою множину, що містить позначення натиснутої кнопки миші або натиснутих одночасно з цим допоміжних клавіш Shift, Alt, Ctrl (ssShift – натиснута клавіша Shift, ssAlt – натиснута клавіша Alt, ssCtrl – натиснута клавіша Ctrl, ssLeft – ліва кнопка миші, ssRight – права кнопка миші, ssMiddle – середня кнопка миші, ssDouble – подвійний щиглик кнопкою миші). Параметри X і Y визначають координати вказівника миші в клієнтській області компонента. Параметр Sender – вказівник на компонент, у якому відбулася подія.

type TMouseMoveEvent = procedure(Sender: TObject; Shift: TShiftState; X, Y: Integer) of object;
property OnMouseMove: TMouseMoveEvent;
Подія виникає при переміщенні миші, коли її курсор знаходиться над елементом керування і натиснута клавіша миші. Параметри мають той же зміст, що і для події OnMouseDown.

property OnMouseUp: TMouseEvent;
Подія виникає при відпусканні клавіші миші, коли курсор знаходиться над елементом керування.

Події, призначені для підтримки перетаскування (Drag & Drop):

type TDragDropEvent = procedure(Sender, Source: TObject; X, Y: Integer) of object;
property OnDragDrop: TDragDropEvent;
Подія виникає, коли елемент керування Source опускається на елемент Sender. X,Y- глобальні координати курсору миші під час опускання елемента керування Source.

type TDragOverEvent = procedure(Sender, Source: TObject; X, Y: Integer; State: TDragState; var Accept: Boolean) of object;
property OnDragOver: TDragOverEvent;
Подія виникає коли, елемент керування Source, що перетаскується, знаходиться над елементом Sender. X,Y- глобальні координати курсору миші. Параметр State типу TDragState визначає стан об'єкта, що перетаскується, стосовно інших об'єктів. Можливі наступні стани:

dsDragEnter – курсор миші входить у межі компонента; DsDragMove – курсор миші переміщається в межах компонента; DsDragLeave – курсор миші виходить за межі компонента. Параметр Accept повинний прийняти значення True в оброблювачі події, якщо елемент керування Sender може прийняти елемент, що перетаскується, і False – у противному випадку.

type TEndDragEvent = procedure(Sender, Target: TObject; X, Y: Integer) of object;
property OnEndDrag: TEndDragEvent;
Подія виникає, коли завершується перетаскування елемента керування Sender над елементом Target. X,Y- глобальні координати курсору миші під час припинення перетаскування..

type TStartDragEvent = procedure (Sender: TObject; var DragObject: TDragObject) of object;
property OnStartDrag: TStartDragEvent;
Подія виникає, коли починається перетаскування елемента керування Sender. Параметр DragObject містить посилання на об'єкт, використовуваний для формування зображення елемента керування під час перетаскування. За замовчуванням дорівнює nil, тобто це означає, що переноситься буде сам компонент.

2.3.6. Клас TWinControl

Клас TWinControl є нащадком класу TControl і використовується як базовий для створення віконних елементів керування. З кожним віконним елементом керування зв'язане вікно Windows, обумовлене спеціальним числовим ідентифікатором – дескриптором вікна. Наявність вікна дозволяє активізувати елемент керування під час виконання програми. Зокрема, це означає, що такі елементи під час виконання програми можуть одержувати фокус введення і реагувати на події, що виникають при використанні клавіатури.

У класі TWinControl завершується визначення характеристик зв'язку батьківський-дочірній, почате в класі TControl. Нагадаємо, що тільки віконні елементи керування можуть містити інші компоненти, тобто бути батьками, чи інакше – компонентами-контейнерами інших, дочірніх компонентів.

Характерними представниками сімейства TWinControl є рядок уведення Edit, багаторядковий редактор Memo, список List Box, кнопка Button і т.д.

Розглянемо основні характеристики цього класу.

Приведемо властивості, призначені для підтримки зв'язку батьківський-дочірній:

property ControlCount: Integer;
Задає кількість дочірніх компонентів елемента керування. Властивість може бути використана тільки для читання і на етапі виконання програми.

property Controls[Index: Integer]: TControl;
Властивість Controls є властивістю-масивом і містить усі дочірні компоненти даного елемента керування. Нумерація компонентів починається з нуля. Властивість може бути використана тільки для читання і на етапі виконання програми.

property ParentCtl3D: Boolean;
Якщо значення властивості дорівнює True, то використовується значення властивості Ctl3D компонента-батька. У противному випадку використовується власна властивість. Властивість Ctl3D розглянута нижче.

Крім властивостей, для реалізації зв'язку батьківський-дочірній використовуються також наступні методи:

function ContainsControl(Control: TControl): Boolean;

PDF-XChange
Click to buy NOW!
www.docu-track.com

Повертає True, якщо поточний елемент керування є батьківським стосовно компонента Control.

function ControlAtPos(const Pos: TPoint; AllowDisabled: Boolean, AllowWinControls:Boolean = False): TControl;
Дозволяє визначити, який дочірній компонент даного віконного елемента розташований у позиції з координатами, зазначеними параметром Pos. Вказівник на дочірній компонент повинний знаходиться у властивості Controls батька. Позиція Pos може знаходитися в будь-якім місці усередині дочірнього компонента. Якщо задана позиція не відповідає ніякому дочірньому компоненту, то функція ControlAtPos повертає nil. Якщо параметр AllowDisabled дорівнює True, то враховуються дочірні компоненти, у яких властивість Enabled дорівнює False. Якщо параметр AllowWinControls дорівнює True, то враховуються дочірні компоненти, що є нащадками класу TWinControl. За замовчуванням ця властивість має значення False.

procedure InsertControl(AControl: TControl);
Поміщає компонент AControl у властивість-масив Controls даного віконного елемента.

procedure RemoveControl(AControl: TControl);
Видаляє компонент AControl із властивості-масиву Controls даного віконного елемента.

Розглянемо властивості віконних елементів керування, що дозволяють формувати їхнє зображення в стилі Windows:

property Brush: TBrush;
Властивість класового типу Brush задає кисть, за допомогою якої малюється тло віконного елемента. Властивість може бути використано тільки для читання і на етапі виконання програми. Клас TBrush буде розглянутий у главі 2.5.

property Ctl3D: Boolean;
Визначає зовнішній вигляд елемента. Якщо Ctl3D дорівнює True, елемент виглядає об'ємним. Якщо ж Ctl3D дорівнює False, то елемент виглядає плоским. За замовчуванням Ctl3D дорівнює True.

property Handle: HWND;
Визначає числовий ідентифікатор (дескриптор) вікна керування, по якому Windows може звертатися до цього вікна. Властивість може бути використано тільки для читання і на етапі виконання програми.

property Showing: Boolean;
Якщо значення властивості – True, то компонент видний на екрані, у противному випадку – ні. Значення True властивості Showing еквівалентно тому, що властивість Visible компонента і всіх його батьків дорівнює True. Якщо ж властивість Visible компонента чи якогось з його батьків дорівнює false, то Showing дорівнює false. Властивість може бути використана тільки для читання і на етапі виконання програми.

type TTabOrder = -1..32767;
property TabOrder: TTabOrder;
Визначає позицію компонента в послідовності табуляції компонента-батька.
Під послідовністю табуляції розуміється послідовність, у якій переключається фокус між дочірніми компонентами у вікні батька, коли користувач послідовно натискає клавішу табуляції Tab. З появою компонента-батька на екрані активним стає елемент, у якого ця властивість дорівнює нулю. Первісна послідовність табуляції визначається тією

послідовністю, у якій розміщалися дочірні елементи у вікні батька. Першому елементу привласнюється значення TabOrder, рівне 0, другому 1 і т.д. При спробі задати занадто велике чи негативне значення властивості автоматично встановлюється значення, рівне числу елементів у батьківському компоненті мінус один у першому випадку і нульове – у другому. Відзначимо, що застосування методів SendToBack і BringToFront змінює послідовність табуляції.

property TabStop: Boolean;
Якщо властивість має значення True, то компонент знаходиться в послідовності табуляції компонента-батька і може бути активізований за допомогою клавіші Tab. Якщо значення TabStop рвно False, то елемент недоступний у послідовності табуляції незалежно від значення TabOrder.

При формуванні зображення віконних елементів керування можуть бути використані наступні методи:

```
type
  TRect = record
  case Integer of
    0: (Left, Top, Right, Bottom: Integer);
    1: (TopLeft, BottomRight: TPoint);
end;
```

procedure AlignControls(AControl: TControl; var Rect: TRect); virtual;
Вирівнює всі дочірні компоненти в межах зазначеної області Rect відповідно до їхніх властивостей Align. Значення параметра AControl звичайно дорівнює nil, але може містити посилання на дочірній компонент, що буде мати пріоритет у порівнянні з іншими дочірніми компонентами при виконанні операції вирівнювання. Тип TRect визначає запис з варіантними полями. Значеннями цього типу можуть бути або записи, що мають чотири полі цілого типу - Left, Top, Right, Bottom, або записи, що мають два полі типу TPoint - TopLeft, BottomRight. У першому і в другому випадку запис типу TRect визначає на екрані прямокутну ділянку, що задається двома кутами – лівим верхнім і правим нижньої.

procedure DisableAlign;
Властивість DisableAlign тимчасово забороняє вирівнювання дочірніх компонентів усередині віконного елемента. Метод застосовується разом з методом EnableAlign, що скасовує дію DisableAlign.

function CanFocus: Boolean; dynamic;
Повертає значення True, якщо віконний елемент може одержати фокус уведення. Для того щоб елемент міг одержати фокус, у нього й у його батьків властивості Visible і Enabled повинні містити True.

procedure ChangeScale(M, D: Integer); override;
Змінює масштаб компонента і його дочірніх компонентів. Значення властивостей компонента Top, Left, Width і Height, а також розміри використовуваних шрифтів змінюються в M/D раз.

procedure EnableAlign;
Скасовує дію попередньо викликаного методу DisableAlign і викликає Realign для вирівнювання дочірніх компонентів.

function Focused: Boolean; dynamic;

PDF-XChange
Click to buy NOW!
www.docu-track.com

Повертає True, якщо віконний елемент одержав фокус уведення.

procedure EnableAlign;
Скасовує дію попередньо викликаного методу DisableAlign і викликає Realign для вирівнювання дочірніх компонентів.

procedure Realign;
Вирівнює дочірні компоненти у віконному елементі.

procedure ScaleBy(M, D: Integer);
Метод ScaleBy аналогічний методу ChangeScale, з тією різницею, що в компоненті-батькові при масштабіюванні властивості Top і Left не змінюються.

procedure ScaleControls(M, D: Integer);
Метод ScaleControls масштабує усі компоненти, що містяться у віконному елементі, не змінюючи масштабу компонента-батька. Метод ScaleControls викликає метод ChangeScale для кожного дочірнього компонента. Відрізняється від методу ScaleBy тільки тим, що не змінює масштаб самого компонента-батька.

procedure ScrollBy(Delta, Delta: Integer);
Зрушує вміст чи форми віконного елемента на Delta пікселей по горизонталі і Delta пікселей по вертикалі. Позитивні значення параметрів задають зрушення вправо і вниз, негативні значення – вліво і нагору.

procedure SetFocus; virtual;
Передає фокус уведення даному віконному елементу.

У класі TWinControl визначені події, що виникають при використанні клавіатури:

TKeyEvent = procedure (Sender: TObject; var Key: Word; Shift: TShiftState) of object;
property OnKeyDown: TKeyEvent;
Подія виникає, коли на клавіатурі натиснута клавіша. Параметр Sender містить посилання на активний елемент керування. Key – код натиснутої клавіші. Параметр Shift являє собою множину, що містить позначення натиснутої кнопки миші або натиснутих одночасно з цим допоміжних клавіш Shift, Alt, Ctrl (ssShift – натиснута клавіша Shift, ssAlt – натиснута клавіша Alt, ssCtrl – натиснута клавіша Ctrl, ssLeft – ліва кнопка миші, ssRight – права кнопка миші, ssMiddle – середня кнопка миші, ssDouble – подвійний щиглик кнопкою миші).

type TKeyPressEvent = procedure (Sender: TObject; var Key: Char) of object;
property OnKeyPress: TKeyPressEvent;
Подія виникає, коли натискається символьна клавіша. Параметр Key містить ANSI-код натиснутої клавіші.

property OnKeyUp: TKeyEvent;
Подія виникає, коли на клавіатурі відпускається натиснута клавіша.

Крім цього, у класі TWinControl визначені події, що виникають у моменти, коли віконний елемент стає активним чи перестає бути таким:

property OnEnter: TNotifyEvent;
Подія виникає, коли віконний елемент одержує фокус уведення.

property OnExit: TNotifyEvent;
Подія виникає, коли віконний елемент утрачає фокус уведення.

2.3.7. Клас TGraphicControl

Клас TGraphicControl є базовим для компонентів, що не одержують фокуса введення і, отже, не можуть виступати в якості батьківських для інших елементів керування. Нащадки класу TGraphicControl мають загальну назву – графічні елементи керування. Основне їхнє призначення – виведення на екран інформації і поліпшення зовнішнього вигляду форми застосування.

Основними представниками сімейства TGraphicControl є: Label – мітка, Shape – геометрична фігура, PaintBox – панель для малювання, Image – зображення, Bevel – тривимірне обрамлення.

Незважаючи на відсутність віконної функції, графічні елементи керування реагують на події, зв'язані з використанням миші. Це стає можливим завдяки компоненту-батькові, що є віконним елементом керування. Якщо подія, зв'язана з використанням миші, відбувається в межах дочірнього компонента, то батьківський віконний елемент передає йому повідомлення.

У класі TGraphicControl додаються одна властивість і один метод, що мають важливе значення для відображення графічного елемента керування:

property Canvas: TCanvas;
Властивість класового типу Canvas містить графічні засоби, призначені для створення на екрані зображення графічного елемента керування. Слово Canvas у перекладі означає полотно або канву. Це відповідає змісту властивості Canvas, що забезпечує простір для створення, збереження і модифікації графічних об'єктів. Більш докладно характеристики класу TCanvas будуть розглянуті в главі 2.5.

procedure Paint; virtual;
Малює зображення графічного елемента керування. У нащадках класу звичайно перевизначається для того, щоб врахувати їхні специфічні особливості.

У наступних главах компоненти Delphi будуть розглянуті більш докладно.

2.4. Текстові компоненти Label, Edit, Memo. Кнопка Button

2.4.1. Мітка Label

Ієрархія:
TObject – TPersistent – TComponent – TControl – TGraphicControl – TCustomLabel.
Сторінка Палітри Компонентів: Standard.

Мітки призначені для розміщення на екрані текстової інформації, що містить різні пояснення, назви, заголовки і т.д. Мітки в Delphi є екземплярами класу TLabel. Клас TLabel є нащадком класу TCustomLabel, у якому визначені основні характеристики міток. Цей клас є безпосереднім нащадком класу TGraphicControl і, отже, реагує тільки на події від миші.

З властивостей класу TCustomLabel відзначимо наступні.

type TAlignment = (taLeftJustify, taRightJustify, taCenter);
property Alignment: TAlignment;
Властивість визначає вирівнювання тексту в компоненті. Воно може приймати наступні значення: taLeftJustify – вирівнювання по лівому краї; taRightJustify – вирівнювання по

правому краї; taCenter – вирівнювання по центрі. За замовчуванням відбувається вирівнювання по лівому краї.

property AutoSize: Boolean;
Якщо значення властивості дорівнює True, то буде автоматично змінюватися ширина і висота мітки відповідно до розміщеного в ній текста.

type TTextLayout = (tlTop, tlCenter, tlBottom);
property Layout: TTextLayout;
Властивість визначає положення тексту мітки по вертикалі:
tlTop – розташований у верхній частині;
tlCenter – розташований у центрі;
tlBottom – розташований у низі.

property Transparent: Boolean;
Якщо властивість має значення True, то тло мітки буде прозорим стосовно інших компонентів. За замовчуванням має значення False.

property WordWrap: Boolean;
Якщо властивість має значення True, то після заповнення поточного рядка буде відбуватися перенос тексту на новий рядок. За замовчуванням має значення False.

Основним для мітки є властивість Caption, наслідувана від класу TControl. Саме сюди міститься текст, що буде виводитися на екран.

Мітка обробляє всі події, зв'язані з використанням миші і перетаскуванням компонентів. Подією за замовчуванням для мітки є подія OnClick.

2.4.2. Клас TCustomEdit

У Delphi є кілька компонентів, що дозволяють за допомогою клавіатури вводити в програму і редагувати різноманітну символьну інформацію. Усі вони мають такі можливості, як виділення, копіювання, видалення, вставка фрагментів, скролінг тексту, у тому випадку, коли він не уміщається у вікні і т.д. Із усього набору редакторів ми розглянемо рядок уведення класу TEdit і редактор тексту класу TMemo.

Більшість стандартних редакторів є нащадками класу TCustomEdit. Клас TCustomEdit, що є нащадком класу TWinControl, містить ряд характеристик, загальних для всіх текстових редакторів. Найбільш важливими є наступні властивості:

property AutoSelect: Boolean;
Якщо властивість має значення True (значення за замовчуванням), текст буде виділятися при активізації редактора.

property AutoSize: Boolean;
Якщо властивість має значення True, то буде автоматично змінюватися висота редактора відповідно до розміру шрифту.

type
TFormBorderStyle = (bsNone, bsSingle, bsSizeable, bsDialog, bsToolWindow, bsSizeToolWin);
TBorderStyle = bsNone..bsSingle;
property BorderStyle: TBorderStyle;
Визначає вид границі редактора: bsSingle – одинарна границя, bsNone – немає границі. За замовчуванням редактор має одинарну границю.

PDF-XChange
Click to buy NOW!
www.docu-track.com

type TEditCharCase = (ecNormal, ecUpperCase, ecLowerCase);
property CharCase: TEditCharCase;
Визначає вид букв тексту: ecUpperCase – великі, ecLowerCase – малі, ecNormal – змішані.

property MaxLength: Integer;
Визначає максимальне число символів, яке можна помістити в редактор. Якщо значення властивості дорівнює нулю (використовується за замовчуванням), то в редактор можна помістити довільне число символів.

property PasswordChar: Char;
Задає символ, що буде відображатися замість символів, що реально вводяться, у редакторі. Таке поводження редактора може придатися, якщо в програмі потрібне використання пароля. Якщо значенням властивості є символ з кодом 0 (значення за замовчуванням), то символи, що вводяться, відображаються без перетворення.

property ReadOnly: Boolean;
Якщо властивість має значення True, то текст призначений тільки для читання, тобто змінювати текст не можна. За замовчуванням має значення False.

У класі TCustomEdit є ряд методів, що реалізують функції редагування:

procedure Clear; virtual;
Видаляє весь текст, поміщений у редактор.

procedure ClearSelection;
Видаляє виділений фрагмент тексту.

procedure CopyToClipboard;
Копіює виділений фрагмент тексту в буфер Clipboard.

procedure CutToClipboard;
Видаляє з тексту виділений фрагмент і поміщає його в буфер Clipboard.

procedure PasteFromClipboard;
Копіює текст із буфера Clipboard у позицію курсору редактора.

procedure SelectAll;
Виділяє весь текст, що міститься в редакторі.

У більшості редакторів є убудовані локальні меню, що дозволяють виконувати наступні команди:

Undo – скасування останньої виконаної операції в тексті;
Cut – вирізувати виділений фрагмент із поміщенням його в буфер обміну;
Copy – скопіювати виділений фрагмент у буфер обміну;
Paste – помістити фрагмент із буфера обміну в текст, у позицію курсору;
Delete – видалити виділений фрагмент;
Select All – виділити весь текст.

2.4.3. Рядок уведення Edit

Ієрархія:
TObject – TPersistent – TComponent – TControl – TWinControl – TCustomEdit.

Сторінка Палітри Компонентів: Standard.

Рядок уведення Edit, що є екземпляром класу TEdit, дозволяє вводити і редагувати один рядок тексту.

Клас TEdit – безпосередній нащадок класу TCustomEdit і успадковує всі його характеристики, розглянуті в пункті 2.4.2.

Основною властивістю рядка введення є властивість Text:

type TCaption = string;
property Text: TCaption;
Містить символьний рядок у редакторі Edit.

Символьний рядок може бути поміщений у властивість Text або на етапі конструювання форми, або під час виконання програми. Найчастіше редактор використовується для введення інформації. У цьому випадку на етапі конструювання форми у властивість Text можна помістити порожній рядок, а на етапі виконання програми витягати з нього введені значення. Якщо вводяться символьні представлення числових даних, то надалі вони повинні бути перетворені за допомогою відповідних підпрограм перетворення типу.

Оскільки рядок уведення є віконним елементом керування, то він обробляє всі події від миші і клавіатури, події, зв'язані з перетаскуванням, активізацією і зняттям активізації, а також подію OnChange:

property OnChange: TNotifyEvent;
Виникає при зміні тексту рядка введення. Є подією за замовчуванням для рядка введення класу TEdit.

2.4.4. Текстовий редактор Memo

Ієрархія:
TObject – TPersistent – TComponent – TControl – TWinControl – TCustomEdit – TCustomMemo.
Сторінка Палітри Компонентів: Standard.

Текстовий редактор Memo може містити на відміну від рядка введення Edit не одну, а будь-яке число рядків. Редактор Memo є екземпляром класу TMemo. У свою чергу клас TMemo породжений безпосередньо від класу TCustomMemo, у якому визначені основні характеристики багаторядкових текстових редакторів.

Відзначимо наступні властивості класу TCustomMemo:

property Lines: TStrings;
Властивість класового типу TStrings. Задає список рядків, поміщених у редактор. Клас TStrings буде розглянутий у пункті 2.4.5.

type TScrollStyle = (ssNone, ssHorizontal, ssVertical, ssBoth);
property ScrollBars: TScrollStyle;
Задає наявність лінійок скролінга:
ssNome – немає лінійок,
ssHorizontal – тільки горизонтальна лінійка,
ssVertical – тільки вертикальна лінійка,
ssBoth – обидві лінійки.

PDF-XChange
Click to buy NOW!
www.docu-track.com

property WantReturns: Boolean;
Визначає дію клавіші Enter. Якщо властивість дорівнює True, то при натисканні на клавішу Enter відбувається перехід на новий рядок тексту. У противному випадку фокус уведення передається формі, а перехід на новий рядок у тексті здійснюється натисканням комбінації клавіш Ctrl+Enter.

Так само, як і рядок уведення Edit, багаторядковий текстовий редактор Memo обробляє всі події від миші і клавіатури, події, зв'язані з перетаскуванням, активізацією і зняттям активізації, а також подію OnChange, що є подією за замовчуванням.

2.4.5. Клас TStrings

TStrings – безпосередній нащадок класу TPersistent. Є абстрактним класом. Об'єкти цього класу являють собою списки рядків. З кожним рядком може бути зв'язаний деякий об'єкт, наприклад малюнок чи піктограма. Якщо з рядком не зв'язаний ніякий об'єкт, то вказівник на нього дорівнює nil. Клас TStrings є класом загального призначення і використовується для визначення властивостей відповідного типу в багатьох компонентах Delphi.

Розглянемо основні властивості класу TStrings:

property Count: Integer;
Визначає число елементів у списку.

property Objects[Index: Integer]: TObject;
Визначає вказівник на об'єкт, асоційований з рядком з індексом Index.

property Strings[Index: Integer]: string;
Визначає рядок списку з індексом Index. Індекс першого рядка – 0.

property Text: string;
Містить усі рядки списку, включаючи роздільники – символи повернення каретки і переведення рядка (#13#10).

Основні методи, визначені в класі TStrings:

function Add(const S: string): Integer; virtual;
Додає рядок S у список і повертає порядковий номер цього рядка в списку.

function AddObject(const S: string; AObject: TObject): Integer; virtual;
Додає рядок S і зв'язаний з ним об'єкт AObject у список і повертає індекс рядка й об'єкта в списку.

procedure AddStrings(Strings: TStrings); virtual;
Додає список іншого об'єкта Strings класу TStrings до поточного списку.

procedure BeginUpdate;
Фіксує початок відновлення списку. При цьому тимчасово припиняються перемальовування на екрані.

procedure Clear; virtual; abstract;
Видаляє всі рядки і вказівники на об'єкти зі списку.

procedure Delete(Index: Integer); virtual; abstract;

Видаляє зі списку елемент з індексом Index.

procedure EndUpdate;
Фіксує завершення відновлення списку і при необхідності сортує рядки.

procedure Exchange(Index1, Index2: Integer); virtual;
Змінює місцями два елементи списку з індексами Index1 і Index2.

procedure Insert(Index: Integer; const S: string); virtual; abstract;
Вставляє в список рядок S під індексом Index.

procedure InsertObject(Index: Integer; const S: string; AObject: TObject);
Вставляє в список рядок S і об'єкт AObject під індексом Index.

procedure LoadFromFile(const FileName: string); virtual;
Завантажує список з файлу з ім'ям FileName.

procedure Move(CurIndex, NewIndex: Integer); virtual;
Переміщає елемент списку з позиції CurIndex у позицію NewIndex.

procedure SaveToFile(const FileName: string); virtual;
Поміщає список у файл з ім'ям FileName.

2.4.6. Кнопка Button

Ієрархія:
TObject – TPersistent – TComponent – TControl – TWinControl – TButtonControl.
Сторінка Палітри Компонентів: Standard.

Основне призначення кнопки - формування події при натисканні на неї. Кнопка може бути розміщена в будь-якім місці форми, де існує необхідність виконати які-небудь дії при її натисканні. Кнопка Button є екземпляром класу TButton, предками якого є класи TButtonControl і TWinControl. Це означает. що кнопка Button являє собою віконний злемент керування.

У класі TButton додані наступні властивості:

property Cancel: Boolean;
Якщо властивість має значення True, то натискання на клавішу Esc буде еквівалентно натисканню на дану кнопку.

property Default: Boolean;
Якщо властивість має значення True, то натискання на клавішу Enter буде еквівалентно натисканню на дану кнопку, якщо яка-небудь інша кнопка не знаходиться у фокусі введення.

type TModalResult = Low(Integer)..High(Integer);
property ModalResult: TModalResult;
Властивість, використовувана при закритті модальних вікон. Для звичайних вікон значення цієї властивості повинне бути дорівнює mrNone.

У класі TButton визначений метод Click:

procedure Click; override;

Виконання цього методу еквівалентно щиглику по кнопці, тобто в результаті його виконання виникає подія OnClick для кнопки.

Особливістю кнопки класу TButton є те, що вона не має властивості Color і колір тла для неї визначається операційною системою Windows.

Кнопка класу TButton обробляє всі події, визначені для віконного елемента керування. Подією за замовчуванням для кнопки є подія OnClick.

2.4.7. Приклад використання компонентів Label, Edit, Memo і Button

Приклад 2.2.

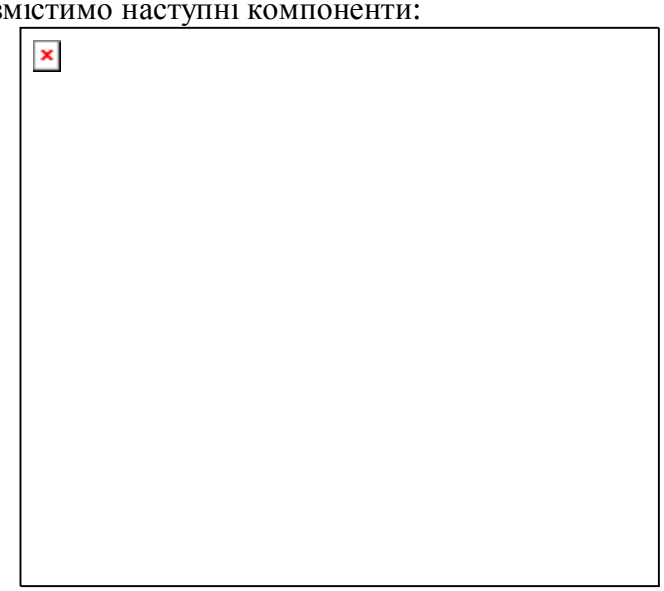
Скласти програму, що обчислює корені квадратного рівняння

$$a_2x^2+a_1x+a_0=0$$

с довільними коефіцієнтами ($a_2\neq0$).

Рішення.

1. Створимо папку D:\MyProject\Text (можна будь-яку іншу).
2. Відкриємо нове застосування за допомогою команди головного меню File|New|Application.
3. На формі Form1 розмістимо наступні компоненти:



Мал. 2.12. Розміщення компонентів класів TLabel, TEdit, TMemo і TButton на формі

Усі компоненти беремо зі сторінки Standard Палітри Компонентів.

4. Властивості Caption мітки Label1 задамо значення:

Уведіть коефіцієнти квадратного рівняння
A2*X^2+A1*X+A0=0

Для того щоб текст розташовувався в двох рядках і був вирівняний по центрі для мітки необхідно установити наступні значення властивостей:

AutoSize – False,
WordWrap – True,
Alignment – taCenter.

Розміри мітки і її місце розташування можна відрегулювати вручну.

Перейдемо у властивість Font і клацнемо по кнопці з трьома крапками. У вікні, що з'явилося, змінимо розмір шрифту з 8 на 10. Натиснемо OK.

5. Для міток Label2, Label3 і Label4 установимо властивість Caption рівною 'A0=', 'A1=' і 'A2=' відповідно.

6. Для компонентів Edit1, Edit2 і Edit3 установимо значення властивості Text рівною порожньому рядку.

7. Для компонента Memo1 виберемо властивість Lines і клацнемо по кнопці з трьома крапками. У вікні String list editor, що з'явилося, видалимо рядок 'Memo1'. Натиснемо ОК.
8. Кнопкам Button1 і Button2 установимо властивість Caption рівною 'Рішення' і 'Очистити' відповідно.
9. Подвійним щигликом активізуємо кнопку **Рішення** і для оброблювача події OnClick уставимо наступні рядки:

```
procedure TForm1.Button1Click(Sender: TObject);
var a0,a1,a2,d,x1,x2:double;
begin
  a0 := StrToFloat(Edit1.Text);
  a1 := StrToFloat(Edit2.Text);
  a2 := StrToFloat(Edit3.Text);
  d := a1*a1-4*a2*a0;
  if d>=0 then
    begin
    x1 := (-a1+sqrt(d))/(2*a2);
    x2 := (-a1-sqrt(d))/(2*a2);
    Memo1.Lines.Add('Результат:');
    Memo1.Lines.Add('x1='+FloatToStrF(x1,ffGeneral,7,2));
    Memo1.Lines.Add('x2='+FloatToStrF(x2,ffGeneral,7,2));
    end;
  if d<0 then
    Memo1.Lines.Add('Рішень немає:');
end;
```

Нагадаємо, що напівжирним шрифтом ми виділяємо рядки, набрані програмістом.
10. Натиснемо клавішу F12 і повернемося у форму. Подвійним щигликом активізуємо кнопку **Очистити** й в оброблювач події OnClick уставимо рядок

```
Memo1.Clear;
```

11. Збережемо проект у папці D:\MyProject\Text і запустимо застосування на виконання, використовуючи функціональну клавішу F9. Для рішення квадратного рівняння введемо коефіцієнти і натиснемо кнопку **Рішення**. Для очищення вікна редактора натиснемо кнопку **Очистити (**див. мал.2.13).

Мал. 2.13. Форма для рішення квадратного рівняння

Приведемо повний текст сформованого модуля.

Текст модуля Unit1.pas.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    Label4: TLabel;
    Edit1: TEdit;
    Edit2: TEdit;
    Edit3: TEdit;
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var a0,a1,a2,d,x1,x2:double;
begin
  a0 := StrToFloat(Edit1.Text);
  a1 := StrToFloat(Edit2.Text);
  a2 := StrToFloat(Edit3.Text);
  d := a1*a1-4*a2*a0;
  if d>=0 then
    begin
    x1 := (-a1+sqrt(d))/(2*a2);
    x2 := (-a1-sqrt(d))/(2*a2);
    Memo1.Lines.Add('Результат:');
    Memo1.Lines.Add('x1='+FloatToStrF(x1,ffGeneral,7,2));
    Memo1.Lines.Add('x2='+FloatToStrF(x2,ffGeneral,7,2));
    end;
```

```
  if d<0 then
    Memo1.Lines.Add('Рішень немає:');
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
Memo1.Clear;
end;

end.
```

2.5. Класи і компоненти Delphi, призначені для створення зображень. Компонент Timer – таймер

2.5.1. Загальна характеристика

У Delphi є кілька класів загального призначення, що дозволяють виводити графічні зображення на поверхню компонентів. До них відносяться класи: TFont (шрифт), TPen (олівець), TBrush (кисть) і TCanvas (канва). У складі багатьох компонентів Delphi є властивості Font, Pen, Brush і Canvas відповідного типу, за допомогою яких створюються зображення будь-яких малюнків і текстів. Предком класів TFont, TPen і TBrush є клас TGraphicsObject, що походить безпосередньо від класу TPersistent. Предком класу TCanvas є клас TPersistent.
У класі TGraphicsObject відзначимо подію OnChange:

property OnChange: TNotifyEvent;
Подія виникає при зміні графічного об'єкта. Після виникнення події графічні об'єкти відображаються з обновленими значеннями своїх властивостей.

У Delphi маються компоненти, спеціально призначені для створення графічних зображень:

Image – являє собою зручний засіб для відображення готових графічних файлів.

Shape – використовується для створення на формі простих геометричних фігур – квадратів, кіл, еліпсів і т.п.

PaintBox – дозволяє на етапі виконання програми створювати нескладні малюнки на своїй канві Canvas.

Крім того, у цій главі ми розглянемо компонент Timer, що призначений для відліку інтервалів реального часу. Він не призначений для створення графічних зображень і є допоміжним компонентом. Нами він буде використаний при створенні проектів, що містять графічні компоненти.

2.5.2. Клас TFont

Клас TFont визначає характеристики шрифту. Властивості, що мають цей класовий тип, присутні в будь-якому компоненті, що може містити деякий текст. Характеристики шрифту в класі TFont задаються за допомогою наступних властивостей:

type TFontCharset = 0..255;
property Charset: TFontCharset nodefault;
Визначає набір символів шрифту. Приведемо деякі константи, що можуть бути використані як значення властивості Charset:

Константа	Значення	Опис
ANSI_CHARSET	0	Символи ANSI.
DEFAULT_CHARSET	1	Задається за замовчуванням. Шрифт вибирається

PDF-XChange Click to buy NOW! www.docu-track.com

		тільки по його імені Name і розміру Size. Якщо описаний шрифт недоступний у системі, то Windows замінить його іншим шрифтом.
SYMBOL_CHARSET	2	Стандартний набір символів.
MAC_CHARSET	77	Символи Macintosh. Недоступні для NT 3.51.
GREEK_CHARSET	161	Грецькі символи. Недоступні для NT 3.51.
RUSSIAN_CHARSET	204	Символи кирилиці. Недоступні для NT 3.51.
OEM_CHARSET	255	Визначається кодовою таблицею операційної системи.

property Color: TColor;
Визначає колір символів.

property Height: Integer;
Установлює висоту шрифту в пікселях.

type TFontName = type string;
property Name: TFontName;
Задає ім'я шрифту.

type TFontPitch = (fpDefault, fpVariable, fpFixed);
property Pitch: TFontPitch;
Задає ширину шрифту і може приймати наступні значення :
fpDefault – ширина задається типом шрифту,
fpVariable – ширина символів перемінна,
fpFixed – ширина символів фіксована.

property Size: Integer;
Задає висоту шрифту в пунктах (1 пункт = 1/72 дюйма).

```
type
  TFontStyle = (fsBold, fsItalic, fsUnderline, fsStrikeOut);
  TFontStyles = set of TFontStyle;
```
property Style: TFontStyles;
Задає тип шрифту і як значення може приймати будь-яку множину наступних величин :
fsBold – напівжирний,
fsItalic – курсив,
fsUnderline – підкреслений,
fsStrikeOut – перекреслений.

Помітимо, що тексти в Delphi пишуться тільки горизонтально.

2.5.3. Клас TPen

Клас TPen задає характеристики олівця, за допомогою якого створюються зображення різних ліній або контурів.

До основних властивостей цього класу можна віднести наступні:

property Color: TColor;
Визначає колір лінії, що малюється олівцем. За замовчуванням колір чорний.

type

TPenMode = (pmBlack, pmWhite, pmNop, pmNot, pmCopy, pmNotCopy, pmMergePenNot, pmMaskPenNot, pmMergeNotPen, pmMaskNotPen, pmMerge, pmNotMerge, pmMask, pmNotMask, pmXor, pmNotXor);
property Mode: TPenMode;
Визначає стиль малювання ліній олівцем. Може приймати наступні значення:

pmBlack – завжди зображується чорна лінія;
pmWhite – завжди зображується біла лінія;
pmNop – безбарвна лінія;
pmNot – колір, інверсний кольору екрана;
pmCopy – колір визначається значенням властивості Color;
pmNotCopy – колір, інверсний кольору, заданому у властивості Color;
pmMergePenNot – комбінація кольору Color і інверсного кольору екрана;
pmMaskPenNot – комбінація кольорів, загальних у Color і інверсного кольору екрана;
pmMergeNotPen – комбінація кольору екрана й інверсного кольору Color;
pmMaskNotPen – комбінація кольорів, загальних у кольору екрана й інверсного кольору Color;
pmMerge – комбінація кольорів екрана і Color;
pmNotMerge – колір, інверсний комбінації кольорів екрана і Color;
pmMask – комбінація кольорів, загальних в екрана і Color;
pmNotMask – колір інверсний комбінації кольорів, загальних в екрана і Color;
pmXor – комбінація кольорів, що є присутнім у кольору чи екрана Color, але не одночасно в обох;
pmNotXor – колір, інверсний комбінації кольорів, що є присутнім у кольору чи екрана Color, але не одночасно в обох.

За замовчуванням колір лінії визначається властивістю Color.

type TPenStyle = (psSolid, psDash, psDot, psDashDot, psDashDotDot, psClear, psInsideFrame);
property Style: TPenStyle;
Визначає стиль лінії, що малюється олівцем. Може приймати наступні значення :

psSolid – суцільна лінія (значення за замовчуванням),
psDash – штрихова лінія,
psDot – пунктирна лінія,
psDashDot – штрихпунктирная лінія,
psDashDotDot – штрихпунктирная лінія з двома пунктирами,
psClear – невидима лінія,
psInsideFrame – лінія усередині замкнутої рамки.

property Width: Integer;
Визначає товщину лінії, що малюється. Значенням за замовчуванням є товщина, рівна 1 пікселю.

2.5.4. Клас TBrush

Клас TBrush містить визначення характеристик кисті, що використовується для заливання (зафарбування) замкнутих областей. Розглянемо основні властивості кисті.

property Bitmap: TBitmap;
Bitmap указує на об'єкт типу TBitmap, що містить побітове зображення, розміром 8x8 пікселей. Якщо Bitmap не порожній, то шаблон заповнення визначається саме їм, а не властивістю Style. У противному випадку властивість повинна мати значення nil.

property Color: TColor;
Визначає колір кисті. Значенням за замовчуванням є білий колір.

PDF-XChange
Click to buy NOW!
www.docu-track.com

type TBrushStyle = (bsSolid, bsClear, bsHorizontal, bsVertical, bsFDiagonal, bsBDiagonal, bsCross, bsDiagCross);
property Style:TBrushStyle;
Визначає орнамент кисті. Може приймати наступні значення:

bsSolid – суцільне розфарбування,
bsClear – відсутність розфарбування,
bsHorizontal – горизонтальні лінії,
bsVertical – вертикальні лінії,
bsFDiagonal – ліві діагональні лінії,
bsBDiagonal – праві діагональні лінії,
bsCross – клітка,
bsDiagCross – коса клітка.

2.5.5. Клас TCanvas

Клас TCanvas визначає об'єкт Canvas, що являє собою поверхню компонента, використовувану для малювання, і інструменти, за допомогою яких створюється зображення: шрифт (клас TFont), олівець (клас TPen) і кисть (клас TBrush). Об'єкти класового типу TCanvas не є компонентами і використовуються як властивості різних елементів керування, наприклад форми.

Канва складається з окремих крапок – пікселей. Кожен піксель має горизонтальну і вертикальну координату. Початок координат, тобто крапка з координатами (0,0), розташовується в лівому верхньому куті канви. Горизонтальна вісь спрямована зліва направа, а вертикальна – зверху вниз. Розмір канви залежить від розміру й особливостей компонента. Так, наприклад, для компонента Image розмір канви визначається властивостями Height і Width, а для форми – властивостями ClientHeight і ClientWidth.

На канві мається невидимий графічний курсор, що визначає поточне положення олівця. Як правило, малювання графічних примітивів – ліній, кіл, прямокутників і т.д. - починається з поточного положення цього курсору. У процесі малювання положення курсору змінюється. У класі TCanvas наявні методи, що дозволяють установлювати курсор у задане положення.

Розглянемо основні властивості TCanvas.

property Brush: TBrush;
Задає кисть канви.

property Font: TFont;
Задає шрифт канви.

property Pen: TPen;
Задає олівець канви.

property PenPos: TPoint;
Визначальне поточне положення графічного курсору.

property Pixels[X, Y: Integer]: TColor;
Задає кольори всіх пікселей канви; x і y – координати пікселя.

У класі TCanvas визначено багато різних методів, призначених для малювання всіляких геометричних фігур. Усі геометричні фігури можна умовно розбити на контурні (тобто не мають внутрішнього зафарбування) і зафарбовані (мають внутрішнє зафарбування). При малюванні контурних фігур використовується тільки олівець Pen із встановленими в ньому

PDF-XChange
Click to buy NOW!
www.docu-track.com

характеристиками (колір лінії, товщина і т.д.). Якщо фігура є зафарбованою (наприклад еліпс, багатокутник), її внутрішність зафарбовується кистю Brush із встановленими в ній характеристиками (колір, орнамент і т.д.). Тексти зображуються відповідно до характеристик (накреслення, розмір і т.д.), заданими в шрифті Font. На етапі виконання програми значення властивостей Pen, Brush і Font можна змінювати. Але варто мати на увазі, що установки повинні бути виконані до того, як вони будуть використані в процесі малювання.

Розглянемо основні методи, призначені для створення простих графічних зображень:

procedure Arc(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Integer);
Малює дугу еліпса, що уписаний у прямокутник з координатами лівого верхнього кута (X1,Y1) і правого нижнього кута (X2,Y2). Дуга починається в крапці перетинання еліпса з прямою, що проходить через центр еліпса і крапку з координатами (X3,Y3), а закінчується в крапці перетинання еліпса з прямої, що проходить через центр еліпса і крапку з координатами (X4,Y4).

procedure Ellipse(X1, Y1, X2, Y2: Integer); overload;
Малює еліпс, уписаний у прямокутник з координатами лівого верхнього кута (X1,Y1) і правого нижнього кута (X2,Y2).

type TFillStyle = (fsSurface, fsBorder);
procedure FloodFill(X, Y: Integer; Color: TColor; FillStyle: TFillStyle);
Зафарбовує кистю Brush поверхню канви, починаючи з крапки з координатами (X,Y). Зафарбовується поверхня до границі, заданих кольором Color, якщо параметр FillStyle має значення fsBorder, або зафарбовується ділянка поверхні, що має колір Color, якщо параметр FillStyle має значення fsSurface.

procedure LineTo(X, Y: Integer);
Проводить лінію олівцем Pen з поточного положення графічного курсору, обумовленого властивістю Penpos, до крапки з координатами (X,Y). Властивість Penpos одержує нове значення, обумовлене координатами (X,Y).

procedure MoveTo(X, Y: Integer);
Переміщає графічний курсор у крапку з координатами (X,Y).

procedure Pie(X1, Y1, X2, Y2, X3, Y3, X4, Y4: Longint);
Малює сектор еліпса. Дуга еліпса задається так само, як і у випадку методу Arc.

procedure Rectangle(X1, Y1, X2, Y2: Integer); overload;
Малює прямокутник, у якого лівий верхній кут має координати (X1,Y1), а правий нижній кут – координати (X2,Y2).

procedure RoundRect(X1, Y1, X2, Y2, X3, Y3: Integer);
Малює прямокутник, у якого лівий верхній кут має координати (X1,Y1), а правий нижній кут – координати (X2,Y2). Кути прямокутника закруглені частинами еліпса з осями X3 і Y3.

procedure TextOut(X, Y: Integer; const Text: string);
Виводить текст Text на поверхню канви таким чином, що координати (X,Y) є координатами лівого верхнього кута прямокутника, у якому виводиться текст. Параметри тексту задаються характеристиками шрифту Font, колір тла – поточним кольором кисті Brush.

function TextHeight(const Text: string): Integer;
Повертає висоту тексту Text у пікселях, обумовлену шрифтом Font.

PDF-XChange
Click to buy NOW!
www.docu-track.com

function TextWidth(const Text: string): Integer;
Повертає довжину тексту Text у пікселях, обумовлену шрифтом Font.

У класі TCanvas визначені події OnChange і OnChanging:

property OnChange: TNotifyEvent;
Виникає перед тим, як у канві повинні бути зроблені зміни.

property OnChanging: TNotifyEvent;
Виникає відразу ж після того, як у канві зроблені зміни.

2.5.6. Компонент Image

Ієрархія:
TObject – TPersistent – TComponent – TControl – TGraphicControl.
Сторінка Палітри Компонентів: Additional.

Компонент Image класу TImage використовується для розміщення на формі деякої картинки. Файл зображення може бути бітовою картою (файл із расширением.bmp), піктограмою (файл із расширением.ico), метафайлом (файл із расширением.wmf).

Клас TImage є безпосереднім нащадком класу TGraphicControl і, отже, належить до сімейства графічних елементів керування. Крім наслідуваних, у класі TImage визначені наступні властивості:

property Canvas: TCanvas;
Призначається для формування зображення на етапі виконання програми.

property Center: Boolean;
Якщо властивість має значення True, зображення вирівнюється по центрі компонента, у противному випадку зображення міститься в лівому верхньому куті компонента. За замовчуванням має значення False.

property Picture: TPicture;
Властивість класового типу TPicture – визначає зображення, поміщене в компоненті.

property Stretch: Boolean;
Якщо властивість має значення True, зображення обумовлене властивістю Picture, збільшується чи зменшується до розмірів компонента. За замовчуванням має значення False.

Як і інші графічні елементи керування, компонент Image обробляє всі події від миші. Події за замовчуванням не має.

При поміщенні зображення в компонент Image на етапі проектування можна використовувати вікно завдання зображення. Це вікно розкривається при виборі властивості Picture в Інспекторі Об'єктів. Основне поле вікна завдання зображення призначено для розміщення обраної картинки. Крім цього, у вікні присутні наступні кнопки:

Load – для завантаження зображення з файлу,
Save – для записування зображення у файл,
Clear – для видалення обраного зображення,
OK – для записування в компонент обраного зображення,
Cancel – для скасування введених змін.

PDF-XChange
Click to buy NOW!
www.docu-track.com

2.5.7. Компонент Shape

Ієрархія:
TObject – TPersistent – TComponent – TControl – TGraphicControl.
Сторінка Палітри Компонентів: Additional.

Екземплярами класу TShape є компоненти-фігури – кола, еліпси, прямокутники і т.п. Ці фігури можуть бути використані для стилізації вашого застосування.

Клас TShape є безпосереднім нащадком класу TGraphicControl і так само, як і клас TImage, входить у сімейство графічних елементів керування. Крім успадкованих, у класі TShape визначені наступні властивості:

property Brush: TBrush;
Визначає кисть для зафарбовування поверхні фігури.

property Pen: TPen;
Визначає олівець для малювання контуру фігури.

Type TShapeType = (stRectangle, stSquare, stRoundRect, stRoundSquare, stEllipse, stCircle);
property Shape: TShapeType;
Визначає фігуру, виведену на екран:

- stRectangle – прямокутник,
- stSquare – квадрат,
- stRoundRect – прямокутник із закругленими краями,
- stRoundSquare – квадрат із закругленими краями,
- stEllipse – еліпс,
- stCircle – коло.

Подією за замовчуванням для компонента Shape є подія OnDragDrop.

2.5.8. Компонент Paint Box

Ієрархія:

TObject – TPersistent – TComponent – TControl – TGraphicControl.
Сторінка Палітри Компонентів: System.

Компонент Paint Box класу TPaintBox надає можливість малювати в обмеженій області форми. Для малювання використовуються всі можливості канви – шрифт, олівець, кисть, а також методи, що дозволяють будувати геометричні фігури на етапі виконання програми.

Клас TPaintBox є нащадком класу TGraphicControl і поряд з раніше розглянутими графічними компонентами входить у сімейство графічних елементів керування.

Серед характеристик, визначених у класі TPaintBox відзначимо подію OnPaint:

property OnPaint: TNotifyEvent;
Виникає перед тим, як компонент повинний бути перемальований.

Для компонента Paint Box подією за замовчуванням є подія OnClick.

2.5.9. Компонент Timer

PDF-XChange
Click to buy NOW!
www.docu-track.com

Ієрархія:
TObject – TPersistent – TComponent.
Сторінка Палітри Компонентів: System.

Таймер класу TTimer є невізуальним компонентом, і його можна віднести до розряду допоміжних компонентів. Таймер призначений для ініціювання якої-небудь операції через задані проміжки часу.

Розглянемо основні властивості класу TTimer:

property Enabled: Boolean;
Якщо властивість має значення True, то таймер реагує на власну подію OnTimer.

property Interval: Cardinal;
Визначає часовий інтервал у мілісекундах, після якого з'являється подія OnTimer. За замовчуванням дорівнює 1000 (1 секунда).

Крім того, у класі TTimer визначена подія OnTimer:

property OnTimer: TNotifyEvent;
Виникає періодично після закінчення інтервалу часу, обумовленого властивістю Interval. Вона є подією, використовуваною за замовчуванням.

2.5.10. Приклади використання компонентів Image, Shape, Paint Box і Timer
Приклад 2.3.

Створити заставку для застосування.

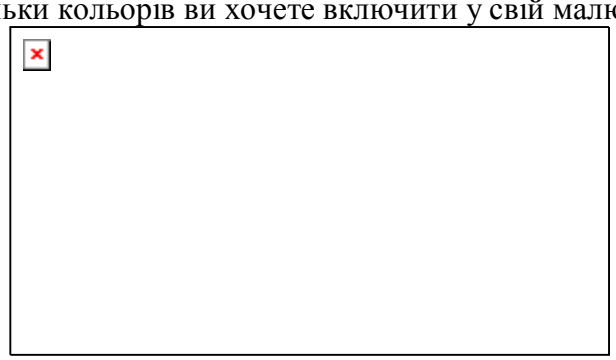
Рішення

Заставки мають багато програм, що працюють в операційних системах сімейства Windows. Заставка являє собою графічне зображення, що з'являється на кілька секунд після запуску програми. У ній може міститися назва програмного продукту й інформація про розроблювачів. Графічне зображення, тобто файл із расширением bmp, створимо за допомогою графічного редактора. Перед запуском Delphi створимо папку D:\MyProject\GRAFICA.

1. Відкриємо будь-яке з раніше створених застосувань чи створимо нове.
2. Графічний редактор запустимо за допомогою команди головного меню Tools|Image Editor (мал.2.14) .

Мал. 2.14. Вікно графічного редактора, що входить у середовище Delphi

3. Робота в графічному редакторі Delphi мало чим відрізняється від роботи у відомому графічному редакторі Paint. При створенні графічного файлу ви можете скористатися одним із двох способів – або створити новий файл, або внести зміни в існуючий. Виберемо перший спосіб. Виконаємо команду головного меню File|New|Bitmap File (.bmp). У діалозі Bitmap Properties, що з'явився, у поле Width (ширина) установимо - 300, а в поле Height (висота) - 200. У групі перемикачів Colors виберемо перемикач VGA (16 colors) чи будь-який інший, у залежності від того, скільки кольорів ви хочете включити у свій малюнок (див. мал. 2.15) .



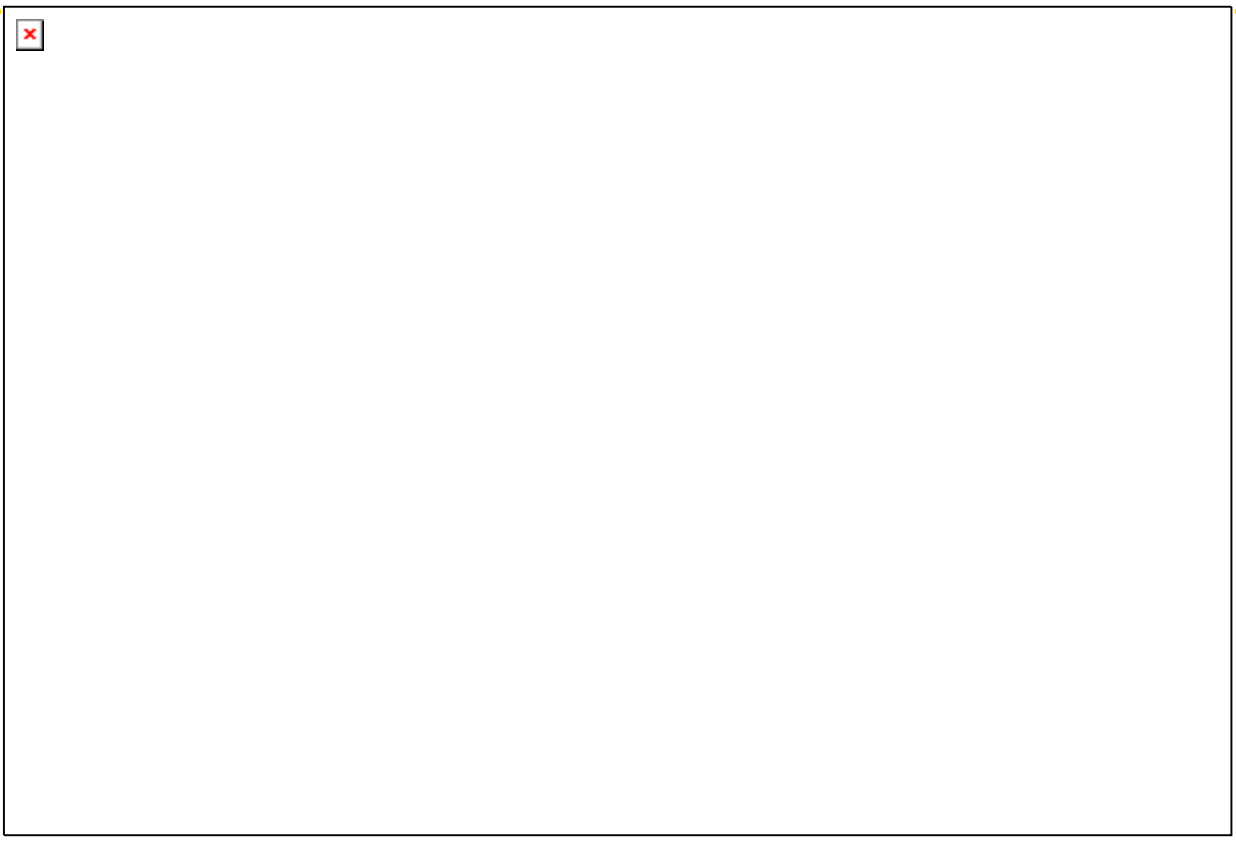
Мал. 2.15. Діалогове вікно для установки параметрів малюнка

Натиснемо ОК.

4. Створимо довільне зображення. Для того щоб вибрати колір символів, варто підвести курсор миші до вікна з необхідним кольором у палітрі кольорів і натиснути ліву клавішу. При натиснутій правій клавіші миші обраний колір буде використовуватися як колір тла. У найпростішому випадку можна зробити, наприклад, так: набрати чорними буквами на білому тлі текст:

Розроблювач – Петренко П.П.
Приклад створення
ЗАСТАВКИ

Виберемо чорний колір для символів, білий для тла в палітрі квітів. Виконаємо команди Edit|Select All і Edit|Cut для очищення малюнка. Далі вставимо приведений вище текст (мал. 2.16).



Мал. 2.16. Зовнішній вигляд створюваної заставки

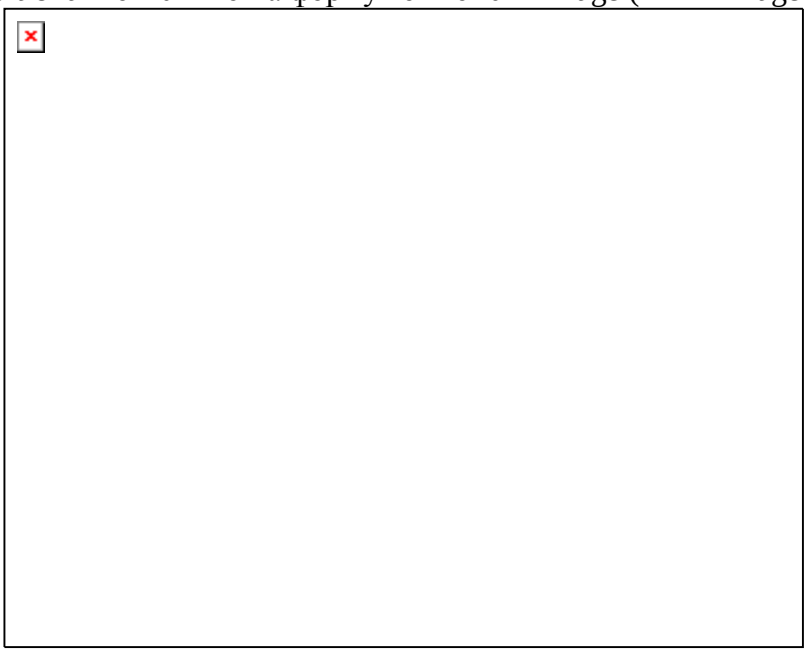
Для цього скористаємося кнопкою  (**Text)** інструментальної панелі. Перш ніж розміщати текст, варто задати характеристики шрифту за допомогою команди Text|Font.

5. Виконаємо команду File|Save і збережемо графічний файл у папці D:\MyProject\GRAFICA під ім'ям ris.bmp .

Вийдемо з графічного редактора по команді File|Exit.

6. Зі сторінки System помістимо на форму Form1 компонент Timer. Він одержить ім'я Timer1. Властивість Interval установимо рівною 3000.

7. Зі сторінки Additional помістимо на форму компонент Image (ім'я – Image1) (мал.2.17).



Мал. 2.17. Розміщення на формі компонентів Timer і Image

Виберемо властивість Picture і натиснемо кнопку з трьома крапками. З'явиться вікно Picture Editor. Натиснемо кнопку Load. Ввійдемо в папку D:\MyProject\GRAFICA і виберемо файл

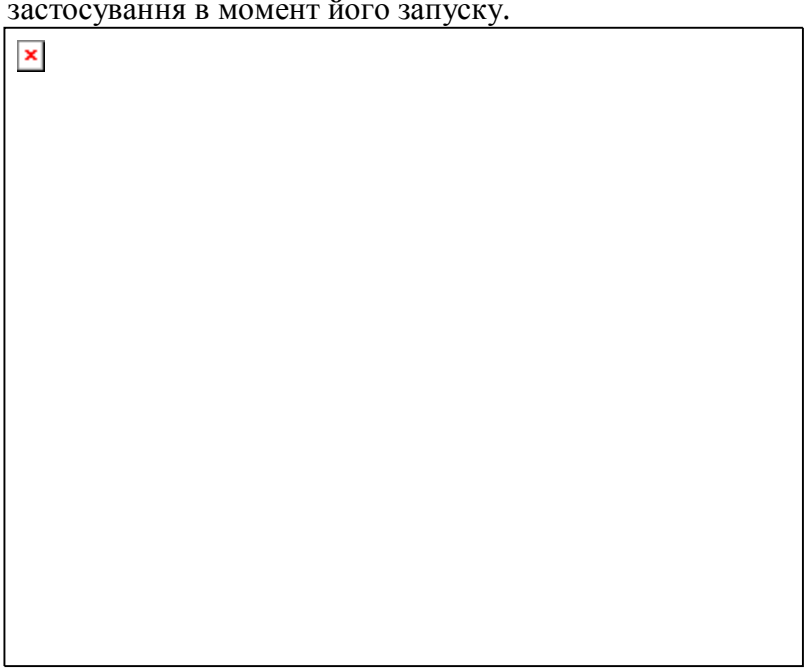
ris.bmp . В Picture Editor натиснемо ОК. Для того щоб малюнок був видний цілком, установимо властивість Autosize у True. Після цього розташуємо малюнок так, щоб він знаходився в центрі форми.

8. Активізуємо компонент Timer1 подвійним щигликом і створимо наступний обраблювач події OnTimer:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Image1.Free;
  Timer1.Free
end;
```

Це означає, що через 3000 мілісекунд після запуску застосування компоненти Image1 і Timer1 будуть вилучені з пам'яті комп'ютера і відповідно – з екрана.

9. Збережемо проект під ім'ям Project1 (можна під будь-яким іншим) і модуль під ім'ям Unit1 у папці D:\MyProject\GRAFICA. Запустимо проект на виконання. На мал. 2.18. зображений зовнішній вигляд застосування в момент його запуску.



Мал. 2.18. Заставка, що з'являється в момент запуску застосування

Приведемо повний текст сформованого модуля.

Текст модуля Unit1.pas.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
  ExtCtrls;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    Image1: TImage;
    procedure Timer1Timer(Sender: TObject);
  private
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  Image1.Free;
  Timer1.Free
end;

end.
```

Помітимо, що якби нам треба було, щоб заставка «мигала», тобто періодично з'являлася і зникала, ми могли б створити наступний оброблювач події OnTimer:

```
procedure TForm1.Timer1Timer(Sender: TObject);
begin
  if Image1.Visible = true then Image1.Hide
        else Image1.Show;
end;
```

Приклад 2.4.

Створимо застосування, що демонструє обертання Місяця навколо Землі.

Рішення

1. Створимо нове застосування.
2. Зі сторінки System помістимо на Form1 компонент Timer під ім'ям Timer1. Його властивість Interval установимо рівним 55. Через кожні 55 мілісекунд буде збуджуватися подія OnTimer, яку ми будемо використовувати для переміщення по формі компонента Shape2. Число 55 – це мінімальне значення, називане тиком, яке можна установити як значення властивості Interval. У загальному випадку будь-яке значення властивості Interval буде округлено у більшій бік до числа, кратного 55. Це зв'язано з особливостями апаратного таймера комп'ютера.
3. Зі сторінки Additional помістимо на Form1 компонент Shape під ім'ям Shape1. Установимо наступні значення для його властивостей:

Shape – stCircle,
Height – 121,
Width – 121,
Left – 240,
Top – 104.

Виберемо властивість Brush і виконаємо по ньому подвійний щиглик мишею. В Інспекторі Об'єктів додатково з'явилися дві властивості: Color і Style. Виберемо властивість Color і установимо її рівною clBlue.

4. Помістимо на Form1 компонент Shape під ім'ям Shape2. Установимо наступні значення для його властивостей:

Shape – stCircle,
Height – 41,

PDF-XChange Click to buy NOW! www.docu-track.com

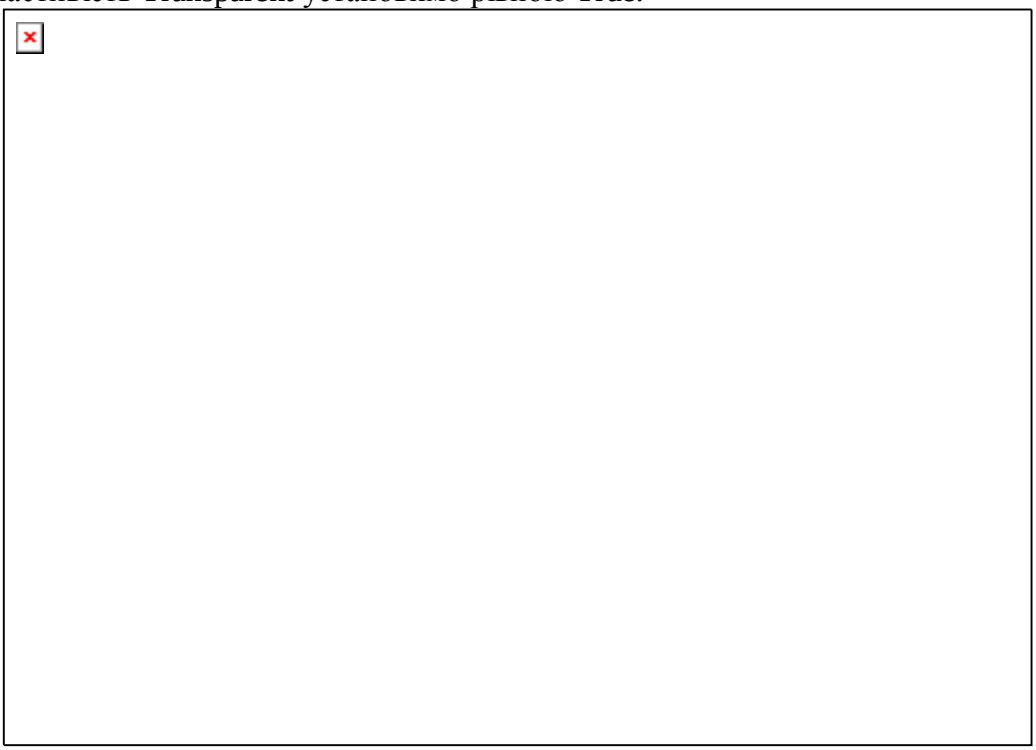
Width – 41,
Left – 400,
Top – 152.

У властивості Brush установимо колір кисті clYellow.

5. У верхній частині Form1 розмістимо компонент Label під ім'ям Label1. Його властивості Caption задамо значення – 'Обертання Місяця навколо Землі'. Ввійдемо у властивість Font і установимо :

Шрифт – Courier New,
Накреслення – напівжирний,
Розмір – 16,
Набір символів – кирилиця.

Властивість Transparent установимо рівною True.



Мал. 2.19. Розташування на формі компонентів Timer, Label і двох компонентів Shape

6. Активізуємо подвійним щигликом компонент Timer1. Внесемо виправлення в текст модуля, починаючи з розділу var.

Текст модуля Unit2.pas.

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, ExtCtrls;

type
  TForm1 = class(TForm)
    Timer1: TTimer;
    Shape1: TShape;
    Shape2: TShape;
    Label1: TLabel;
    procedure Timer1Timer(Sender: TObject);
```

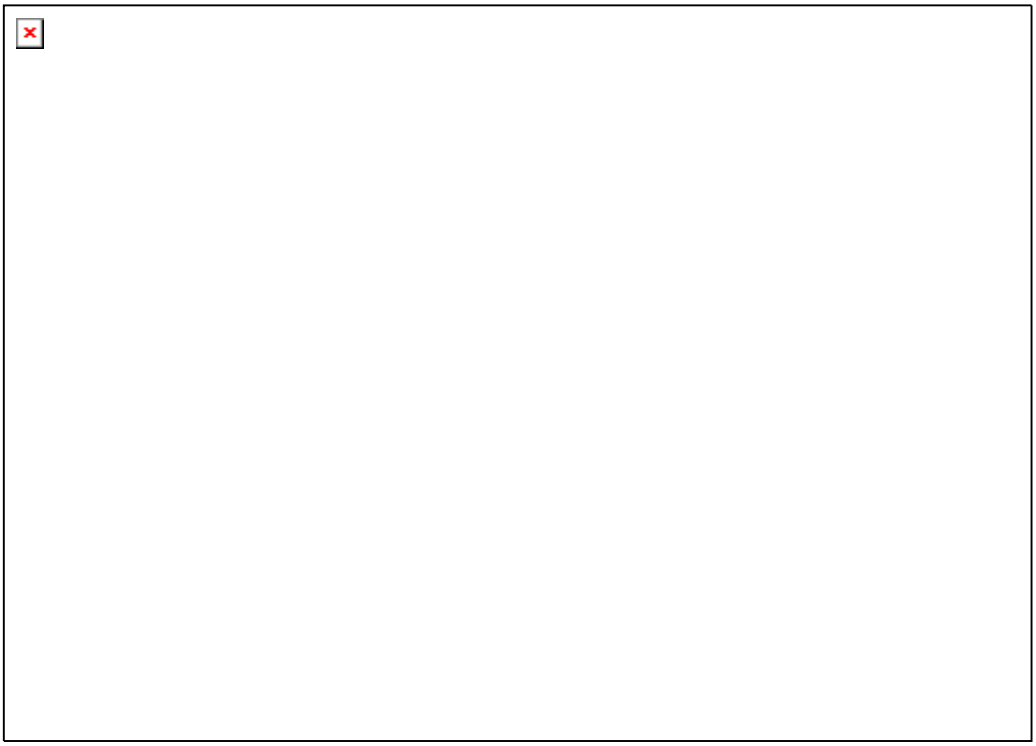
PDF-XChange
Click to buy NOW!
www.docu-track.com

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  x:real;
implementation

{$R *.DFM}

procedure TForm1.Timer1Timer(Sender: TObject);
begin
  x := x+0.1;
  Shape2.Left := 265+trunc(150*cos(x));
  Shape2.Top := 150-trunc(150*sin(x))
end;

initialization
  x := 0
end.
```

Додані програмістом рядка виділені напівжирним шрифтом.

Помітимо, що оброблювач події OnTimer призначений для збільшення поточного значення змінної x і, відповідно, для зміни поточного положення компонента Shape2 на формі. Це означає, що змінна x повинна бути глобальною стосовно процедури TForm1.Timer1Timer і її варто описати в розділі var секції interface. Початкове значення перемінної x може бути задане в секції initialization.

7. Виконаємо команду головного меню File | Save All і збережемо проект під ім'ям Project2, модуль під ім'ям Unit2 у папці D:\MyProject\Flight. Запустимо проект на виконання (мал. 2.20).



Мал. 2.20. Форма застосування на етапі виконання

Приклад 2.5.

Створимо зображення за допомогою компонента PaintBox.

Рішення

Компонент PaintBox надає в наше розпорядження вікно, у якому ми можемо створити довільне зображення. Програмний код, що створює зображення, повинний розташовуватися в оброблювачі події OnPaint.

Намалюємо кругову діаграму, що містить чотири сектори ясно-зелений, білий, небесний і жовтий кольори. Для кожного сектора виведемо текст, що містить розмір сектора у відсотках.

1.Створимо нове застосування.

2.Помістимо на Form1 компонент PaintBox під ім'ям PaintBox1. Установимо його розміри: Height=200; Width=200.

3.В Інспекторі Об'єктів перейдемо на сторінку Events і виконаємо подвійний щиглик по правому полю в рядку, що містить ім'я події OnPaint. У модулі Unit3.pas сформуємо обраблювач події OnPaint.

Текст модуля Unit3.pas.

```
unit Unit3;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls;

type
  TForm1 = class(TForm)
    PaintBox1: TPaintBox;
    procedure PaintBox1Paint(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.PaintBox1Paint(Sender: TObject);

procedure sector(clr:TColor;angle1,angle2:real;msg:string);
var x1,x2,y1,y2:integer;
begin
  PaintBox1.Canvas.Brush.Color := clr;
  x1 := trunc(cos(angle1)*PaintBox1.Width/2+PaintBox1.Width/2);
  y1 := PaintBox1.Height - trunc(sin(angle1)*PaintBox1.Height/2+PaintBox1.Height/2);
  x2 := trunc(cos(angle2)*PaintBox1.Width/2+PaintBox1.Width/2);
```

```
  y2 := PaintBox1.Height - trunc(sin(angle2)*PaintBox1.Height/2+PaintBox1.Height/2);
  PaintBox1.Canvas.Pie(0,0,PaintBox1.Width,PaintBox1.Height,x1,y1,x2,y2);
  PaintBox1.Font.Name := 'Arial';
  PaintBox1.Font.Size := 8;
  PaintBox1.Font.Color:= clBlack;
  PaintBox1.Font.Style := [fsBold];
  PaintBox1.Canvas.TextOut(trunc((x1+x2)/2)-20,trunc((y1+y2)/2),msg);
end;
begin
  sector(clLime,0,pi/3,'16,5%');
  sector(clWhite,pi/3,5*pi/6,'25%');
  sector(clSkyBlue,5*pi/6,7*pi/5,'28,4%');
  sector(clYellow,7*pi/5,2*pi,'30%');
end;

end.
```

Напівжирним шрифтом виділені рядки, уставлені програмістом.

Для малювання сектора кола в оброблювачі TForm1.PaintBox1Paint визначена процедура sector:

procedure sector(clr:TColor;angle1,angle2:real;msg:string);

що має наступні параметри:

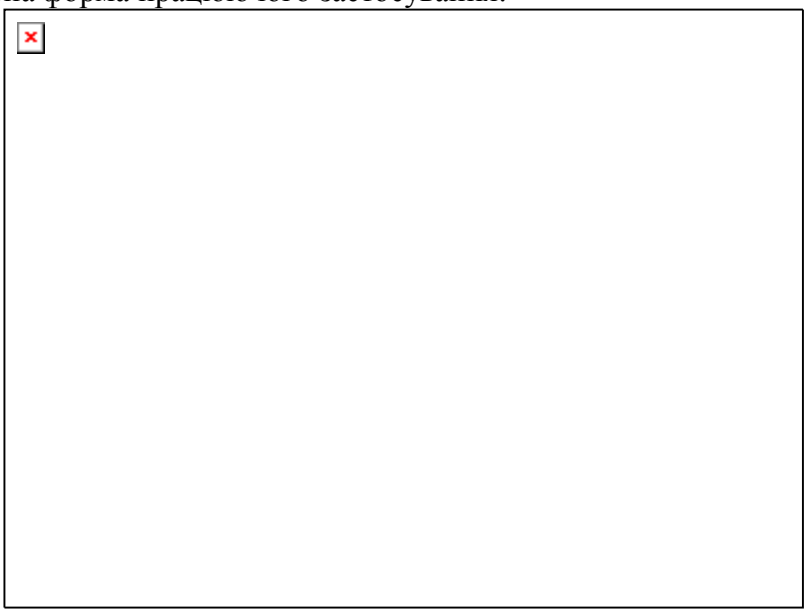
clr – колір, яким зафарбований сектор;

angle1 – початковий кут, тобто кут між віссю $\overrightarrow{OX}$ і правою стороною сектора (задається в радіанах);

angle2 – кінцевий кут, тобто кут між віссю $\overrightarrow{OX}$ і лівою стороною сектора (задається в радіанах);

msg – текст, використовуваний для підпису сектора.

4. Виконаємо команду головного меню File|Save All і збережемо проект під ім'ям Project3, модуль під ім'ям Unit3 у папці D:\MyProject\Diagram. Запустимо проект на виконання. На мал. 2.21 зображена форма працюючого застосування.



Мал. 2.21. Використання компонента PaintBox

Для скорочення запису при звертанні до властивостей і полів об'єктів можна використовувати оператор приєднання with. Цей оператор уже розглядався нами стосовно до записів, тому досить сказати, що використання оператора with для об'єктів зовсім аналогічно.

Як приклад приведемо код оброблювача TForm1.PaintBox1Paint з використанням оператора with:

```
procedure TForm1.PaintBox1Paint(Sender: TObject);

procedure sector(clr:TColor;angle1,angle2:real;msg:string);
var x1,x2,y1,y2:integer;
begin
  with PaintBox1,Canvas do
    begin
    Brush.Color := clr;
    x1 := trunc(cos(angle1)*Width/2+Width/2);
    y1 := Height - trunc(sin(angle1)*Height/2+Height/2);
    x2 := trunc(cos(angle2)*Width/2+Width/2);
    y2 := Height - trunc(sin(angle2)*Height/2+Height/2);
    Pie(0,0,Width,Height,x1,y1,x2,y2);
    Font.Name := 'Arial';
    Font.Size := 8;
    Font.Color:= clBlack;
    Font.Style := [fsBold];
    TextOut(trunc((x1+x2)/2)-20,trunc((y1+y2)/2),msg);
    end;
end;
begin
  sector(clLime,0,pi/3,'16,5%');
  sector(clWhite,pi/3,5*pi/6,'25%');
  sector(clSkyBlue,5*pi/6,7*pi/5,'28,4%');
  sector(clYellow,7*pi/5,2*pi,'30%');
end;
```

2.6. Панель перемикачів Radio Group і список вимикачів Check List Box

2.6.1. Панель перемикачів Radio Group

Ієрархія:

TObject – TPersistent – TComponent – TControl – TWinControl – TCustomControl – TCustomGroupBox – TCustomRadioGroup.
Сторінка Палітри Компонентів: Standard.

Панель перемикачів є екземпляром класу TRadioGroup і призначена для вибору одного з декількох варіантів. Панель перемикачів є більш загальним випадком у порівнянні з перемикачем RadioButton  – екземпляром класу TRadioButton, що може знаходитися в одному з двох станів: натиснутий – не натиснутий. Панель перемикачів дозволяє створити групу перемикачів і визначити, який з них натиснутий.

Клас TRadioGroup є безпосереднім нащадком класу TCustomRadioGroup, у якому визначені основні особливості панелі перемикачів RadioGroup. Приведемо опис основних властивостей панелі перемикачів.

```
property Columns: Integer;
```

Визначає число стовпчиків, у которые будуть міститися перемикачі. За замовчуванням число стовпчиків дорівнює одиниці.

property ItemIndex: Integer;
Визначає порядковий номер виділеного перемикача. Нумерація починається з нуля. Якщо жоден з перемикачів не виділений, властивість має значення -1.

property Items: TStrings;
Містить список назв перемикачів.

Панель перемикачів є нащадком класу TWinControl і, отже, входить у сімейство віконних елементів керування, що обробляють усі події, які виникають при використанні клавіатури і миші. Подією за замовчуванням для панелі перемикачів є OnClick, що виникає при виділенні нового перемикача за допомогою клавіатури чи миші.

2.6.2. Список вимикачів Check List Box

Ієрархія:

TObject – TPersistent – TComponent – TControl – TWinControl – TCustomListBox.
Сторінка Палітри Компонентів: Additional.

Список вимикачів Check List Box подібний до панелі перемикачів Radio Group у тому розумінні, що ці обидва компоненти призначені для групування більш простих елементів керування: Radio Group поєднує залежні перемикачі, а Check List Box - незалежні вимикачі.

Окремо узятий вимикач (прапорець) являє собою компонент Check Box  , що є екземпляром класу TCheckBox. Він може знаходитися в одному з двох (рідше – трьох) станах:

включений – у вимикач міститься символ «галочка»;
виключений – вимикач порожній;
нейтральне – у вимикач міститься символ «галочка» сірого кольору.

Якщо в панелі перемикачів Radio Group обраним (натиснутим) може бути тільки один перемикач, то в списку вимикачів Check List Box кожен вимикач може знаходитися в одному з трьох станів.

Розглянемо основні властивості, визначені в класі TCheckListBox, екземпляром якого є список вимикачів.

property AllowGrayed: Boolean;
Дозволяє чи забороняє використовувати в перемикачах третій стан cbGrayed (нейтральне).

property Checked[Index: Integer]: Boolean;
Містить стан вимикача з індексом Index. Індексація починається з нуля. Якщо i-ий вимикач включений, то Checked[i] має значення True, в інших випадках – False.

property Flat: Boolean;
Визначає, чи має вимикач 3D-бордюр, що додає вимикачу виступаючий чи утоплений вид.

property ItemEnabled[Index: Integer]: Boolean;
Визначає приступність чи неприступність кожного вимикача в списку вимикачів.

type TCheckBoxState = (cbUnchecked, cbChecked, cbGrayed);

property State[Index: Integer]: TCheckBoxState;
Містить стан вимикача з індексом Index :

cbUnchecked – виключений;
cbChecked – включений;
cbGrayed – нейтральне.

Відзначимо також властивість Items, успадковану від класу TCustomListBox:

property Items: TStrings;
Містить список назв вимикачів.

У класі TCheckListBox визначена подія OnClickCheck:

property OnClickCheck: TNotifyEvent;
Настає при зміні стану будь-якого вимикача.

Подією за замовчуванням є подія OnClick .

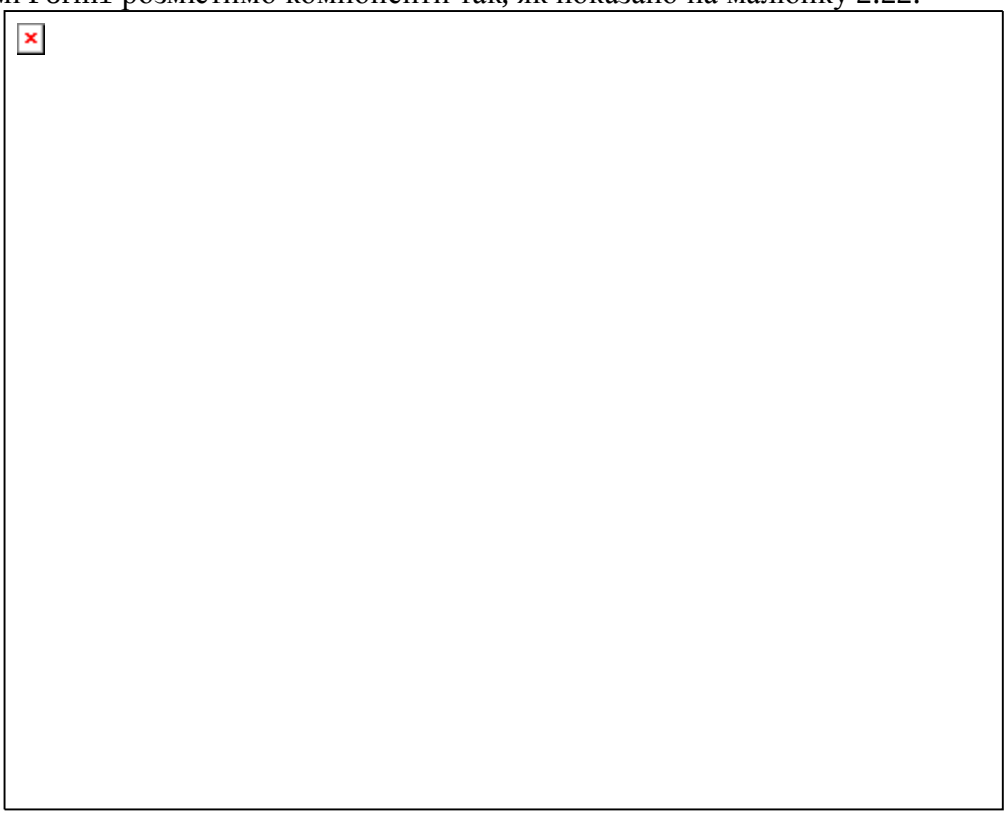
2.6.3. Приклад використання компонентів Radio Group і Check List Box
Приклад 2.6.

Створимо застосування, що дозволяє змінювати характеристики тексту, набраного в рядку введення Edit.

Рішення

Для зміни накреслення, розміру і кольору шрифту будемо використовувати панелі перемикачів, а для зміни атрибутів шрифту (закреслений, підкреслений) будемо використовувати список вимикачів.

1. Створимо нову папку, наприклад D:\MyProject\RadioCheck.
2. Відкриємо нове застосування за допомогою команди головного меню File|New|Application.
3. На формі Form1 розмістимо компоненти так, як показано на малюнку 2.22.



Мал. 2.22. Розміщення компонентів Radio Group, Check List Box, Label і Edit на формі

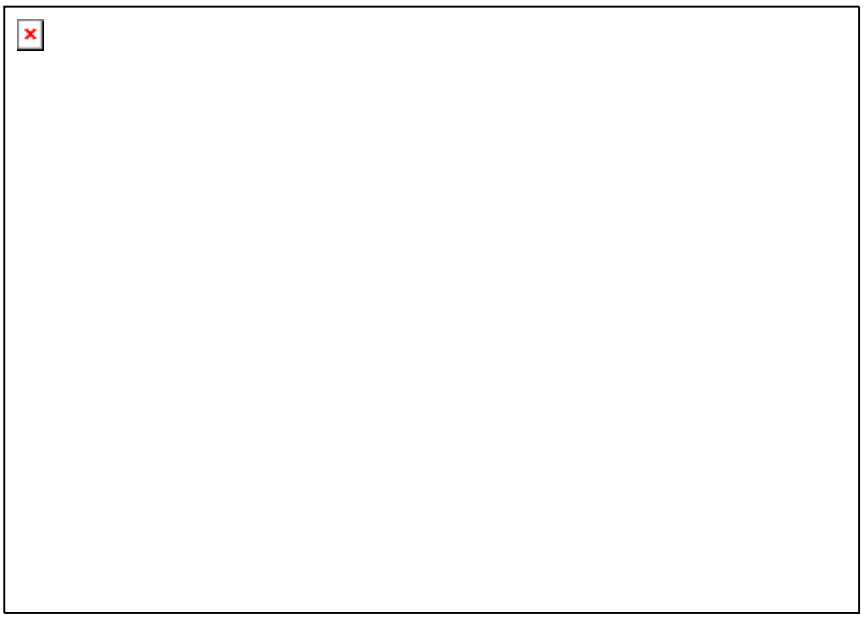
Компонент CheckListBox1 беремо зі сторінки Additional Палітри Компонентів, інші – зі сторінки Standard.

4. Властивостям Caption компонентів RadioGroup1, RadioGroup2 і RadioGroup3 задамо значення 'Накреслення', 'Розмір' і 'Колір' відповідно.

5. Ввійдемо у властивість Items для кожного зазначеного вище компонента і клацнемо по кнопці з трьома крапками. У вікні String list Editor (див. мал. 2.23), що з'явилося, варто ввести імена перемикачів. Для кожного перемикача виділяється один рядок.

Для компонента RadioGroup1 уведемо наступні рядки:

звичайний
курсив
напівжирний
напівжирний курсив



Мал. 2.23. Вікно String list Editor

Для компонента RadioGroup2:

8
10
12
14

Для компонента RadioGroup3:

чорний
зелений
червоний
синій

Після завершення введення натискаємо кнопку OK.

6. Для всіх трьох розглянутих компонентів установимо властивість ItemIndex рівною 0. Це означає, що на початку роботи програми виділеним перемикачем буде перший.

7. Для міток Label1 і Label2 властивість Caption установимо рівною 'Атрибути' і 'Зразок' відповідно.

8. Для компонента CheckListBox1 виберемо властивість Items і введемо назви вимикачів:

Закреслений
Підкреслений

9. Для компонента Edit1 установимо властивість Text рівною: 'АаВвБбФф'.

PDF-XChange
Click to buy NOW!
www.docu-track.com

10. Перейдемо до написання оброблювачів подій. Виконаємо подвійний щиглик на компоненті RadioGroup1. У вікні Редактора Коду, що з'явилося, введемо оператори для оброблювача події OnClick, що виникає при виділенні нового перемикача на панелі:

```
if RadioGroup1.ItemIndex = 0
  then Edit1.Font.Style := [ ];
if RadioGroup1.ItemIndex = 1
  then Edit1.Font.Style := [fsItalic];
if RadioGroup1.ItemIndex = 2
  then Edit1.Font.Style := [fsBold];
if RadioGroup1.ItemIndex = 3
  then Edit1.Font.Style := [fsItalic,fsBold];
CheckListBox1ClickCheck(Self);
```

Виклик оброблювача CheckListBox1ClickCheck, що буде створений трохи пізніше, дозволить нам врахувати характеристики стилю шрифту, що задаються за допомогою списку вимикачів CheckListBox1.

11. Для компонента RadioGroup2 уведемо наступні операторы в оброблювач події OnClick:

```
if RadioGroup2.ItemIndex = 0
  then Edit1.Font.Size := 8;
if RadioGroup2.ItemIndex = 1
  then Edit1.Font.Size := 10;
if RadioGroup2.ItemIndex = 2
  then Edit1.Font.Size := 12;
if RadioGroup2.ItemIndex = 3
  then Edit1.Font.Size := 14;
```

12. Для компонента RadioGroup3 уведемо такі операторы в оброблювач події OnClick:

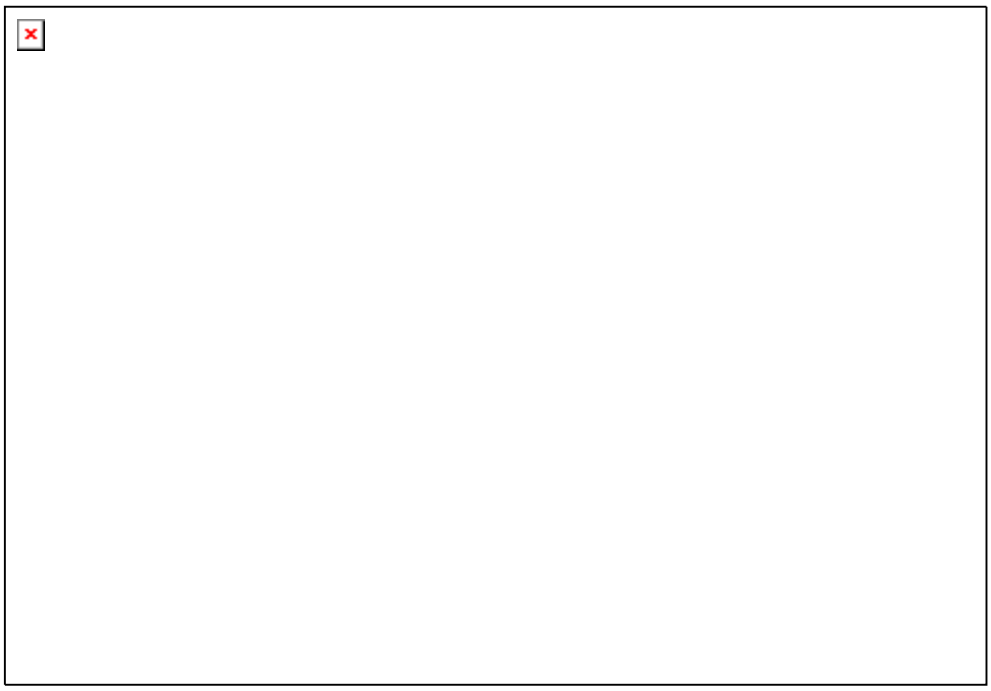
```
if RadioGroup3.ItemIndex = 0
  then Edit1.Font.Color := clBlack;
if RadioGroup3.ItemIndex = 1
  then Edit1.Font.Color := clGreen;
if RadioGroup3.ItemIndex = 2
  then Edit1.Font.Color := clRed;
if RadioGroup3.ItemIndex = 3
  then Edit1.Font.Color := clBlue;
```

13. Для компонента CheckListBox1 напишемо оброблювач події OnClickCheck, що виникає, коли змінюється стан якого-небудь вимикача:

```
if CheckListBox1.Checked[0]
  then Edit1.Font.Style := Edit1.Font.Style + [fsStrikeOut]
  else Edit1.Font.Style := Edit1.Font.Style - [fsStrikeOut];
if CheckListBox1.Checked[1]
  then Edit1.Font.Style := Edit1.Font.Style + [fsUnderline]
  else Edit1.Font.Style := Edit1.Font.Style - [fsUnderline];
```

14. Збережемо застосування, виконавши команду головного меню File|Save All. Запустимо застосування на виконання за допомогою команди Run|Run.

Виділяючи різні перемикачі чи задаючи той чи інший вимикач, будемо змінювати характеристики тексту в рядку введення (див. мал. 2.24).



Мал. 2.24. Використання компонентів Radio Group і Check List Box для зміни характеристик шрифту

Приведемо повний текст сформованого модуля.

Текст модуля Unit1.pas.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, CheckLst, ExtCtrls;

type
  TForm1 = class(TForm)
    RadioGroup1: TRadioGroup;
    RadioGroup2: TRadioGroup;
    RadioGroup3: TRadioGroup;
    CheckListBox1: TCheckListBox;
    Edit1: TEdit;
    Label1: TLabel;
    Label2: TLabel;
    procedure RadioGroup1Click(Sender: TObject);
    procedure RadioGroup2Click(Sender: TObject);
    procedure RadioGroup3Click(Sender: TObject);
    procedure CheckListBox1ClickCheck(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
implementation

{$R *.DFM}

procedure TForm1.RadioGroup1Click(Sender: TObject);
begin
  if RadioGroup1.ItemIndex = 0
    then Edit1.Font.Style := [ ];
  if RadioGroup1.ItemIndex = 1
    then Edit1.Font.Style := [fsItalic];
  if RadioGroup1.ItemIndex = 2
    then Edit1.Font.Style := [fsBold];
  if RadioGroup1.ItemIndex = 3
    then Edit1.Font.Style := [fsItalic,fsBold];
  CheckListBox1ClickCheck(Self)
end;

procedure TForm1.RadioGroup2Click(Sender: TObject);
begin
  if RadioGroup2.ItemIndex = 0
    then Edit1.Font.Size := 8;
  if RadioGroup2.ItemIndex = 1
    then Edit1.Font.Size := 10;
  if RadioGroup2.ItemIndex = 2
    then Edit1.Font.Size := 12;
  if RadioGroup2.ItemIndex = 3
    then Edit1.Font.Size := 14;
end;

procedure TForm1.RadioGroup3Click(Sender: TObject);
begin
  if RadioGroup3.ItemIndex = 0
    then Edit1.Font.Color := clBlack;
  if RadioGroup3.ItemIndex = 1
    then Edit1.Font.Color := clGreen;
  if RadioGroup3.ItemIndex = 2
    then Edit1.Font.Color := clRed;
  if RadioGroup3.ItemIndex = 3
    then Edit1.Font.Color := clBlue;
end;

procedure TForm1.CheckListBox1ClickCheck(Sender: TObject);
begin
  if CheckListBox1.Checked[0]
    then Edit1.Font.Style := Edit1.Font.Style + [fsStrikeOut]
    else Edit1.Font.Style := Edit1.Font.Style - [fsStrikeOut];
  if CheckListBox1.Checked[1]
    then Edit1.Font.Style := Edit1.Font.Style + [fsUnderline]
    else Edit1.Font.Style := Edit1.Font.Style - [fsUnderline];
end;
```

end.

2.7. Списки: List Box і Combo Box

2.7.1. Список List Box

Ієрархія:

TObject – TPersistent – TComponent –TControl – TWinControl – TCustomListControl – TCustomMultiSelectListControl – TCustomListBox.
Сторінка Палітри Компонентів: Standard.

У Delphi часто доводиться мати справу з різними списками – списками рядків текстового редактора Memo, списками імен закладок у багатосторінковому компоненті Page Control, списками назв перемикачів і вимикачів в елементах керування Radio Group і Check List Box і т.д. Крім того, існує один стандартний компонент, призначений безпосередньо для відображення списку рядків на екрані – список List Box.

Цей компонент і усі вищезгадані компоненти мають у своєму складі об'єкт-список, що є екземпляром класу TStrings і містить набір рядків з асоційованими з ними довільними об'єктами. Якщо говорити точніше, то в перерахованих вище компонентах використовується не сам абстрактний клас TStrings, розглянутий у пункті 2.4.5., а його однойменні нащадки з перевизначеними методами.

Таким чином, компонент List Box, що є екземпляром класу TListBox, дозволяє відображати на екрані список рядків, кожен з який може бути зв'язан, наприклад, з деяким малюнком. Безпосереднім предком класу TListBox є клас TCustomListBox, що породжений від класу TWinControl. Отже, компонент List Box є віконним керуючим елементом.

Основні характеристики списку List Box закладені в класі TCustomListBox. Головними з них є наявність списку рядків класу TStrings, засобів відображення списку і можливість маніпулювання його елементами.

Розглянемо основні властивості класу TCustomListBox.

property ItemIndex: Integer;
Визначає індекс виділеного елемента в списку. Нумерація елементів починається з нуля. Якщо виділеного елемента немає, ця властивість приймає значення рівне -1. Якщо в списку може бути виділено кілька елементів, вказується індекс активного виділеного елемента. Властивість доступна тільки на етапі виконання програми.

property Items: TStrings;
Задає елементи списку.

property MultiSelect: Boolean;
Визначає, чи дозволяється одночасно виділяти кілька елементів списку (якщо має значення True, те таке виділення можливо, у противному випадку виділити можна тільки один елемент).

property Selected[Index: Integer]: Boolean;
Якщо і-ий елемент списку виділений, то Selected[i] дорівнює True, у противному випадку – False. Властивість доступна тільки на етапі виконання програми.

property Sorted: Boolean;
Указує, чи належні рядки в списку автоматично сортуватися за абеткою.

У класі TCustomListBox відзначимо метод Clear:

procedure Clear;
Видаляє всі елементи списку.

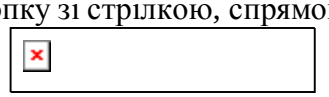
Подією за замовчуванням для списку є подія OnClick.

2.7.2. Combo Box – комбінований рядок уведення

Ієрархія:

TObject – TPersistent – TComponent – TControl – TWinControl – TCustomListControl – TCustomCombo – TCustomComboBox.
Сторінка Палітри Компонентів: Standard.

Комбінований рядок уведення (інакше – поле зі списком) Combo Box є екземпляром класу TCustomBox і поєднує в собі можливості рядка введення Edit і списку List Box. По зовнішньому вигляді компонент Combo Box нагадує звичайний рядок уведення Edit, але додатково має в правій частині кнопку зі стрілкою, спрямовану вниз.



Мал. 2.25. Комбінований рядок уведення - Combo Box

Якщо клацнути мишею по цій кнопці, з'явиться список, що випадає, подібний до списку компонента List Box. Наявність рядка введення в компоненті Combo Box розширює його можливості в порівнянні зі звичайним списком List Box. Наприклад, використовуючи рядок уведення, можна вводити в список нові елементи, здійснювати пошук потрібного елемента в списку, відображати активний елемент списку.

Основні характеристики комбінованого рядка введення закладені в класах TCustomCombo і TCustomComboBox, що є безпосередніми предками класу TCustomBox. Усі зазначені класи є нащадками базового класу усіх віконних елементів керування TWinControl.

Розглянемо деякі основні властивості класу TCustomCombo.

property DropDownCount: Integer;
Визначає максимальне число елементів, відображуване в списку, що розкривається.

property DroppedDown: Boolean;
Указує, чи відображається список, що розкривається, у даний момент. Властивість доступна тільки на етапі виконання програми.

property ItemIndex: Integer;
Визначає індекс виділеного елемента в списку. Нумерація елементів починається з нуля. Якщо виділеного елемента немає, воно приймає значення рівне -1.

property Items: TStrings;
Так само, як і в класі TCustomListBox, задає елементи списку.

Відзначимо властивість Style, визначену в класі TCustomComboBox:

type TComboBoxStyle = (csDropDown, csSimple, csDropDownList, csOwnerDrawFixed, csOwnerDrawVariable).
property Style: TComboBoxStyle;
Визначає стиль зображення списку:

csDropDown – рядки списку мають однакову фіксовану висоту, яка збігається з вікном редагування, що дозволяє вводити або редагувати текст;
csSimple – список завжди розкритий;
csDropDownList – рядка списку мають однакову фіксовану висоту, вікно редагування відсутне;
csOwnerDrawFixed – рядки списку мають висоту, обумовлену властивістю ItemHeight, і не можуть редагуватися в рядку введення;
csOwnerDrawVariable – рядки списку мають висоту, обумовлену в оброблювачі події OnMeasureItem, що виникає при необхідності одержання висоти елемента перед його перемальовуванням, і не можуть редагуватися в рядку введення.

type TCaption = string;
property Text: TCaption;
Містить текст обраного чи введеного користувачем рядка. Властивість успадкована від класу TControl.

Оскільки в класі TComboBox у порівнянні з класом TListBox відсутня властивість MultiSelect, то в компоненті Combo Box не дозволяється множинний вибір.

Основні операції для обробки списку в компоненті Combo Box – додавання, видалення, пошук, сортування елементів – здійснюється так само, як і в списку List Box.

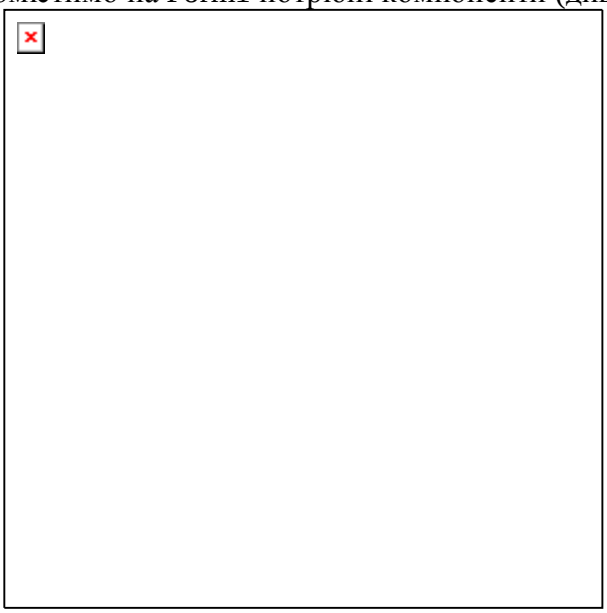
Подією за замовчуванням для комбінованого рядка введення Combo Box є подія OnChange, що виникає при зміні тексту у вікні редагування в результаті прямого редагування тексту чи в результаті вибору зі списку.

2.7.3. Приклади використання компонентів List Box і Combo Box
Приклад 2.7.

У цілочисленому масиві A(10) знайти максимальний і мінімальний елементи і поміняти їх місцями.

Рішення

1. Для нового проекту створимо нову папку, наприклад, D:\MyProject\LBOX.
2. Відкриємо новий проект, використовуючи команду головного меню File | New | Application.
3. Зі сторінки Standard помістимо на Form1 потрібні компоненти (див. мал. 2.26).



Мал. 2.26. Розташування компонентів Edit, Button і List Box на формі

PDF-XChange Click to buy NOW! www.docu-track.com

4. За допомогою Інспектора Об'єктів задамо як значення для властивості Text компонента Edit1 порожній рядок. Установимо для лівої кнопки значення властивості Caption рівною 'Уведення', а для правої – 'Рішення'.

5. У розділі interface вставимо опис масиву a і використовуваних змінних:

```
a:array[1..10] of integer;
i, min,max,imax,imin:integer;
```

6. Виконаємо подвійний щиглик по формі Form1, у результаті чого буде створена заготовка події OnCreate для форми, що виникає в момент створення форми. Розміщення операторів у цьому оброблювачі в багатьох випадках еквівалентно розміщенню операторів у секції initialization модуля. Помістимо там наступні рядки:

```
i:=0;
ListBox1.Clear
ListBox2.Clear;
```

7. Активізуємо подвійним щигликом кнопку «Уведення» і в заготовку оброблювача події OnClick помістимо наступний код:

```
ListBox1.Items.Add(Edit1.Text);
i := i+1;
a[i] := StrToInt(Edit1.Text);
Edit1.SetFocus
```

8. Активізуємо кнопку «Рішення» і в заготовку оброблювача події помістимо наступний текст:

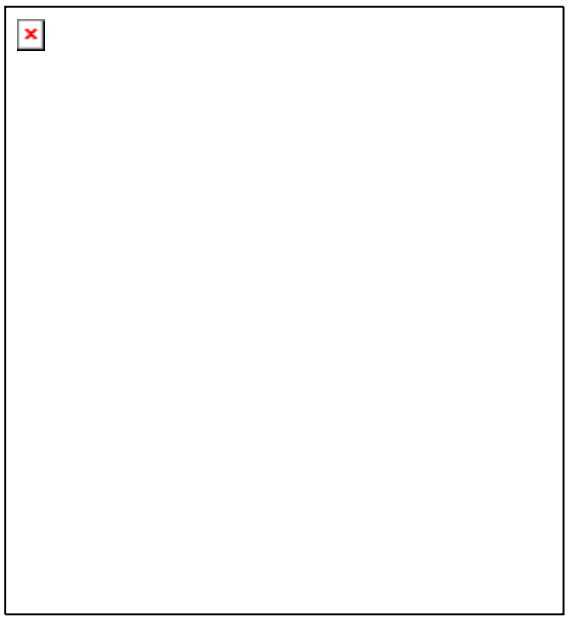
```
max := a[1];
imax := 1;
min := a[1];
imin := 1;
for k := 2 to 10 do
  begin
  if max < a[k] then
    begin
    max := a[k];
    imax := k
    end;
  if min > a[k] then
    begin
    min := a[k];
    imin := k
    end;
  end;
a[imax] := min;
a[imin] := max;
for k := 1 to 10 do
  ListBox2.Items.Add(IntToStr(a[k]));
```

9. Для зручності роботи з застосуванням створимо обраблювач події OnKeyDown для рядка введення Edit1, у якому при натисканні на клавішу Enter фокус уведення буде переводитися на кнопку «Уведення»:

```
if key = 13 then Button1.SetFocus
```

10. Збережемо проект за допомогою команди головного меню File|Save All.

11. Запустимо програму за допомогою клавіші F9. Для введення цілих чисел використовуємо рядок уведення Edit. Уведення кожного числа завершуємо натисканням кнопки Enter або щигликом миші по кнопці «Уведення». Уведені числа відображаються в першому списку. Після натискання на клавішу «Рішення» одержимо результат у другому списку (див. мал. 2.27).



Мал. 2.27. Перестановка місцями максимального і мінімального елементів з використанням компонентів List Box

Приведемо повний текст сформованого модуля.

Текст модуля Unit1.pas.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Edit1: TEdit;
    ListBox1: TListBox;
    ListBox2: TListBox;
    Button1: TButton;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Edit1KeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  a:array[1..10] of integer;
  i, min,max,imax,imin:integer;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items.Add(Edit1.Text);
  i := i+1;
  a[i] := StrToInt(Edit1.Text);
  Edit1.SetFocus
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  i:=0;
  ListBox1.Clear;
  ListBox2.Clear;
end;

procedure TForm1.Button2Click(Sender: TObject);
var k:integer;
begin
  max := a[1];
  imax := 1;
  min := a[1];
  imin := 1;
  for k := 2 to 10 do
    begin
    if max < a[k] then
      begin
      max := a[k];
      imax := k
      end;
    if min > a[k] then
      begin
      min := a[k];
      imin := k
      end;
    end;
  a[imax] := min;
  a[imin] := max;
  for k := 1 to 10 do
    ListBox2.Items.Add(IntToStr(a[k]));
end;

procedure TForm1.Edit1KeyDown(Sender: TObject; var Key: Word;
  Shift: TShiftState);
begin
  if key = 13 then Button1.SetFocus
end;

end.
```

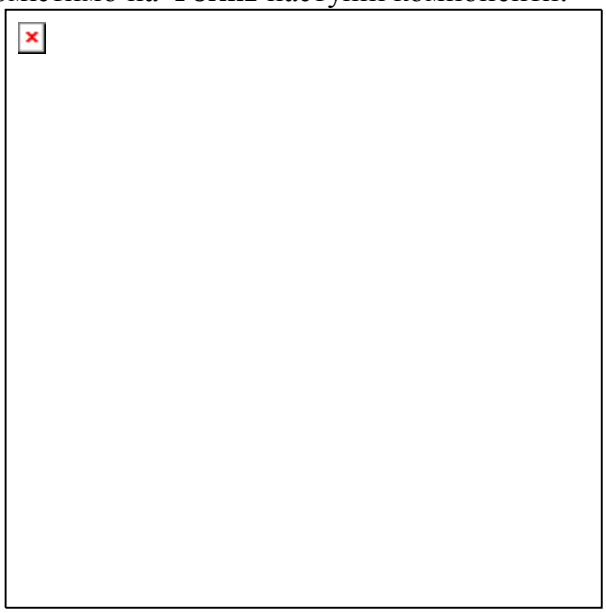
PDF-XChange
Click to buy NOW!
www.docu-track.com

Приклад 2.8.

Виконаємо завдання з Приклада 2.7 з використанням компонента Combo Box.

Рішення

1. Для нового проекту створимо нову папку, наприклад D:\MyProject\CBOX.
2. Відкриємо новий проект.
3. Зі сторінки Standard помістимо на Form1 наступні компоненти:



Мал. 2.28. Розміщення на формі компонентів Button і Combo Box

4. Покладемо властивість DropDownCount для компонентів ComboBox1 і ComboBox2 рівною 10.

Подальша послідовність дій аналогічна приведеної в рішенні приклада 2.7. Відмінності, що з'являються при написанні оброблювачів подій, стають зрозумілими при порівнянні програмного коду з попереднього приклада і програмного коду, приведеного нижче.

Текст модуля Unit1.pas.

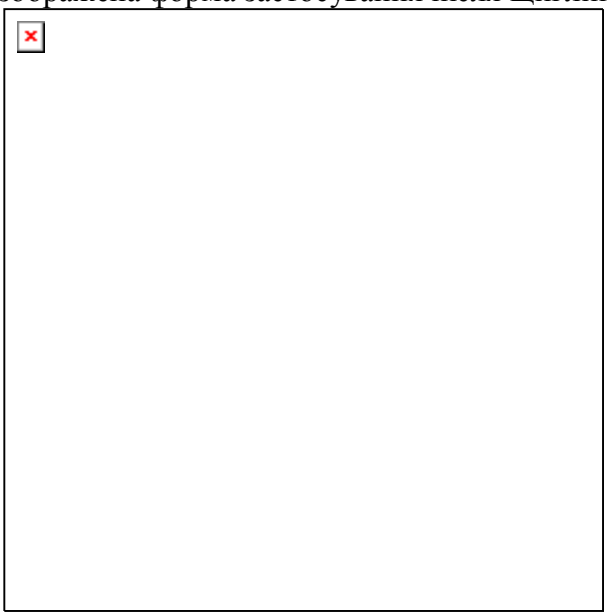
```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    ComboBox1: TComboBox;
    ComboBox2: TComboBox;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure ComboBox1KeyDown(Sender: TObject; var Key: Word;
      Shift: TShiftState);
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
  a:array[1..10] of integer;
  i, min,max,imax,imin:integer;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  ComboBox1.Items.Add(ComboBox1.Text);
  i := i+1;
  a[i] := StrToInt(ComboBox1.Text);
  ComboBox1.SetFocus;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
  i:=0;
  ComboBox1.Clear;
  ComboBox2.Clear;
  ComboBox1.TabOrder := 0;
end;

procedure TForm1.Button2Click(Sender: TObject);
var k:integer;
begin
  max := a[1];
  imax := 1;
  min := a[1];
  imin := 1;
  for k := 2 to 10 do
    begin
    if max < a[k] then
      begin
      max := a[k];
      imax := k
      end;
    if min > a[k] then
      begin
      min := a[k];
      imin := k
      end;
    end;
  a[imax] := min;
```

```
  a[imin] := max;
  for k := 1 to 10 do
    ComboBox2.Items.Add(IntToStr(a[k]));
  ComboBox2.DroppedDown := true;
end;

procedure TForm1.ComboBox1KeyDown(Sender: TObject; var Key:
Word; Shift: TShiftState);
begin
  if key =13 then Button1.SetFocus
end;

end.
```

На малюнку 2.29 зображена форма застосування після щиглика по кнопці «Рішення».



Мал. 2.29. Перестановка місцями максимального і мінімального елементів з використанням компонентів Combo Box

2.8. Таблиця String Grid

2.8.1. Основні характеристики таблиці String Grid

Ієрархія:

TObject – TPersistent – TComponent – TControl – TWinControl – TCustomControl – TCustomGrid – TCustomDrawGrid – TDrawGrid.
Сторінка Палітри Компонентів: Additional.

Таблиця String Grid є екземпляром класу TStringGrid і призначена для відображення двовимірної інформації, наприклад елементів матриці. У таблиці може знаходитися довільна кількість рядків і стовпців. Якщо зафіксувати необхідну кількість перших рядків і стовпців, то можна задати заголовки рядків і стовпців, що є постійно присутніми у вікні компонента, у тому числі при горизонтальному і вертикальному скролінгу.

На перетинанні рядків і стовпців знаходяться комірки. Кожна комірка може містити символьний рядок і довільний об'єкт, асоційований з коміркою. Найчастіше таким об'єктом є

PDF-XChange
Click to buy NOW!
www.docu-track.com

деякий малюнок. Якщо для комірок заданий режим редагування, то на етапі виконання програми дозволяється вводити і редагувати дані, що знаходяться в комірці.

Нумерація рядків і стовпців таблиці починається з нуля. Координати кожної комірки таблиці задаються парою чисел, перше з яких є номером стовпця, а друге – номером рядка. Наприклад, комірка з координатами (3,5) розташована в четвертому стовпці і шостому рядку.

Клас TCustomGrid, що є предком класу TStringGrid, містить визначення багатьох характеристик, загальних для будь-яких таблиць. Багато які з них визначають зовнішній вигляд таблиці. Розглянемо основні властивості класу TCustomGrid.

property Col: Longint;
Задає стовпець, у якому знаходиться активна комірка. Властивість доступна тільки на етапі виконання програми.

property ColCount: Longint;
Задає число стовпців у таблиці.

property ColWidths[Index: Longint]: Integer;
Задає ширину кожного стовпця в таблиці. Доступна тільки на етапі виконання програми.

property DefaultColWidth: Integer;
Задає вихідну ширину всіх стовпців. Для завдання ширини окремого стовпця варто використовувати властивість ColWidths.

property DefaultDrawing: Boolean;
Якщо властивість має значення True, то промальовування комірок при малюванні таблиці буде відбуватися автоматично, у противному випадку необхідно створити свої засоби відображення.

property DefaultRowHeight: Integer;
Задає вихідну висоту всіх рядків. Для завдання висоти окремого рядка варто використовувати властивість RowHeights.

property FixedColor: TColor;
Задає колір фіксованих комірок.

property FixedCols: Integer;
Задає число фіксованих стовпців. За замовчуванням задається один фіксований стовпець.

property FixedRows: Integer;
Задає число фіксованих рядків. За замовчуванням задається один фіксований рядок.

property GridLineWidth: Integer;
Задає товщину ліній між комірками в пікселях.

property Row: Longint;
Задає рядок, у якому знаходиться активна комірка. Доступно тільки на етапі виконання програми.

property RowCount: Longint;
Задає число рядків таблиці.

property RowHeights[Index: Longint]: Integer;

Задає висоту кожного рядка таблиці. Доступна тільки на етапі виконання програми.

```
type TGridCoord = record
    X: Longint;
    Y: Longint;
  end;
type TGridRect = record
    case Integer of
      0: (Left, Top, Right, Bottom: Longint);
      1: (TopLeft, BottomRight: TGridCoord);
  end;
property Selection: TGridRect;
```

Вказує область поточного виділення. При виділенні вказуються номери рядків і стовпців. Доступна тільки на етапі виконання програми.

Для визначення поводження таблиці використовується властивість Options. Перш ніж привести опис цієї властивості, помітимо, що кожна комірка таблиці може знаходиться в одному з п'яти станів: пасивному, виділеному (виділена особливим кольором), активному (виділена рамкою з крапок), фіксованому (теж виділена особливим кольором) і в стані редагування (значення прапора goEditing властивості Options дорівнює True).

```
type
  TGridOption = (goFixedVertLine, goFixedHorzLine, goVertLine, goHorzLine, goRangeSelect, goDrawFocusSelected, goRowSizing, goColSizing, goRowMoving, goColMoving, goEditing, goTabs, goRowSelect, goAlwaysShowEditor, goThumbTracking);
TGridOptions = set of TGridOption;
property Options: TGridOptions;
```

Задає прапори, що визначають поводження таблиці. Прапори означають наступне:

goFixedVertLine – фіксовані комірки розділяються вертикальними лініями;

goFixedHorzLine – фіксовані комірки розділяються горизонтальними лініями;

goVertLine – інші комірки розділяються вертикальними лініями;

goHorzLine – інші комірки розділяються горизонтальними лініями;

goRangeSelect – припустиме виділення декількох комірок;

goDrawFocusSelected – активна комірка зафарбовується тим же кольором, яким зафарбовується і виділена, у противному випадку зафарбовується кольором нейтральних комірок;

goRowSizing – висота рядків таблиці може змінюватися ;

goColSizing – ширина стовпців таблиці може змінюватися;

goRowMoving – рядки таблиці можуть переміщатися ;

goColMoving – стовпці таблиці можуть переміщатися ;

goEditing – комірки можуть редагуватися;

goTabs – перехід від комірки до комірки можливий за допомогою клавіші Tab (клавіш Shift+Tab);

goRowSelect – виділення тільки цілих рядків таблиці;

goAlwaysShowEditor – при виділенні комірки вона відразу ж стає й активною (у противному випадку активізується або клавішею F2, або подвійним натисканням клавіші миші, або натисканням якої-небудь символьної клавіші);

goThumbTracking – переміщення рухливої частини таблиці синхронно з переміщенням повзунка лінійки скролинга (у противному випадку переміщення здійснюється тільки після того, як повзунок буде відпущений).

Розглянемо тепер основні властивості класу TStringGrid, які дозволяють здійснювати доступ до даних, що знаходяться в комірках таблиці. Усі перераховані нижче властивості доступні тільки на етапі виконання програми.

property Cells[ACol, ARow: Integer]: string;
Містить двовимірний масив символьних рядків, кожна з який належить комірці, що знаходиться в стовпці ACol, і рядку ARow.

property Cols[Index: Integer]: TStrings;
Містить список рядків, що належать коміркам стовпця з індексом Index.

property Objects[ACol, ARow: Integer]: TObject;
Містить двовимірний масив, елементами якого є вказівники на об'єкти, кожний з який асоційований з відповідною йому коміркою, що знаходиться в стовпці ACol і рядку ARow.

property Rows[Index: Integer]: TStrings;
Містить список рядків, що належать коміркам рядка з індексом Index.

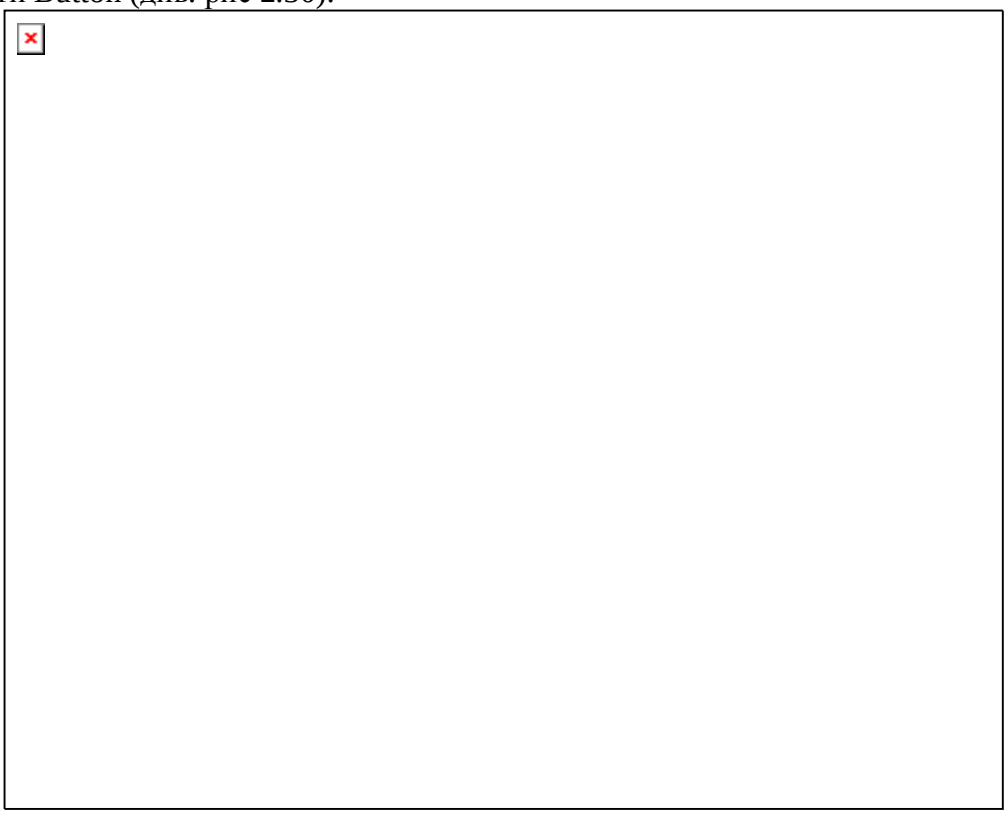
Подією за замовчуванням для таблиці StringGrid є подія OnClick.

2.8.2. Приклад використання компонента StringGrid
Приклад 2.9.

У цілочисленій матриці A(4,4) поміняти місцями перший й останній рядки.

Рішення

1. Для нового проекту створимо нову папку, наприклад D:\MyProject\StringGrid.
2. Відкриємо новий проект, використовуючи команду головного меню File|New|Application.
3. Зі сторінок Standard і Additional на Form1 помістимо два компоненти String Grid і три компоненти Button (див. рис 2.30).



Мал. 2.30. Розташування компонентів String Grid і Button на формі

PDF-XChange Click to buy NOW! www.docu-track.com

4. За допомогою Інспектора Об'єктів властивостям компонентів StringGrid1 і StringGrid2 задамо наступні значення:

FixedCols – 0,
FixedRows – 0,
ColCount – 4,
RowCount – 4.

Змінимо розміри компонентів StringGrid1 і StringGrid2 таким чином, щоб у них уміщалося 4 рядка і 4 стовпці.

5. В Інспекторі Об'єктів для обох таблиць виберемо властивість Options. Ввійдемо в нею за допомогою подвійного щиглика миші. Значення прапора goEditing зробимо рівним True.

6. Для компонентів Button1, Button2 і Button3 задамо значення властивості Caption рівною: 'Рішення 1', 'Рішення 2' і 'Очистити' відповідно.

7. Активізуємо кнопку «Рішення 1» і створимо наступний обраблювач події OnClick:

```
procedure TForm1.Button1Click(Sender: TObject);
var   i:Integer;
begin
  for  i := 0  to 3 do
    begin
    StringGrid2.Cells[i,0]:=StringGrid1.Cells[i,3];
    StringGrid2.Cells[i,3]:=StringGrid1.Cells[i,0];
    StringGrid2.Cells[i,1]:=StringGrid1.Cells[i,1];
    StringGrid2.Cells[i,2]:=StringGrid1.Cells[i,2];
    end
end;
```

Нагадаємо, що нумерація рядків і стовпців починається з 0, а у властивості Cells[j,i] перший індекс позначає номер стовпця, а другий – номер рядка.

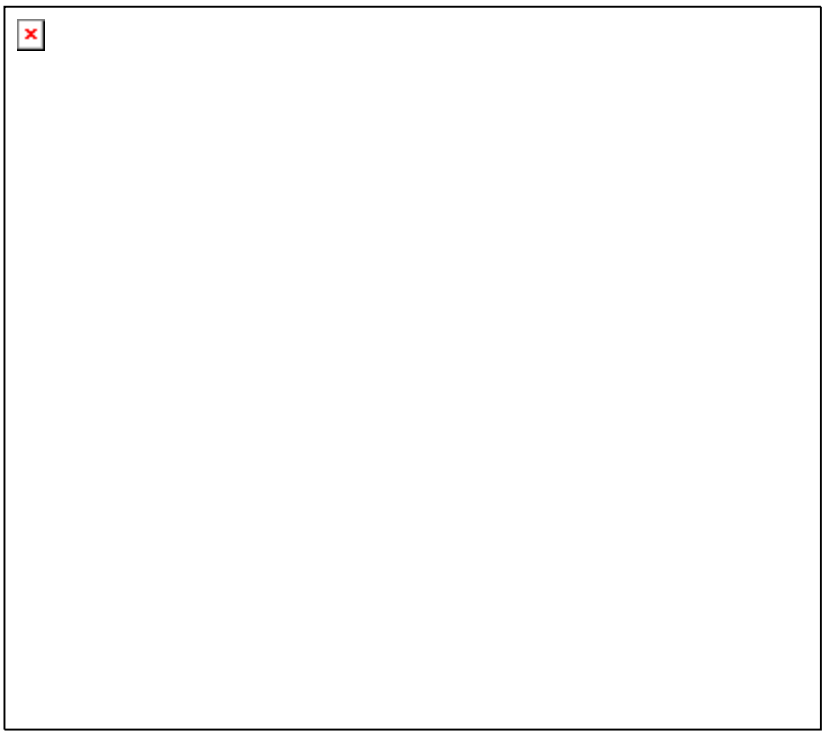
8. Вихідна задача може бути вирішена за допомогою властивості Rows. Активізуємо кнопку «Рішення 2» і в заготовці оброблювача події OnClick розмістимо наступний код:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  StringGrid2.Rows[0]:=StringGrid1.Rows[3];
  StringGrid2.Rows[1]:=StringGrid1.Rows[1];
  StringGrid2.Rows[2]:=StringGrid1.Rows[2];
  StringGrid2.Rows[3]:=StringGrid1.Rows[0];
end;
```

9. Для очищення другої таблиці призначена кнопка «Очистити». Активізуємо її й у заготовку оброблювача події помістимо наступний код:

```
procedure TForm1.Button3Click(Sender: TObject);
var   i,j:Integer;
begin
  for  i := 0  to 3 do
    for  j := 0  to 3 do
       StringGrid2.Cells[j,i] := ' '
end;
```

10. Збережемо проект за допомогою команди головного меню File|Save All і запустимо його на виконання (див. мал. 2.31).



Мал. 2.31. Використання компонентів String Grid для перестановки рядків матриці

Приведемо повний текст сформованого модуля.

Текст модуля Unit1.pas.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, Grids;

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var   i:Integer;
begin
  for  i := 0  to 3 do
    begin
    StringGrid2.Cells[i,0]:=StringGrid1.Cells[i,3];
    StringGrid2.Cells[i,3]:=StringGrid1.Cells[i,0];
    StringGrid2.Cells[i,1]:=StringGrid1.Cells[i,1];
    StringGrid2.Cells[i,2]:=StringGrid1.Cells[i,2];
    end

end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  StringGrid2.Rows[0]:=StringGrid1.Rows[3];
  StringGrid2.Rows[1]:=StringGrid1.Rows[1];
  StringGrid2.Rows[2]:=StringGrid1.Rows[2];
  StringGrid2.Rows[3]:=StringGrid1.Rows[0];
end;

procedure TForm1.Button3Click(Sender: TObject);
var   i,j:Integer;
begin
  for  i := 0  to 3 do
    for  j := 0  to 3 do
       StringGrid2.Cells[j,i] := ' '
end;

end.
```

2.9. Створення меню. Компоненти Main Menu і Popup Menu

2.9.1. Загальний опис

Більшість застосувань має головне меню, що містить перелік припустимих операцій. Пункти головного меню інакше називають ще елементами меню нульового (верхнього) рівня, оскільки з кожним з них може бути зв'язане розкривне підменю, що містить елементи меню першого рівня і т.д. Крім того, з кожним елементом меню може бути зв'язана деяка процедура, що буде виконуватися при щиглику мишею по елементі меню або при натисканні на клавішу Enter у той момент, коли елемент меню є активним.

У Delphi для створення головного меню існує невізуальний компонент Main Menu, розташований на сторінці Standard Палітри Компонентів. Процес створення головного меню на етапі конструювання форми здійснюється за допомогою убудованого конструктора меню (Menu Designer).

Кожен елемент меню являє собою об'єкт класу TMenuItem. Характеристики цього класу будуть розглянуті в пункті 2.9.3.

Крім головного меню, зв'язаного з формою застосування, у Delphi існує компонент Popup Menu, призначений для створення контекстного меню. Контекстне меню може бути створене для будь-якого віконного елемента керування. Для виклику контекстного меню необхідно помістити курсор миші на віконний елемент і натиснути праву кнопку миші. Для

встановлення зв'язку між віконним елементом і компонентом Popup Menu використовується властивість Popup Menu віконного елемента, у якому варто вказати ім'я відповідного меню.

Контекстне меню, так само, як і головне меню, створюється, як правило, за допомогою конструктора меню на етапі конструювання форми. Елементами контекстного меню також є об'єкти класу TMenuItem.

2.9.2. Main Menu – головне меню застосування

Ієрархія:

TObject – TPersistent – TComponent –TMenu.
Сторінка Палітри Компонентів: Standard.

Головне меню застосування Main Menu є екземпляром класу TMainMenu, безпосередній предок якого – клас TMenu містить характеристики, що визначають меню як єдине ціле.

Основною властивістю класу TMainMenu, успадкованою від класу TMenu, є властивість Items:

property Items: TMenuItem; default;
Містить елементи нульового рівня головного меню застосування.

Найчастіше головне меню створюється на етапі конструювання форми. Для цього необхідно спочатку помістити компонент Main Menu зі сторінки Standard Палітри Компонентів на форму, а потім викликати конструктор меню. Для виклику конструктора меню можна двічі клацнути лівою кнопкою миші по компоненті Main Menu, або клацнути по компоненті правою кнопкою миші й у контекстному меню, що появиться, вибрати команду Menu Designer, або, нарешті, можна скористатися властивістю Items в Інспекторі Об'єктів. При активізації мишею цієї властивості, так само, як і в двох попередніх випадках, відкривається конструктор меню, що дозволяє набрати елементи меню. Для завдання характеристик кожного елемента меню варто використовувати Інспектор Об'єктів, попередньо виділивши відповідний елемент структури меню. Для створення підменю можна використовувати контекстне меню конструктора меню.

2.9.3. Клас TMenuItem

Елементами як головного, так і контекстного меню є об'єкти класу TMenuItem, що є безпосереднім нащадком класу TComponent. Елемент меню може являти собою підменю, чи команду, чи розділову лінію.

Якщо елемент меню являє собою підменю, то його властивість Items повинна містити відповідні пункти цього підменю. Якщо ця властивість не містить жодного пункту, то елемент меню є або командою, або розділовою лінією.

Якщо елемент меню є розділовою лінією, то його властивість Caption повинна містити значення « - » (знак «мінус»).

В всіх інших випадках елемент меню буде командою, тобто з цим елементом меню буде зв'язаний оброблювач події OnClick, що виникає при щиглику лівою кнопкою миші по елементі меню або при натисканні клавіші Enter, коли елемент меню є активним.

Розглянемо основні властивості класу TMenuItem.

property Bitmap: TBitmap;
Властивість класового типу. Його значенням є вказівник на об'єкт типу TBitmap, що містить побітове зображення. Використовується для зв'язування з елементом меню невеликого зображення.

PDF-XChange
Click to buy NOW!
www.docu-track.com

property Caption: string;
Містить текст елемента меню. Якщо перед деяким символом тексту помістити символ &, то в такий спосіб можна задати клавішу швидкого переходу (акселератор). Клавіші-акселератори на етапі виконання програми діють у такий спосіб. Якщо клавіша-акселератор задана для елемента меню нульового рівня, то при натисканні на комбінацію Alt+акселератор відкриється відповідне розкривне меню першого рівня. Якщо клавіша-акселератор задана для елемента меню першого рівня і це меню відкрито, то при натисканні на клавішу-акселератор буде виконана відповідна елементу меню команда.

property Checked: Boolean;
Якщо властивість має значення True, то елемент меню позначається «галочкою».

property Count: Integer;
Містить число молодших елементів меню, що знаходяться у властивості Items. Властивість доступна тільки для читання.

property Default: Boolean;
Якщо властивість має значення True, то текст елемента меню виділяється напівжирним шрифтом, а подвійне натискання мишею старшого елемента приводить до появи в поточного елемента події OnClick. За замовчуванням має значення False.

property Enabled: Boolean;
Якщо властивість має значення True, то елемент меню реагує на події від миші і клавіатури. У противному випадку елемент меню не доступний і виділяється тьмяним кольором. За замовчуванням має значення True.

property Items[Index: Integer]: TMenuItem; default;
Властивість задає молодші елементи меню стосовно поточного елементу. Число елементів визначається властивістю Count. Нумерація починається з нуля. Властивість доступна тільки для читання.

type TShortCut = Low(Word) .. High(Word);
property ShortCut: TShortCut;
Визначає комбінацію «гарячих» клавіш (клавіш швидкого керування), що забезпечують швидкий вибір даного елемента меню. Це означає, що якщо на етапі виконання програми ви натиснете на комбінацію «гарячих» клавіш, те відразу ж буде виконана зв'язана з нею команда, незалежно від рівня елемента меню і його видимості в момент натискання комбінації «гарячих» клавіш.

У класі TMenuItem визначена подія OnClick:

property OnClick: TNotifyEvent;
Виникає при виборі елемента меню мишею чи при натисканні на клавішу Enter, коли елемент меню є активним. Ця ж подія є і подією, використовуваною за замовчуванням.

2.9.4. Popup Menu – контекстне меню

Ієрархія:

TObject – TPersistent – TComponent –TMenu.
Сторінка Палітри Компонентів: Standard.

PDF-XChange
Click to buy NOW!
www.docu-track.com

Контекстне меню Popup Menu є екземпляром класу TPopupMenu, що так само, як і клас TMainMenu, є нащадком класу TMenu. Розглянемо основні характеристики, що вводяться в класі TPopupMenu.

type TPopupAlignment = (paLeft, paRight, paCenter);
property Alignment: TPopupAlignment;
Визначає розташування контекстного меню щодо курсору миші:
paLeft - лівий верхній кут меню знаходиться біля курсору,
paRight - правий верхній кут меню знаходиться біля курсору,
paCenter - середина верхньої границі меню знаходиться біля курсору.
За замовчуванням має значення paLeft .

property AutoPopup: Boolean;
Якщо властивість має значення True , контекстне меню з'являється при натисканні правої клавіші миші, якщо має значення False, меню не з'являється (у цьому випадку варто використовувати метод Popup). За замовчуванням має значення True.

Метод Popup визначається в такий спосіб:

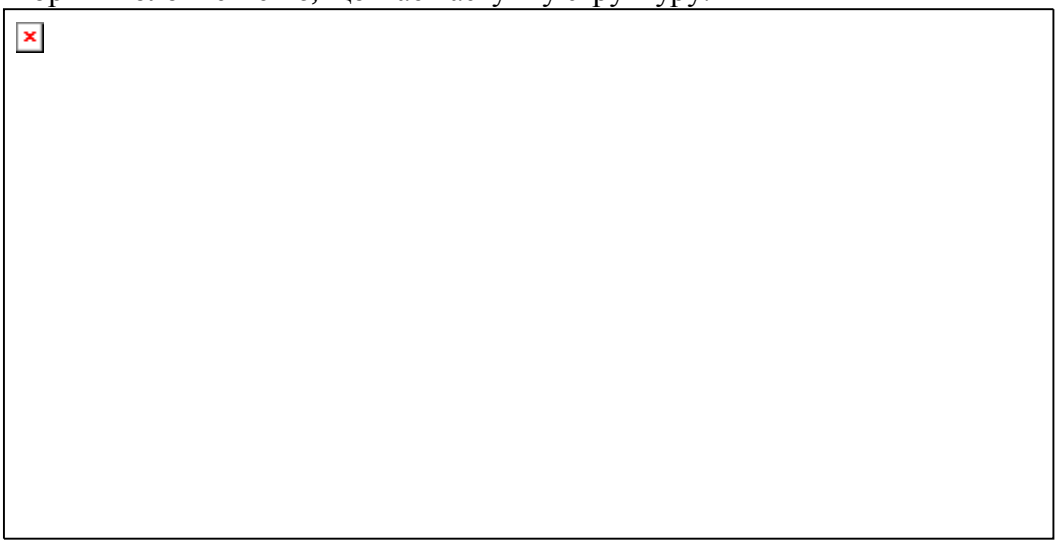
procedure Popup(X, Y: Integer); virtual;
Виводить на екран меню, при цьому координати його лівого верхнього кута рівні X и Y.

У класі TPopupMenu визначена подія OnPopup:

property OnPopup: TNotifyEvent;
Виникає при виклику контекстного меню при натисканні правої клавіші миші, якщо властивість AutoPopup має значення True, або при виклику методу Popup.

Контекстне меню створюється аналогічно головному меню застосування. Відмінність полягає в тім, що завершується створення контекстного меню прив'язкою його до одного або до декількох віконних елементів керування. При цьому використовується властивість PopupMenu цих елементів керування.

2.9.5. Приклади використання компонента Main Menu і Popup Menu
Приклад 2.10.
Створити головне меню, що має наступну структуру:



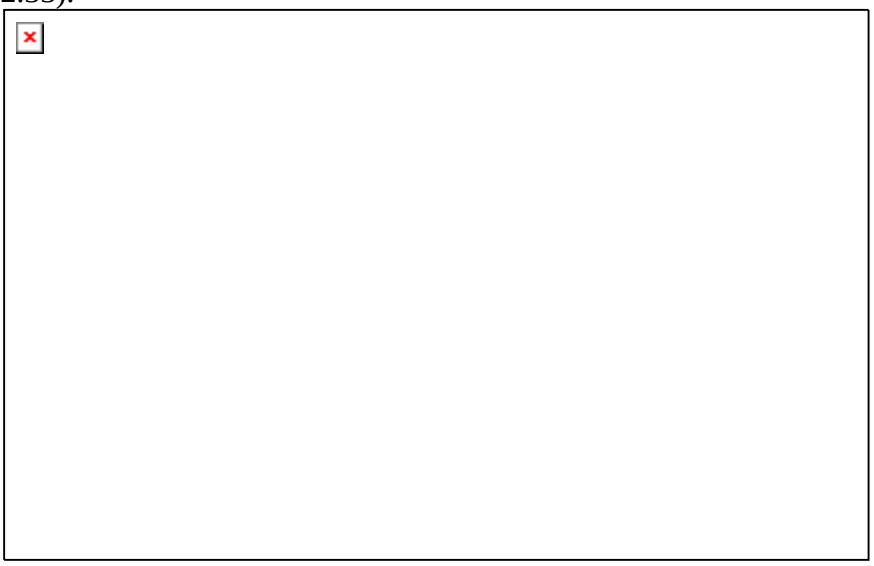
Мал. 2.32. Структура головного меню застосування

PDF-XChange Click to buy NOW! www.docu-track.com

Як реакцію на вибір пункту меню видавати повідомлення про вибір відповідного пункту.

Рішення.

1. Створимо нову папку, наприклад D:\MyProject\MAINMENU.
2. Відкриємо новий проект.
3. Зі сторінки Standard помістимо на Form1 компонент MainMenu1.
4. В Інспекторі Об'єктів виберемо властивість Items компонента MainMenu1, а потім клацнемо по кнопці з трьома крапками. Після цього на екрані з'явиться вікно конструктора меню.
5. За допомогою конструктора меню наберемо елементи меню, зазначені в умові приклада 2.10 (див. мал. 2.33).



Мал. 2.33. Вікно конструктора меню під час створення головного меню застосування

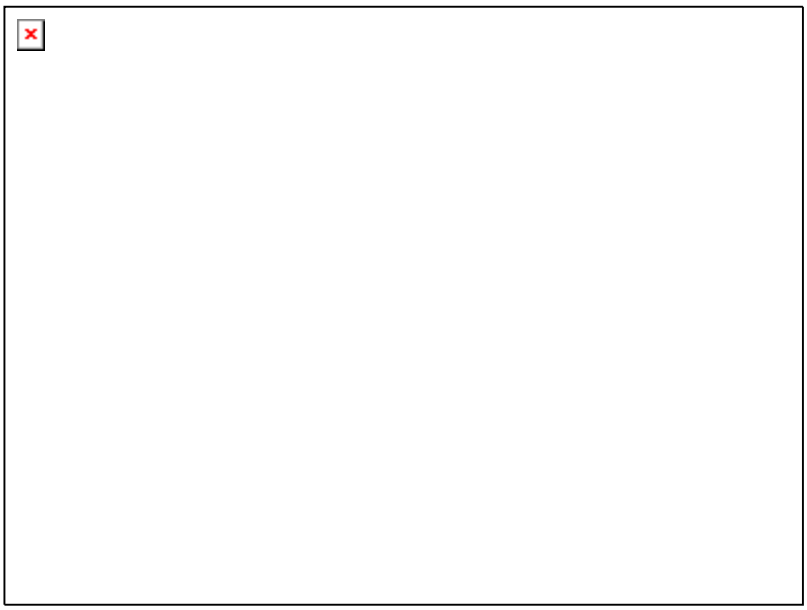
Нагадаємо, що підкреслена буква в імені пункту означає клавішу швидкого переходу. За допомогою цієї клавіші можна здійснити швидкий перехід на даний елемент меню. Задати клавішу швидкого переходу можна за допомогою символу & , що поміщається перед відповідним символом тексту. Наприклад: &File, Cu&t. Зверніть увагу, що, уводячи назву елемента меню, ви задаєте значення властивості Caption для цього елемента меню. Для того щоб увести код клавіші швидкого керування, наприклад Ctrl+S для пункту Save, необхідно для відповідного елемента меню вибрати властивість ShortCut в Інспекторі Об'єктів і в списку, що випадає, вибрати потрібне значення. Для того щоб елемент меню являв собою розділову лінію, треба його властивості Caption задати значення '-'. Для створення підменю, наприклад для пункту Reopen, варто вибрати потрібний елемент меню і клацнути правою кнопкою миші. У появившемся контекстному меню конструктора виберіть пункт Create Submenu. Далі введіть пункти підменю.

6. Якщо ви увели всі пункти меню, то закрийте вікно конструктора меню. На формі Form1 залишилося створене меню. Для визначення реакції на вибір пунктів меню треба по черзі вибирати всі пункти меню і клацати по них мишкою. У результаті з'явиться вікно Редактора Коду, у якому потрібно ввести програмний код для оброблювача події OnClick. Наприклад, для елемента меню New оброблювач події OnClick буде мати вид:

```
procedure TForm1.New1Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт New');
end;
```

Тут ShowMessage – це стандартна функція, що виводить на екран вікно з заданим повідомленням.

7. Збережіть проект і запустіть його на виконання.



Мал. 2.34. Використання компонента Main Menu

Приведемо повний текст сформованого модуля.

Текст модуля Unit1.pas.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, Menus;

type
  TForm1 = class(TForm)
    MainMenu1: TMainMenu;
    File1: TMenuItem;
    New1: TMenuItem;
    Open1: TMenuItem;
    Save1: TMenuItem;
    Close1: TMenuItem;
    N1: TMenuItem;
    Reopen1: TMenuItem;
    Project11: TMenuItem;
    Project21: TMenuItem;
    Project31: TMenuItem;
    Edit1: TMenuItem;
    Cut1: TMenuItem;
    Copy1: TMenuItem;
    Past1: TMenuItem;
    Delete1: TMenuItem;
    Run1: TMenuItem;
    Run2: TMenuItem;
    N2: TMenuItem;
    StepOver1: TMenuItem;
    TraceInto1: TMenuItem;
    procedure New1Click(Sender: TObject);
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
    procedure Open1Click(Sender: TObject);
    procedure Save1Click(Sender: TObject);
    procedure Close1Click(Sender: TObject);
    procedure Project11Click(Sender: TObject);
    procedure Project21Click(Sender: TObject);
    procedure Project31Click(Sender: TObject);
    procedure Cut1Click(Sender: TObject);
    procedure Copy1Click(Sender: TObject);
    procedure Past1Click(Sender: TObject);
    procedure Delete1Click(Sender: TObject);
    procedure Run2Click(Sender: TObject);
    procedure StepOver1Click(Sender: TObject);
    procedure TraceInto1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.New1Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт New');
end;

procedure TForm1.Open1Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт Open');
end;

procedure TForm1.Save1Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт Save');
end;

procedure TForm1.Close1Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт Close');
end;

procedure TForm1.Project11Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт Project1');
end;

procedure TForm1.Project21Click(Sender: TObject);
```

```
begin
   ShowMessage('Обраний пункт Project2');
end;

procedure TForm1.Project31Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт Project3');
end;

procedure TForm1.Cut1Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт Cut');
end;

procedure TForm1.Copy1Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт Copy');
end;

procedure TForm1.Past1Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт Past');
end;

procedure TForm1.Delete1Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт Delete');
end;

procedure TForm1.Run2Click(Sender: TObject);
begin
   ShowMessage('Обраний пункт Run');
end;

procedure TForm1.StepOver1Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт Step Over');
end;

procedure TForm1.TraceInto1Click(Sender: TObject);
begin
  ShowMessage('Обраний пункт Trace Into');
end;

end.
```

Приклад 2.11.

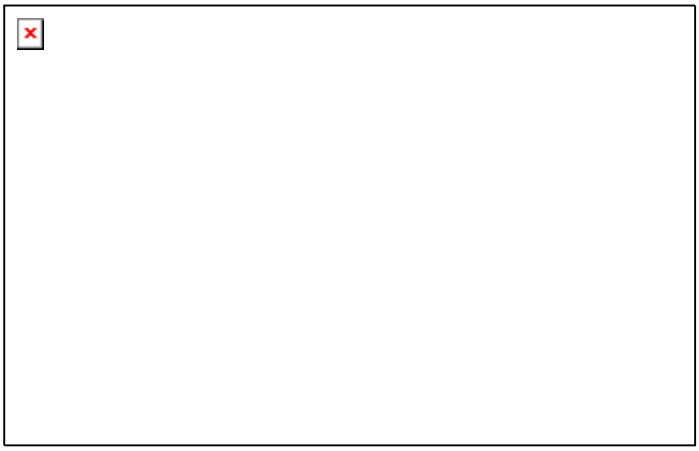
Створимо контекстне меню Popup Menu для проекту з приклада 2.9 (таблиці).

Рішення

1. Скопіюємо вміст папки D:\MyProject\StringGrid у папку D:\MyProject\PopupMenu.
2. За допомогою команди головного меню File|Open відкриємо Project1 з папки D:\MyProject\PopupMenu.

PDF-XChange
Click to buy NOW!
www.docu-track.com

3. Зі сторінки Standard помістимо на Form1 компонент PopupMenu1.
4. Ввійдемо у властивість Items компонента PopupMenu1 і задамо елементи контекстного меню:



Мал. 2.35. Вікно конструктора меню під час створення контекстного меню

5. Не виходячи з конструктора меню, подвійним щигликом по кожному елементі будемо викликати Редактор Коду. Так, для елемента меню **Рішення 1** оброблювач події OnClick буде мати вид:

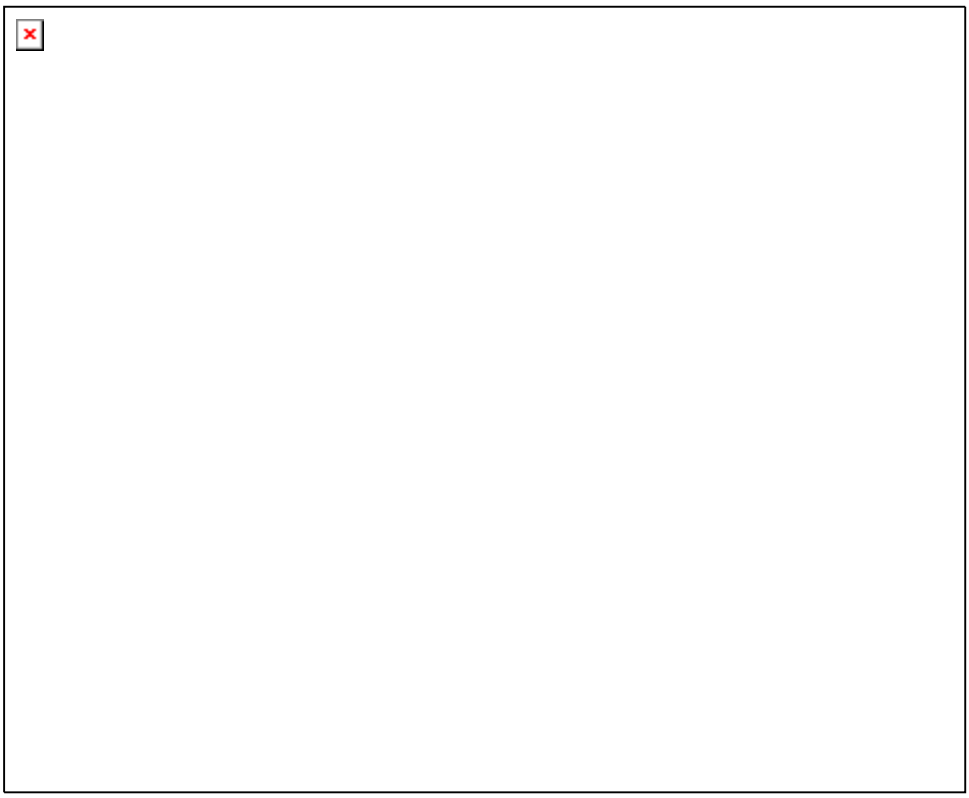
```
procedure TForm1.N11Click (Sender:TObject);
begin
  Button1Click(Button1);
end;
```

де Button1Click – процедура, створена раніше (див. приклад 2.9). Як параметр процедури можна використовувати вказівник на будь-який об'єкт, наприклад компонент Button1.

Аналогічно з елементів меню **Рішення 2** і **Очистити** викличемо процедури Button2Click і Button3Click відповідно.

6. Для того щоб локальне меню було доступним у будь-якій точці форми Form1, установимо її властивість PopupMenu рівною PopupMenu1.
7. Збережемо проект і запустимо його на виконання. Щиглик правою кнопкою миші активізує локальне меню (див. мал. 2.36).

PDF-XChange
Click to buy NOW!
www.docu-track.com



Мал. 2.36. Використання компонента Popup Menu

Приведемо текст модифікованого модуля.

Текст модуля Unit1.pas.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls, Grids, Menus;

type
  TForm1 = class(TForm)
    StringGrid1: TStringGrid;
    StringGrid2: TStringGrid;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    PopupMenu1: TPopupMenu;
    N11: TMenuItem;
    N21: TMenuItem;
    N1: TMenuItem;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
    procedure N11Click(Sender: TObject);
    procedure N21Click(Sender: TObject);
    procedure N1Click(Sender: TObject);
  private
    { Private declarations }
  public
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var   i:Integer;
begin
  for  i := 0  to 3 do
    begin
    StringGrid2.Cells[i,0]:=StringGrid1.Cells[i,3];
    StringGrid2.Cells[i,3]:=StringGrid1.Cells[i,0];
    StringGrid2.Cells[i,1]:=StringGrid1.Cells[i,1];
    StringGrid2.Cells[i,2]:=StringGrid1.Cells[i,2];
    end

end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  StringGrid2.Rows[0]:=StringGrid1.Rows[3];
  StringGrid2.Rows[1]:=StringGrid1.Rows[1];
  StringGrid2.Rows[2]:=StringGrid1.Rows[2];
  StringGrid2.Rows[3]:=StringGrid1.Rows[0];
end;

procedure TForm1.Button3Click(Sender: TObject);
var   i,j:Integer;
begin
  for  i := 0  to 3 do
    for  j := 0  to 3 do
       StringGrid2.Cells[j,i] := ' '
end;

procedure TForm1.N11Click(Sender: TObject);
begin
  Button1Click(Button1);
end;

procedure TForm1.N21Click(Sender: TObject);
begin
  Button2Click(Button2);
end;

procedure TForm1.N1Click(Sender: TObject);
begin
  Button3Click(Button3);
```

```
end;

end.
```

2.10. Діалогові вікна. Компоненти Open Dialog, Save Dialog, Font Dialog

2.10.1. Загальний опис

Операційні системи сімейства Windows підтримують стандартні діалогові вікна, призначені для відкриття і збереження файлів, вибору і настроювання шрифту, кольору, принтера і деякі інші. Delphi також підтримує стандартні діалоги. Для цього в бібліотеці компонентів Delphi існують спеціальні класи, що мають одного загального предка – TCommonDialog, який у свою чергу є безпосереднім нащадком класу TComponent. Діалогові вікна є прикладами невізуальних компонентів. Це виявляється, наприклад, у тім, що зовнішній вигляд компонента, розташовуваного на формі на етапі конструювання, не збігається з зовнішнім виглядом діалогового вікна, створюваного на етапі виконання програми.

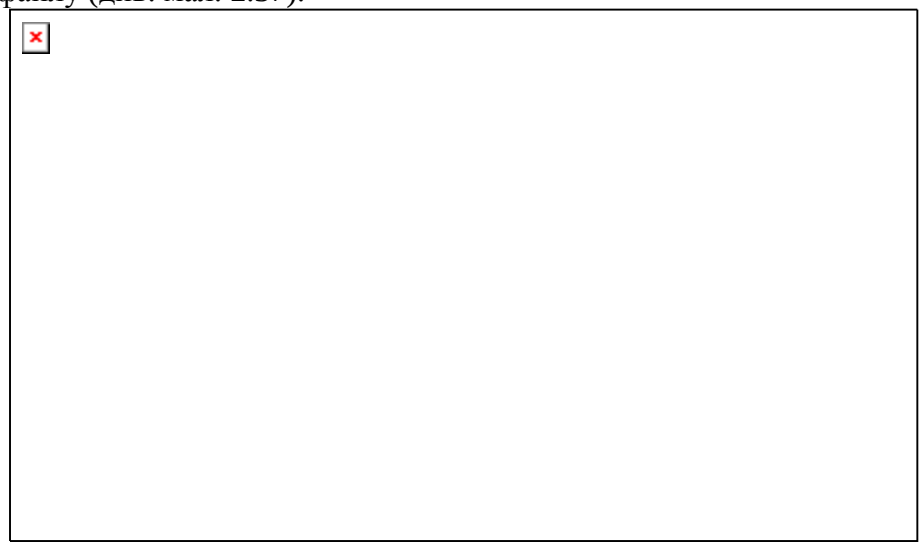
Особливістю реалізації діалогових вікон Delphi є те, що вони використовують стандартні можливості, які існують в операційних системах сімейства Windows. Наприклад, діалогові вікна містять інформацію на тій мові, що використовується в операційній системі.

2.10.2. Open Dialog – діалогове вікно вибору імені файлу, що відкривається

Ієрархія:

Object – TPersistent – TComponent – TCommonDialog.
Сторінка Палітри Компонентів: Dialogs.

Діалогове вікно вибору імені файлу, що відкривається, є екземпляром класу TOpenDialog. Воно призначено для перегляду файлової системи комп'ютера і вибору імені необхідного файлу (див. мал. 2.37).



Мал. 2.37. Діалогове вікно відкриття файлу

Помітимо, що компонент Open Dialog не призначений для автоматичного відкриття файлів. Він дозволяє лише одержати ім'я обраного користувачем файлу. Безпосереднє відкриття файлу здійснюється за допомогою стандартних процедур мови Object Pascal (див. частина 1) або спеціальних методів. Такі методи визначені, наприклад, у класі TStrings.

Розглянемо основні властивості класу TOpenDialog.

PDF-XChange Click to buy NOW! www.docu-track.com

property DefaultExt: string;
Містить розширення, що додається до імені файлу, якщо в нього не зазначене розширення. Може містити до трьох символів, не включаючи розділову крапку.

type TFileName = String;
property FileName: TFileName;
Містить ім'я обраного файлу. Це ж ім'я міститься в рядку **Ім'я файлу:** (File Name:) діалогового вікна.

property Files: TStrings;
Містить список імен виділених файлів. Властивість призначена тільки для читання.

property Filter: string;
Містить опис одного чи декількох файлових фільтрів. Файловий фільтр – це один чи кілька шаблонів імені файлу (масок файлу), що містять спеціальні символи. Наприклад, маска *.pas допоможе користувачу відображати в діалоговому вікні тільки файли, що мають розширення pas . У властивості Filter може міститися декілька пар послідовностей символів, розділених вертикальними лініями. Кожна така пара відповідає одному файловому фільтру і складається, у свою чергу, із двох частин, що також розділяються між собою вертикальною лінією. Перша частина задає текст, виведений для даного фільтра в комбінованому рядку **Тип файлів:** діалогового вікна, а друга частина містить сам фільтр. Фільтр являє собою перераховані через крапку з комою маски файлів. Наприклад, у результаті виконання оператора

```
OpenDialog1.Filter := ‘Файли модулів Delphi (*.pas)|*.pas|’+
                      ‘Текстові документи (*.txt, *.doc)|*.txt; *.doc’;
```

у комбінованому рядку **Тип файлів:** діалогового вікна буде міститися два рядки:

Файли модулів Delphi (*.pas)
і
Текстові документи (*.txt, *.doc)

Першому рядку відповідає файловий фільтр *.pas , а другому – *.txt; *.doc . Відповідно при виборі першого фільтра в діалоговому вікні відкриття файлу будуть відображатися тільки файли, що мають розширення pas , а при виборі другого фільтра – файли з розширеннями txt чи doc .

При виборі цієї властивості в Інспекторі Об'єктів на етапі конструювання форми відкривається допоміжне вікно, за допомогою якого можна задати тексти фільтрів і самі фільтри, не записуючи розділові вертикальні лінії (мал. 2.41). У цьому вікні є два стовпці. Перший стовпець, що має ім'я Filter Name, призначений для створення тексту фільтра, другий – Filter – призначений для завдання самого фільтра.

property FilterIndex: Integer;
Визначає, який елемент фільтра буде показаний за замовчуванням при відкритті діалогового вікна. Нумерація елементів починається з одиниці. Звичайно використовується перший елемент фільтра.

property InitialDir: string;
Визначає папку, уміст якої з'являється при відкритті діалогового вікна. Якщо значення цієї властивості не задано, то відкривається поточна папка чи та папка, що була відкрита при останнім звертанні користувача до діалогового вікна.

При використанні діалогового вікна відкриття файлу особливе місце займає метод Execute, визначений у класі TOpenDialog:

function Execute: Boolean; override;
Розміщає діалогове вікно на екрані в модальному режимі. Модальний режим означає, що виконання застосування припиняється доти, поки користувач не закриє модальне вікно. Функція повертає значення True, якщо вікно закрите кнопкою **Відкрити,** і False, якщо закрито кнопкою **Скасування** .

2.10.3. Save Dialog – діалогове вікно вибору імені файлу, що зберігається

Ієрархія:

TObject – TPersistent – TComponent – TCommonDialog – TOpenDialog .
Сторінка Палітри Компонентів: Dialogs.

Діалогове вікно Save Dialog дуже схоже на вікно Open Dialog, але на відміну від його використовується при збереженні файлу (див. мал. 2.38).

Мал. 2.38. Діалогове вікно збереження файлу

Діалогове вікно вибору імені файлу, що зберігається, є екземпляром класу TSaveDialog, породженого безпосередньо від класу TOpenDialog, і успадковує всієї його характеристики.

У класі TSaveDialog визначена своя функція Execute, що виконує ті ж самі дії, що й аналогічна функція класу TOpenDialog.

По своїй структурі і зовнішньому вигляді діалогові вікна Save Dialog і Open Dialog відрізняються незначно.

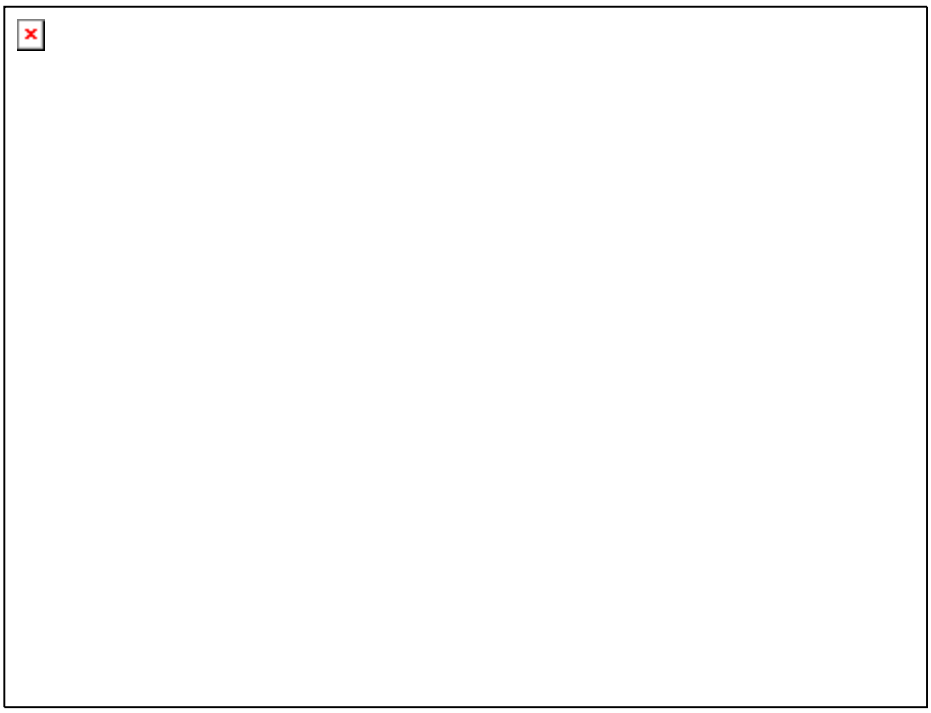
2.10.4. Font Dialog - діалогове вікно вибору шрифту

Ієрархія:

TObject – TPersistent – TComponent – TCommonDialog .
Сторінка Палітри Компонентів: Dialogs.

Діалогове вікно вибору шрифту Font Dialog дозволяє користувачу вибирати шрифт і встановлювати його характеристики (див. мал. 2.39).

PDF-XChange
Click to buy NOW!
www.docu-track.com



Мал. 2.39. Діалогове вікно вибору шрифту

Компонент Font Dialog є екземпляром класу TFontDialog, що породжений безпосередньо від класу TCommonDialog.

Відзначимо дві властивості, визначені в класі TFontDialog.

type TFontDialogDevice = (fdScreen, fdPrinter, fdBoth);
property Device: TFontDialogDevice;
Визначає, для якого пристрою задається шрифт. Може приймати наступні значення:

fdScreen – шрифт для екрана,
fdPrinter – шрифт для принтера,
fdBoth – шрифт для екрана і принтера.

За замовчуванням шрифт задається для екрана.

property Font: TFont;
Задає характеристики шрифту. Для відображення тексту буде використовуватися шрифт, що належить множині шрифтів, зареєстрованих в операційній системі Windows.

Так само, як і для компонентів Open Dialog і Save Dialog, діалогове вікно вибору шрифту активізується за допомогою методу Execute.

2.10.5. Приклад використання компонентів Open Dialog, Save Dialog і Font Dialog
Приклад 2.12.

Створимо простий текстовий редактор, що дозволив би за допомогою діалогових вікон зберігати і відкривати текстові файли, а також змінювати характеристики шрифту.

Рішення

1. Створимо нову папку: D:\MyProject\DIALOG.
2. Відкриємо новий проект за допомогою команди головного меню File | New | Application.
3. Розмістимо на Form1 наступні компоненти:

Мал. 2.40. Розташування компонентів Open Dialog, Save Dialog, Font Dialog, Memo і Button на формі

Компоненти OpenDialog1, SaveDialog1 і FontDialog1 виберемо зі сторінки Dialogs, а Memo1, Button1, Button2, і Button3 – зі сторінки Standard Палітри Компонентів.

4. Виберемо властивість Lines компонента Memo1 і клацнемо по кнопці із трьома крапками, що з'явилася. У вікні, що вивелося на екран, видалимо текст 'Memo1'. Натиснемо ОК. Ми домоглися того, що при запуску програми вікно редактора буде порожнім.

5. Виберемо властивість Filter компонента OpenDialog1 і клацнемо по кнопці із трьома крапками, що з'явилася. З'явиться діалогове вікно Filter Editor, за допомогою якого можна задати тексти фільтрів і самі фільтри (див. мал. 2.41).

У першому рядку у вікні Filter Name уведемо:

Текстові файли (*.txt,*.doc)

а у вікні Filter:

.txt;.doc

В другому рядку у вікні Filter Name уведемо:

Усі файли (*.*)

а у вікні Filter:

.

Натиснемо кнопку ОК.

Мал. 2.41. Вікно редактора файлових фільтрів

6. Для компонента SaveDialog1 значення властивості DefaultExt установимо рівним txt. Тобто, якщо при збереженні файлу розширення не буде зазначено, то за замовчуванням додасться розширення txt.
7. Кнопкам Button1, Button2, і Button3 установимо властивість Caption рівною 'Відкрити', 'Зберегти' і 'Шрифт' відповідно.
8. В оброблювач події OnClick для кнопки Button1 уставимо наступні оператори:

```
with OpenDialog1 do
  begin
  if not Execute then Exit;
  Memo1.Lines.LoadFromFile(FileName)
  end;
```

9. В оброблювач події OnClick для кнопки Button2 уставимо такі оператори:

```
with SaveDialog1 do
  begin
  if not Execute then Exit;
  Memo1.Lines.SaveToFile(FileName)
  end;
```

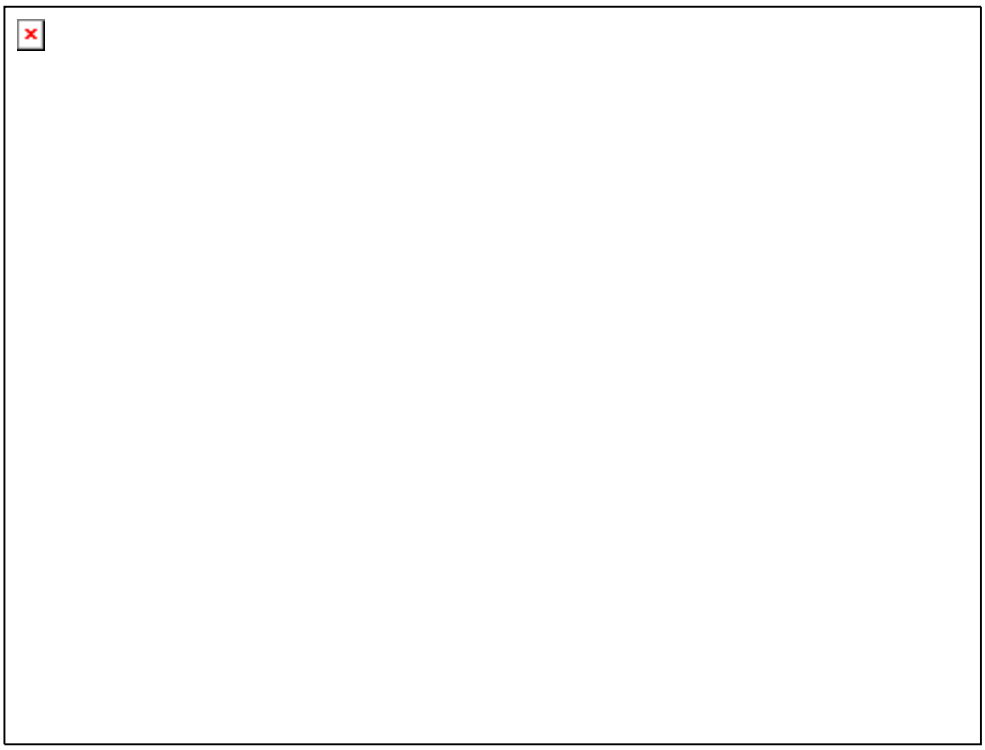
10. В оброблювач події OnClick для кнопки Button3 уставимо

```
with FontDialog1 do
  begin
  if not Execute then Exit;
  Memo1.Font := Font
  end;
```

11. Збережете проект і запустите його на виконання. Наберіть довільний текст у вікні редактора (див. мал. 2.42). Збережіть його в папці D:\MyProject\DIALOG.

Очистить за допомогою локального меню вікно редактора. Для появи локального меню варто клацнути правою клавішею миші, коли її вказівник знаходиться в межах вікна редактора. Відкрийте раніше створений файл. Змініть його шрифт.

PDF-XChange
Click to buy NOW!
www.docu-track.com



Мал. 2.42. Використання компонентів Open Dialog, Save Dialog і Font Dialog

Приведемо повний текст сформованого модуля.

Текст модуля Unit1.pas.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    OpenDialog1: TOpenDialog;
    SaveDialog1: TSaveDialog;
    FontDialog1: TFontDialog;
    Memo1: TMemo;
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  with OpenDialog1 do
    begin
    if not Execute then Exit;
    Memo1.Lines.LoadFromFile(FileName)
    end
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
  with SaveDialog1 do
    begin
    if not Execute then Exit;
    Memo1.Lines.SaveToFile(FileName)
    end
end;

procedure TForm1.Button3Click(Sender: TObject);
begin
  with FontDialog1 do
    begin
    if not Execute then Exit;
    Memo1.Font := Font
    end
end;

end.
```

2.11. Застосування з багатьма формами. Компоненти OleContainer і Panel

Ця глава присвячена формам. Але самі форми без розташованих на них компонентів не можуть виконувати корисну роботу. Глава містить приклади застосувань, у яких використовуються компоненти OleContainer і Panel. У зв'язку з цим буде приведений також і їхній опис.

2.11.1. Форми

Розробка застосувань у Delphi полягає у проектуванні форм, включенні в них компонентів і написанні програмного коду, що визначає поведінку форми. Форми в Delphi є основними будівельними блоками для застосування, що створюється. Значення форм настільки велике, що Delphi після запуску відразу ж створює нову порожню форму. У наступному пункті розглядаються основні властивості й методи форм.

2.11.1.1. Загальні характеристики форм

Ієрархія:

TObject – TPersistent – TComponent – TControl – TWinControl – TScrollingWinControl – TCustomForm.

Застосування, що працюють у Windows, найчастіше складаються з декількох вікон. Кожна форма в Delphi – це вікно застосування, що розробляється. Любій формі відповідає свій модуль, що містить опис класу форми. Якщо застосування містить більш однієї форми, то одна з них є головною. Головній формі передається керування після запуску застосування. Закриття головної форми приводить до закриття всіх інших вікон і означає завершення роботи застосування.

Як ми вже знаємо, кожне започатковане застосування містить у собі вже одну форму. Для того щоб додати нову форму необхідно виконати команду головного меню File|New|Form.

Якщо в застосуванні містяться дві форми, назвемо їх умовно A і B, причому в модулі форми A використовуються властивості й методи форми B, то модуль B повинний бути підключений до модуля A. Це означає, що в операторі uses модуля A повинне бути зазначене ім'я модуля B. Для вставки відповідного оператора необхідно переключитися в модуль A (Ctrl+F12) і виконати команду File|UseUnit головного меню. Виконання цієї команди приведе до вставки оператора uses у розділ implementation модуля A, що гарантує відсутність кругових посилань.

Форма може створюватися або автоматично при запуску програми, або її створює при необхідності програміст, записуючи в програмі необхідні для цього оператори. Головна форма завжди створюється автоматично.

За замовчуванням передбачається, що всі поміщені в проект форми створюються автоматично при запуску програми. У цьому випадку ніяких додаткових зусиль, зв'язаних із створенням форми, додавати не слід. Однак цю ситуацію можна змінити, використовуючи вкладку Forms вікна Project Options, що відкривається за допомогою команди головного меню Project|Options (див. мал. 2.53). На вкладці Forms маються два списки – автоматично створюваних форм (Auto-create forms) і доступних форм (Available forms). Кожну з форм проекту (за винятком головної форми) можна помістити в кожній з цих списків. У верхньому списку Main form, що випадає, можна вибрати головну форму, серед наявних у проекті.

Якщо форма створюється автоматично, то у файлі проекту міститься відповідний метод CreateForm, наприклад:

```
Application.Initialize;
Application.CreateForm(TFmain, Fmain);
Application.Run;
```

Приведемо опис методу CreateForm:

procedure CreateForm(FormClass: TFormClass; var Reference);
Створює нову форму. FormClass – ім'я класу створюваної форми, Reference – змінна, що містить вказівник на створену форму.

У випадку якщо форма не створюється автоматично, у процесі проектування вона лише підготовлюється для створення, тобто їй задаються відповідні характеристики. Підготовлену, але не створену автоматично форму можна створити програмно за допомогою конструктора класу форми Create:

constructor Create(AOwner: TComponent); override;
Створює і ініціалізує нову форму-об'єкт. AOwner – компонент-власник, у якому розміщується створювана форма (звичайно це є застосування Application).

У момент створення форми виникає подія OnCreate, що може бути використана для настроювання властивостей компонентів форми:

property OnCreate: TNotifyEvent;

PDF-XChange
Click to buy NOW!
www.docu-track.com

Подія, що виникає в момент створення форми.

Створення форми ще не означає, що вона відразу з'явиться на екрані. Зробити невидиму форму видимою можна за допомогою методів Show чи ShowModal:

procedure Show;
Робить форму видимою.

function ShowModal: Integer; virtual;
Показує форму в модальному режимі (див. пункт 2.11.1.2).

З методами Show і ShowModal тісно зв'язана властивість Visible:

property Visible: Boolean;
Показує, чи є форма видимою. Якщо форма є видимою, то Visible дорівнює true, незалежно від того чи знаходиться вона на передньому плані екрана чи закрита іншими вікнами. Для невидимої форми Visible дорівнює false. Методи Show і ShowModal установлюють властивість рівним true і розміщають форму поверх всіх інших вікон.

Якщо форма зроблена видимою за допомогою методу Show і не є дочірнім вікном у MDI-застосуванні (див. пункт 2.11.1.4), її можна зробити невидимою за допомогою методу Hide:

procedure Hide;
Робить форму невидимою і встановлює її властивість Visible рівною false.

Коли форма стає видимою чи невидимою виникають відповідно події OnShow чи OnHide:

property OnShow: TNotifyEvent;
Виникає, коли форма стає видимою. Може бути використане для настроювання зовнішнього вигляду форми на основі поточного стану застосування.

property OnHide: TNotifyEvent;
Виникає, коли форма стає невидимою.

Закриття будь-якої форми можна здійснити за допомогою методу Close:

procedure Close;
Закриває форму.

Закриття форми являє собою багатоступінчастий процес, що може мати різні завершення. Розглянемо цей процес докладніше.

Насамперед, метод Close викликає метод CloseQuery, що є функцією логічного типу. Якщо вона повертає значення true, форму можна закрити. Якщо CloseQuery повертає false, то операція закриття переривається. Метод CloseQuery формує подію OnCloseQuery.

Якщо оброблювач події OnCloseQuery поверне значення false (true) у своєму параметрі CanClose, сама функція CloseQuery також повертає значення false (true). За замовчуванням у параметр CanClose міститься значення true.

Якщо форма є батьківською у багатодокументному інтерфейсі, функція CloseQuery викликає аналогічні функції всіх дочірніх форм, і, якщо хоча б одна з них поверне значення false, функція CloseQuery батьківської форми також поверне значення false.

Якщо оброблювач події OnCloseQuery відсутній чи в цьому оброблювачі збережено для параметра CanClose значення true, то настає подія OnClose.

Оброблювач події OnClose повертає параметр Action, що визначає наслідки обробки події OnClose:

caNone – форма не закривається (значення за замовчуванням для дочірніх форм багатодокументного інтерфейсу);

caHide – форма стає невидимою, але з застосування не вилучається;

caFree – форма дійсно вилучається з застосування і з динамічної області пам'яті;

caMinimize – форма мінімізується.

Щодо значення caHide варто помітити, що воно неприпустимо для форм із MDI-застосувань. Для всіх інших форм це значення є значенням за замовчуванням.

Головна форма закривається в будь-якому випадку, крім випадку caNone.

Якщо в параметрі Action оброблювача події OnClose задане значення caFree, то при звільненні динамічної пам'яті виникає остання подія – OnDestroy.

Приведемо опис методу CloseQuery і всіх згаданих подій:

function CloseQuery: Boolean; virtual;
Визначає – чи можна закрити форму. Якщо можна, то повертає true, якщо немає – false.

type TCloseQueryEvent = procedure(Sender: TObject; var CanClose: Boolean) of object;
property OnCloseQuery: TCloseQueryEvent;
Подія виникає при виконанні методу Close чи коли вікно закривається стандартною кнопкою закриття вікна, що розташована в заголовку. Якщо параметр CanClose оброблювача повертає значення true, процес закриття буде продовжений. Якщо повертає false – процес закриття переривається. За замовчуванням параметр має значення true.

type
  TCloseAction = (caNone, caHide, caFree, caMinimize);
  TCloseEvent = procedure(Sender: TObject; var Action: TCloseAction) of object;
property OnClose: TCloseEvent;
Подія виникає при закритті вікна після події OnCloseQuery. Оброблювач повертає в параметрі Action значення (див. вище), яке визначає, що варто розуміти під закриттям форми.

property OnDestroy: TNotifyEvent;
Виникає при знищенні вікна і вилученні його з динамічної пам'яті.

Таким чином, при закритті форми програміст може передбачити створення оброблювачів розглянутих подій. Звичайно в оброблювачі події OnCloseQuery аналізується поточний стан застосування, виводиться на екран пропозиція зберегти не збережені дані чи документи і т.п. На підставі отриманої відповіді приймається рішення – продовжити закриття форми чи ні. В оброблювачі події OnClose програміст визначає, що, на його думку, варто розуміти під закриттям форми.

Підкреслимо, що варто розрізняти невидиму на екрані форму і знищену форму. Невидима форма лише не відображається на екрані, але вона присутня в динамічній області пам'яті, зберігає усі свої параметри і при необхідності може бути знову виведена на екран.

Знищена форма вилучається з динамічної області пам'яті, і для повторного використання її варто знову створити.

Ми розглянули характеристики форм, що забезпечують її життєвий цикл. Нижче приведені найбільш важливі властивості, що ще не були розглянуті:

property Active: Boolean;

Властивість тільки для читання. Містить true, якщо вікно активне, тобто має фокус уведення. У противному випадку містить false.

property ActiveControl: TWinControl;
Визначає дочірній компонент, що має фокус уведення.

type TBorderIcon = (biSystemMenu, biMinimize, biMaximize, biHelp);
TBorderIcons = set of TBorderIcon;
property BorderIcons:TBorderIcons;
Визначає, які кнопки будуть розташовуватися в заголовку вікна. У таблиці 2.2 приведені константи, з яких можуть складатися припустимі множини.
Таблиця 2.2. Значення властивості BorderIcons

Значення	**Пояснення**
biSystemMenu	Є кнопка виклику системного меню
biMinimize	Є кнопка мінімізації
biMaximize	Є кнопка максимізації
biHelp	Є кнопка виклику довідкової служби

type
TFormBorderStyle = (bsNone, bsSingle, bsSizeable, bsDialog, bsToolWindow, bsSizeToolWin);
property BorderStyle: TFormBorderStyle;
Визначає зовнішній вигляд і поводження рамки вікна. У таблиці 2.3 припустимі значення властивості BorderStyle.
Таблиця 2.3. Значення властивості BorderStyle

Значення	**Пояснення**
bsDialog	Рамка діалогового вікна. Зміна розмірів вікна не допускається
bsSingle	Рамка товщиною в 1 пиксель. Зміна розмірів вікна не допускається
bsNone	Вікно не має рамки і заголовка. Зміна розмірів і положення вікна не допускається
bsSizeable	Звичайна рамка (значення за замовчуванням)
bsToolWindow	подібно bsSingle, але зі зменшеним заголовком
bsSizeToolWin	подібно bsSizeable, але зі зменшеним заголовком

type TFormStyle = (fsNormal, fsMDIChild, fsMDIForm, fsStayOnTop);
property FormStyle: TFormStyle;
Визначає стиль форми. Припустимі значення приведені в таблиці 2.4.
Таблиця 2.4. Значення властивості FormStyle

Значення	**Пояснення**
fsNormal	Звичайна форма
fsMDIChild	Форма є дочірньої в MDI-застосуванні
fsMDIForm	Форма є батьківської в MDI-застосуванні
fsStayOnTop	Форма повинна розташовуватися над всіма іншими формами проекту

property Icon: TIcon;
Містить піктограму форми. Для головної форми визначає піктограму всього застосування.

type TPosition = (poDesigned, poDefault, poDefaultPosOnly, poDefaultSizeOnly, poScreenCenter, poDesktopCenter, poMainFormCenter, poOwnerFormCenter);
property Position: TPosition;
Визначає положення і розміри форми в момент її появи на екрані. Припустимі значення приведені в таблиці 2.5.
Таблиця 2.5. Значення властивості Position

Значення	**Пояснення**
poDesigned	Положення і розміри такі ж, як і на етапі конструювання вікна
poDefault	Положення і розміри визначає Windows
poDefaultPosOnly	Положення визначає Windows
poDefaultSizeOnly	Розміри визначає Windows
poScreenCenter	Форма розташовується в центрі екрана
poDesktopCenter	Для застосувань, що працюють з одним монітором, збігається з poScreenCenter
poMainFormCenter	Форма розташовується в центрі головної форми
poOwnerFormCenter	Форма розташовується в центрі форми-власника

type TWindowState = (wsNormal, wsMinimized, wsMaximized);
property WindowState: TWindowState;
Визначає стан вікна в момент його появи на екрані. Припустимі значення приведені в таблиці 2.6.
Таблиця 2.6. Значення властивості WindowState

Значення	**Пояснення**
wsNormal	Звичайне вікно
wsMinimized	Вікно мінімізоване
wsMaximized	Вікно максимізоване

2.11.1.2. Модальні форми

Модальні форми призначені для організації діалогу користувача з програмою і тому їх ще називають діалоговими вікнами. Такі вікна звичайно відкривається лише на час передачі інформації й у так називаному модальному режимі. Модальний режим припускає, що всі повідомлення, що надходять у програму у випадку відкритого модального вікна, обробляються тільки цим вікном. У зв'язку з цим користувач не може вийти в програмі за границі цього вікна, поки воно не буде закрито.

Для того щоб вікно працювало в модальному режимі, його варто відобразити на екрані не за допомогою методу Show, а за допомогою методу ShowModal.

function ShowModal: Integer; virtual;

Функція призначена для відображення форми в модальному режимі. Робить форму видимою, задаючи її властивості Visible значення True, і модальною. Повертає значення властивості ModalResult форми, коли ця властивість стане відмінна від нуля.

Робота з модальним вікном здійснюється в такий спосіб. Спочатку вікно виводиться на екран за допомогою методу ShowModal. Метод, вивівши вікно на екран, не припиняє свою роботу, а чекає його закриття.

Як правило, припинення роботи з модальним вікном здійснюється при натисканні тієї чи іншої розміщеної в ньому кнопки. Оброблювач події, що виникає при натисканні кнопки, повинний містити в цьому випадку оператор, за допомогою якого властивості ModalResult привласнюється ненульове значення. Нижче приведений опис властивості ModalResult.

type TModalResult = Low(Integer)..High(Integer);
property ModalResult: TModalResult;
Властивість використовується для закриття форми, відображеної в модальному режимі. Доступно тільки на етапі виконання програми. За замовчуванням, перед закриттям форми, властивість має значення mrNone, що має числовий еквівалент 0. Присвоєння властивості ModalResult будь-якого ненульового значення приводить до закриття форми. Константи, що використовуються для закриття модальної форми, приведені в таблиці 2.7.
Таблиця 2.7. Значення властивості ModalResult для модальної форми

PDF-XChange Click to buy NOW! www.docu-track.com

Константа	Числовий еквівалент	Значення
mrNone	0	Значення властивості ModalResult за замовчуванням перед закриттям модальної форми
mrOk	1	Натиснута кнопка OK
mrCancel	2	Натиснута кнопка Cancel
mrAbort	3	Натиснута кнопка Abort
mrRetry	4	Натиснута кнопка Retry
mrIgnore	5	Натиснута кнопка Ignore
mrYes	6	Натиснута кнопка Yes
mrNo	7	Натиснута кнопка No
mrAll	8	Натиснута кнопка ALL
mrNoToAll	9	Натиснута кнопка NO TO ALL
mrYesToAll	10	Натиснута кнопка YES TO ALL

Якщо модальна форма закрита за допомогою кнопки Cancel, або стандартної кнопки закриття вікна, розташованої в заголовку вікна, або методу Close, то властивості ModalResult буде привласнене значення mrCancel, що еквівалентно числовому значенню 2. Таким чином, усі константи з таблиці 2.7, крім значень 0 і 2, носять умовний характер, і програміст може їх використовувати за своїм розсудом.

Значення, що привласнюється властивості ModalResult, стає значенням, що повертається функцією ShowModal, яка була використана для відображення форми в модальному режимі. Значення, яке повертається у властивості ModalResult, можна проаналізувати і визначити, якою кнопкою була закрита модальна форма.

Властивість ModalResult форми може автоматично змінюватися деякими компонентами, наприклад кнопками класу TButton. У цих кнопок є однойменна властивість ModalResult, що має за замовчуванням значення mrNone (0). На етапі проектування форми цій властивості можна задати деяке ненульове значення. При виконанні програми щиглик по такій кнопці приведе до відповідної зміні властивості ModalResult форми. Таким чином, якщо використовувати властивість ModalResult кнопки, то можна не писати оброблювач події, зв'язаний з натисканням кнопки.

Закриття форми за допомогою властивості ModalResult за замовчуванням не означає її вилучення з динамічної області пам'яті, а робить тільке форму невидимою. При цьому інформація, що зберігається у формі доступна з інших модулів застосування. Якщо ми хочемо видалити форму з пам'яті, то можна, наприклад, скористатися методом Free:

```
Fmyform := TFmyform.Create(Self);
Fmyform.showmodal;
Fmyform.Free;
Fmyform := nil;
```

тут Fmyform – форма-об'єкт класу TFmyform. Присвоєння вказівнику на форму Fmyform значення nil обов'язково лише в тому випадку, якщо в застосуванні передбачена можливість повторного створення форми-об'єкта класу TFmyform.

Крім того, як ми вже знаємо з пункту 2.11.1.1, форма може бути знищена в оброблювачі події OnClose:

```
procedure TFmyform.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  Action := caFree;
  Fmyform := nil;
end;
```

Оператор

```
Fmyform := nil;
```

має той же зміст, що і для попереднього фрагмента.

PDF-XChange
Click to buy NOW!
www.docu-track.com

2.11.1.3. Особливості створення SDI-застосувань

При розробці застосування для Windows з декількома формами необхідно вибрати стиль інтерфейсу для свого застосування. Умовно наявні стилі можна розбити на наступні типи:

1) SDI (Single Document Interface) – інтерфейс з одним документом;
2) MDI (Multiple Document Interface) – багатодокументний інтерфейс;
3) комбінація стилів SDI і MDI.

Хоча назва SDI припускає, що застосування складається з єдиного вікна, найчастіше повноцінне SDI-застосування має кілька вікон. Прикладами SDI-застосувань є Delphi і Visual Basic фірми Microsoft. Головною особливістю SDI-стилю є наявність багатьох вікон, що не прив'язані до клієнтської області головного вікна. Звичайне SDI-застосування складається з головного вікна, що постійно є присутнім на екрані, і додаткових вікон, що можуть займати на екрані довільне положення.

Зробити видимими додаткові вікна можна за допомогою методів Show чи Showmodal. Вікно, відкрите за допомогою методу Show, можна зробити невидимим методом Hide, а закрити – методом Close. Властивість ModalResult не використовується в цьому випадку для закриття вікна, але може бути використане для передачі інформації про те, якою кнопкою було закрите вікно. Відзначимо також, що у вікні, відкритому методом Show, метод Close не змінює значення властивості ModalResult . Якщо вікно відкрите методом Showmodal, то, як ми вже знаємо, його варто закрити за допомогою властивості ModalResult.

Головне вікно може бути закрито методом Close. Закриття головного вікна приводить до закриття всіх додаткових вікон.

2.11.1.4. Особливості створення MDI-застосувань

Прикладами MDI-застосувань є MS Word, MS Excel, Mathcad і т.д. У відповідності зі стилем MDI застосування має батьківське вікно, у клієнтській області якого відображаються одне чи кілька дочірніх вікон, інакше, вікон документів. Дочірні вікна розташовуються в клієнтській області батьківського вікна і не можуть вийти за її межі. Дочірні вікна можуть бути мінімізовані, максимізовані, а також приймати будь-як розміри і положення в межах батьківського вікна.

При створенні багатодокументного інтерфейсу властивості FormStyle батьківської форми повинне бути встановлене значення fsMDIForm. В усіх дочірніх формах цій властивості повинне бути задане значення fsMDIChild.

Для випадку багатодокументного інтерфейсу з'являються додаткові властивості:

property ActiveMDIChild: TForm;
Властивість тільки для читання. Містить вказівник на дочірнє вікно, що має фокус. Може використовуватися для маніпуляцій з активним дочірнім вікном.

property MDIChildCount: Integer;
Властивість тільки для читання. Містить кількість відкритих дочірніх вікон.

property MDIChildren[I: Integer]: TForm;
Властивість тільки для читання. Дає доступ до I-го вікна. Нумерація дочірніх вікон змінюється щораз при зміні активного вікна. Отже властивість може бути корисна, якщо потрібно виконати перебір усіх дочірніх вікон, а не одержати доступ до якогось конкретного вікна. Властивість не може бути використана, якщо до звертання до неї було вилучено з пам'яті яке-небудь дочірнє вікно.

type TTileMode = (tbHorizontal, tbVertical);
property TileMode: TTileMode;

PDF-XChange
Click to buy NOW!
www.docu-track.com

Визначає спосіб розміщення дочірніх вікон у клієнтській області батьківського вікна. Саме розміщення здійснюється методом Tile, що розташовує дочірні вікна так, щоб вони не перекривалися. Значення властивості – tbHorizontal і tbVertical – забезпечують відповідно горизонтальне і вертикальне розташування дочірніх вікон.

Для керування дочірніми вікнами в батьківському вікні доступні наступні методи:

procedure ArrangeIcons;
Розміщає піктограми мінімізованих дочірніх вікон без перекриття і впритул друг до друга.

procedure Cascade;
Розміщає в батьківському вікні всі дочірні вікна каскадно (з перекриттям і зі зрушенням відносно одно одного).

procedure Next;
Робить активним наступне дочірнє вікно.

procedure Previous;
Робить активним попереднє дочірнє вікно.

procedure Tile;
Розміщає всі дочірні вікна в батьківському вікні впритул (без перекриття).

Створення, відображення і закриття дочірніх вікон у MDI-застосуваннях має свої особливості. Дочірні вікна в MDI-застосуваннях не можуть бути автоматично створюваними. Для того щоб зробити видимим дочірнє вікно варто скористатися методом Show, але ні в якому разі методом Showmodal. Вікно, відкрите за допомогою методу Show, не можна зробити невидимим методом Hide. Закрити дочірнє вікно можна методом Close. Оскільки в MDI-застосуваннях батьківське і дочірнє вікна не можна зробити невидимими, то в оброблювачі події OnClose параметру Action не можна привласнювати значення caHide.

Батьківське вікно може бути закрито методом Close. Як і для SDI-застосування, закриття батьківського вікна приводить до закриття всіх дочірніх вікон.

2.11.2. Компонент OleContainer

Ієрархія:

TObject – TPersistent – TComponent – TControl – TWinControl – TCustomControl.

Сторінка Палітри Компонентів: System.

OleContainer використовується в тих випадках, коли необхідно взаємодіяти з об'єктами, що є зовнішніми стосовно Delphi, наприклад, з документом редактора MS Word чи електронною таблицею MS Excel. При цьому застосування, що створило об'єкт, повинно підтримувати технологію OLE. Компонент OleContainer відкриває широкі можливості для використання файлів, створених в інших застосуваннях.

Нижче приведені основні властивості і методи компонента OleContainer:

type TAutoActivate = (aaManual, aaGetFocus, aaDoubleClick);
property AutoActivate: TAutoActivate;
Визначає спосіб активізації OLE-об'єкта, поміщеного в компонент OleContainer. Можливі значення властивості приведені в таблиці 2.8.

Таблиця 2.8. Значення властивості AutoActivate

Значення	**Пояснення**
aaManual	OLE-об'єкт може бути активізований тільки програмно за допомогою методу DoVerb(з параметром ovShow)
aaGetFocus	OLE-об'єкт буде активізований при одержанні їм фокуса
aaDoubleClick	Значення за замовчуванням. OLE-об'єкт буде активізований подвійним щигликом чи натисканням клавіші Enter коли компонент OleContainer має фокус уведення

property AutoVerbMenu: Boolean;
Якщо властивість установлена рівною true, то на етапі виконання для OLE-об'єкта, поміщеного в компонент OleContainer, буде створюватися контекстне меню, у противному випадку – ні.

procedure Copy;
Копіює OLE-об'єкт у буфер обміну Windows.

procedure CreateLinkToFile(FileName: string; Iconic: Boolean);
Даний метод аналогічний методу CreateObjectFromFile. Відмінність полягає в тім, що CreateLinkToFile не впроваджує, а зв'язує файл із застосуванням.

Procedure CreateObjectFromFile(const FileName: string; Iconic: Boolean);
Створює і впроваджує OLE-об'єкт у компонент OleContainer з файлу, ім'я якого міститься в параметрі FileName. Другий параметр – Iconic – визначає, у якому вигляді буде відображатися об'єкт. Якщо Iconic містить true, то об'єкт відображається у вигляді піктограми, у противному випадку – у вигляді вихідного файла.

function InsertObjectDialog: Boolean;
Метод InsertObjectDialog призначений для створення OLE-об'єкта. OLE-об'єкт може бути впровадженим чи зв'язаним. Метод викликає діалогове вікно **Вставка об'єкта** (мал.2.43), у якому треба вибрати тип об'єкта, і завантажує об'єкт у контейнер. InsertObjectDialog повертає true, якщо діалогове вікно **Вставка об'єкта** було успішно відображене, і користувач вибрав кнопку OK. У противному випадку InsertObjectDialog повертає false.

procedure LoadFromFile(const FileName: string);
Завантажує в OleContainer тільки файли, що були збережені раніше за допомогою методу SaveToFile і містить OLE-об'єкти.

procedure Paste;
Уставляє вміст буфера обміну Windows у OleContainer як впроваджений об'єкт.

procedure SaveAsDocument(const FileName: string);
Зберігає OLE-об'єкт у зазначеному файлі у вихідному форматі.

procedure SaveToFile(const FileName: string);
Зберігає в зазначеному файлі OLE-об'єкт. Збережений OLE-об'єкт може бути надалі завантажений у компонент OleContainer за допомогою методу LoadFromFile.

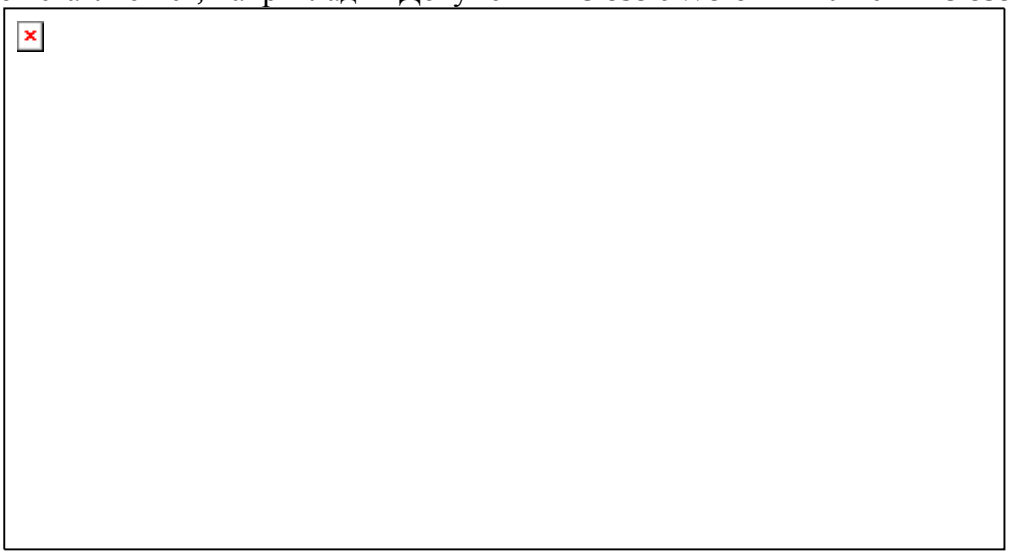
Звичайно використання компонента OleContainer відбувається за наступною схемою. Спочатку в компонент OleContainer завантажується деякий OLE-об'єкт, створений за допомогою якого-небудь OLE-сервера, установленого на вашому комп'ютері. У прикладах, розглянутих у пунктах 2.11.4 – 2.11.6, як OLE-сервер використовується MS Word. На етапі

PDF-XChange
Click to buy NOW!
www.docu-track.com

виконання застосування, що розроблюється, OLE-об'єкт, поміщений у компонент OleContainer, можна активізувати. При цьому буде запущена програма, що створила цей OLE-об'єкт. Таким чином, з'являється можливість переглядати і редагувати OLE-об'єкт. Якщо операція редагування передбачена, то, мабуть, варто передбачити й операцію збереження даних.

Помістити в OleContainer деякий об'єкт можна як на етапі проектування, так і на етапі виконання програми.

Якщо на етапі проектування виконати по компоненту OleContainer подвійний щиглик лівою кнопкою миші чи вибрати в контекстному меню команду Insert Object, то на екрані з'явиться діалогове вікно **Вставка об'єкта** (див. мал. 2.43), у якому потрібно вказати тип об'єкта, що вставляється, наприклад - «Документ Microsoft Word» чи «Лист Microsoft Excel».

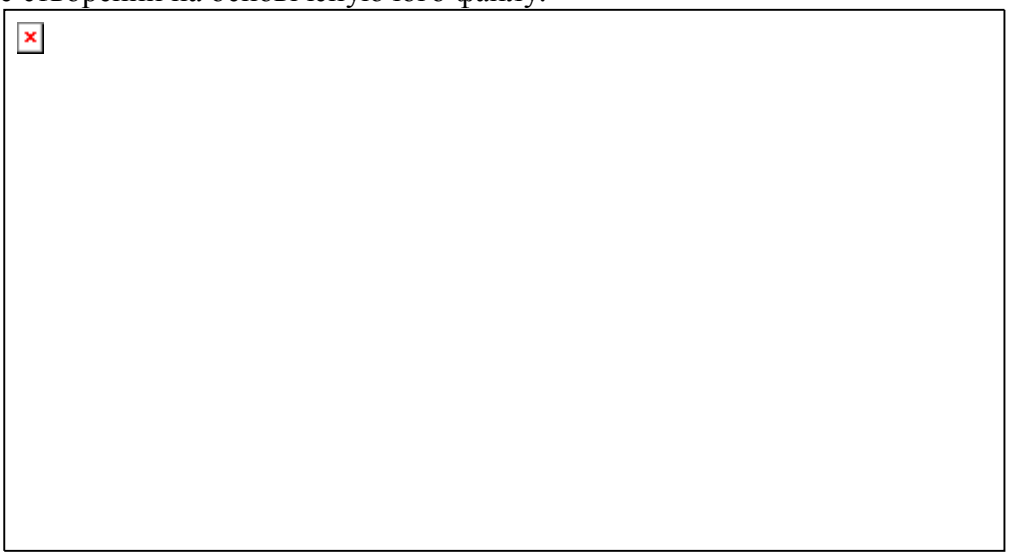


Мал. 2.43. Створення нового OLE-об'єкта

Тип об'єкта, що вставляється, вибирається зі **списку Тип** об'єкта. Після натискання на клавішу ОК запуститься OLE-сервер для створення нового OLE-об'єкта, що потім буде вставлений у компонент OleContainer.

Розглянутий спосіб вставки OLE-об'єкта здійснювався при перемикачі, встановленому в положення **Створити новий**, і був упровадженням OLE-об'єкта в компонент OleContainer.

Якщо установити перемикач у положення **Створити з файлу** (див. мал. 2.44), то OLE-об'єкт буде створений на основі існуючого файлу.



Мал. 2.44. Створення OLE-об'єкта на основі існуючого файлу

Створений на основі існуючого файлу OLE-об'єкт може бути впровадженим чи зв'язаним з компонентом OleContainer. Для зв'язування необхідно установити прапорець **Зв'язок**.

Активізувати об'єкт, що знаходиться в компоненті OleContainer можна як на етапі проектування, так і на етапі виконання, використовуючи контекстне меню компонента або виконавши по компоненті подвійний щиглик лівою кнопкою миші.

У залежності від того, використовувалося впровадження чи зв'язування, OLE-сервер по-різному поводиться на етапі виконання при активізації OLE-об'єкта. Розглянемо ці відмінності, узявши як приклад OLE-сервера MS Word.

Якщо використовувалося впровадження, то вікно Word у працюючому застосуванні буде обмежено розмірами компонента OleContainer. Цікавою особливістю поводження Word є те, що його головне меню буде убудоване в головне меню застосування, що розроблюється. Причому, якщо в головному меню застосування міститься пункт **Файл**, то він замінить однойменний пункт програми Word. Якщо в застосуванні відсутне головне меню, то на етапі виконання головне меню Word також буде недоступним. Звідси випливає, що застосування, що розроблюється, повинно мати головне меню, у пункті **Файл** якого можуть бути передбачені операції відкриття і збереження даних.

Якщо використовувалося зв'язування, то при активізації OLE-об'єкта під час виконання програми вікно програми Word буде мати звичний вид і займе весь екран.

Для відкриття вікна **Вставка об'єкта** під час роботи застосування варто скористатися методом InsertObjectDialog. Для упровадження файлу під час виконання необхідно викликати метод CreateObjectFromFile, а для зв'язування файлу з застосуванням – метод CreateLinkToFile.

Методи SaveToFile і LoadFromFile дозволяють відповідно створювати і відкривати файли спеціального типу, що містять OLE-об'єкти. Ці файли не можна прочитати за допомогою програми, що створила відповідні OLE-об'єкти.

Якщо необхідно зберегти OLE-об'єкт у файлі зі збереженням формату, що відповідає даному типу файлів, то варто використовувати метод SaveAsDocument.

У прикладах програм, розглянутих у пунктах 2.11.4 – 2.11.6 OleContainer використовується тільки для відображення даних, створених у MS Word, тобто, як переглядач (viewer). Тому така операція як активізація OLE-об'єкта стає зайвою. Для того, щоб зробити активізацію недоступної на етапі виконання можна установити властивість AutoActivate рівною aaManual, а властивість AutoVerbMenu рівною False.

2.11.3. Компонент Panel

Ієрархія:

TObject – TPersistent – TComponent – TControl – TWinControl – TCustomControl – TCustomPanel.

Сторінка Палітри Компонентів: Standard.

Компонент Panel являє собою контейнер, у який можна помістити інші компоненти. Найчастіше панелі використовуються для угруповання на формі компонентів, функціонально зв'язаних між собою – перемикачів, вимикачів, рядків уведення, кнопок і т.п. При цьому можна використовувати наявні в компоненті розвиті засоби створення тривимірних ефектів. Ці ефекти ґрунтуються на маніпулюванні двома рамками компонента – внутрішньою і зовнішньою.

Іншою важливою областю застосування панелей є створення вікон з регульованими розмірами. При цьому використовуються такі властивості панелі, як Align і Anchors, описані в пункті 2.3.5.

Нарешті, панель можна використовувати для виведення повідомлень чи надписів, використовуючи її властивість Caption.

Нижче приведені властивості, специфічні для панелі:

PDF-XChange Click to buy NOW! www.docu-track.com

type TBevelCut = (bvNone, bvLowered, bvRaised, bvSpace);
type TPanelBevel = TBevelCut;
property BevelInner: TPanelBevel;
Задає тип внутрішньої рамки. Можливі значення властивості приведені в таблиці 2.9.
Таблиця 2.9. Значення властивості BevelInner

Значення	**Пояснення**
bvNone	Рамка відсутня
bvLowered	Утиснена рамка
bvRaised	Опукла рамка
bvSpace	Рамки нема, але для неї виділяється місце товщиною 1 піксель

property BevelOuter: TPanelBevel;
Задає тип зовнішньої рамки. Приймає ті ж значення, що і властивість BevelInner.

type TBevelWidth = 1..MaxInt;
property BevelWidth: TBevelWidth;
Задає товщину в пікселях внутрішньої і зовнішньої рамок панелі.

property BorderStyle: TBorderStyle;
Задає наявність границі елемента упрваління. Можливі значення властивості приведені в таблиці 2.10.
Таблиця 2.10. Значення властивості BorderStyle

Значення	**Пояснення**
bsNone	Немає границі
bsSingle	Одинична границя

type TBorderWidth = 0..MaxInt;
property BorderWidth: TBorderWidth;
Задає відстань у пікселях між внутрішньою і зовнішньою рамками.

property FullRepaint: Boolean;
Якщо властивість установлена рівною true, то панель із усіма компонентами, що знаходяться в ній, перемальовується при зміні розмірів форми, у якій вона знаходиться. У противному випадку перемальовується тільки область, що знаходиться між зовнішньою і внутрішньою рамками.

2.11.4. Приклад створення застосування з модальними вікнами

У цьому і наступних пунктах ми розглянемо особливості розробки багатовіконих застосувань різних типів: з модальними вікнами, SDI-застосування і MDI-застосування. Для того щоб краще усвідомити відмінності між ними, як приклади будуть розглядатися рішення однієї і тієї ж задачі – створення контролюючої системи «Електронний екзаменатор». Варто помітити, що зазначена вище розбивка на типи чи, інакше, стилі є умовною, оскільки в одному і тому ж самому застосуванні можуть бути присутніми різні їхні комбінації.

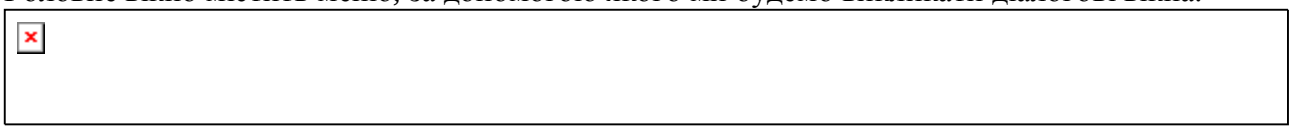
Оскільки відкриття форм як модальних використовується на практиці найчастіше, то перший приклад присвячений створенню застосування з модальними вікнами.

2.11.4.1. Огляд готового застосування

Розглянемо наступну задачу. Нехай потрібно створити застосування, що:
1) може відображати довільні документи, створені за допомогою текстового процесора Word;
2) дозволяє вибирати правильну відповідь з наданого набору варіантів;
3) веде статистичний облік кількості правильних і неправильних відповідей.

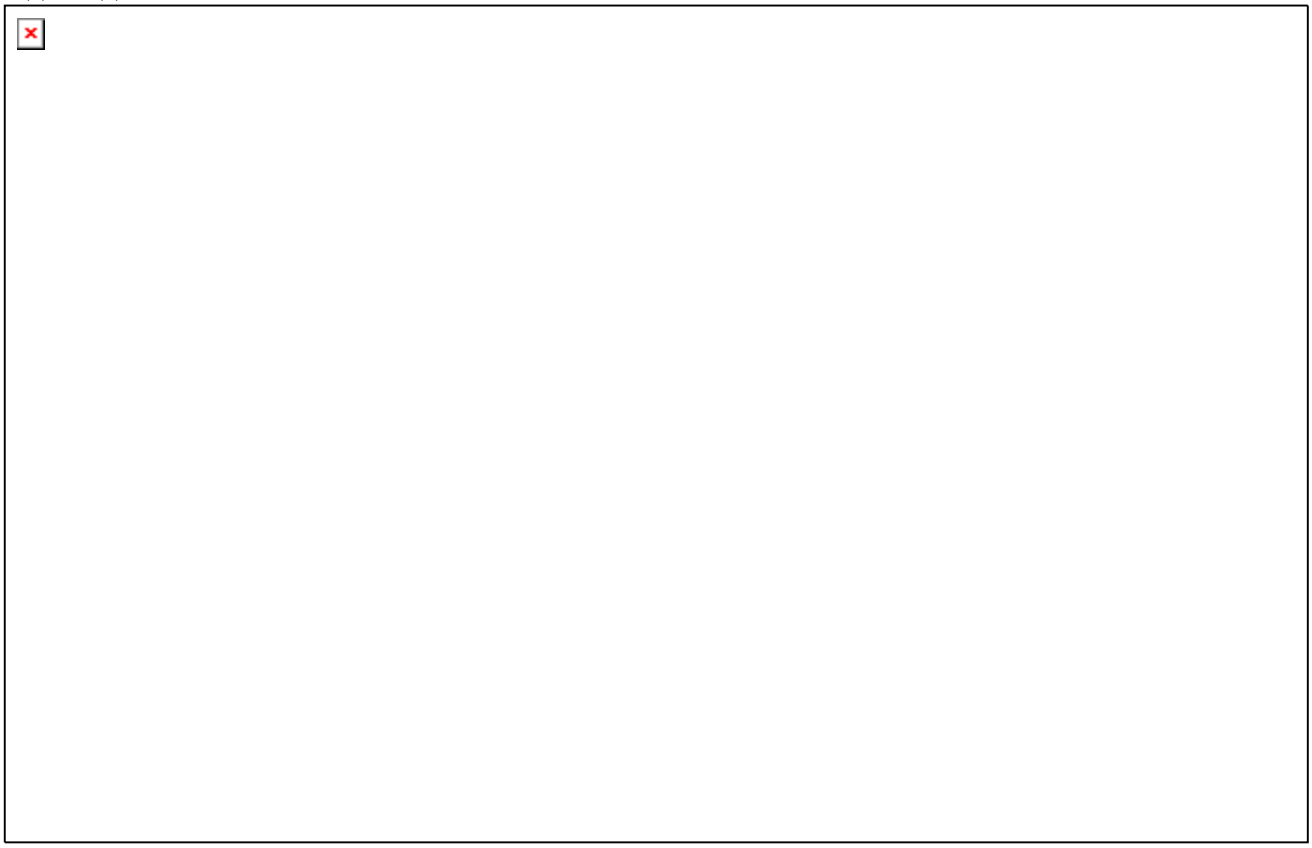
Назвемо розроблювальне застосування «Електронний екзаменатор». Відразу обмовимося, що наше застосування не є повноцінною контролюючою системою, а являє собою лише деяку її частину, у якій вирішені подзадачи перераховані вище.

Після запуску застосування, на екрані дисплея з'являється його головне вікно (див. мал. 2.45). Головне вікно застосування «Електронний екзаменатор» подібно головному вікну Delphi і являє собою нешироку смугу, розташовану у верхній частині екрана дисплея. Головне вікно містить меню, за допомогою якого ми будемо викликати діалогові вікна.



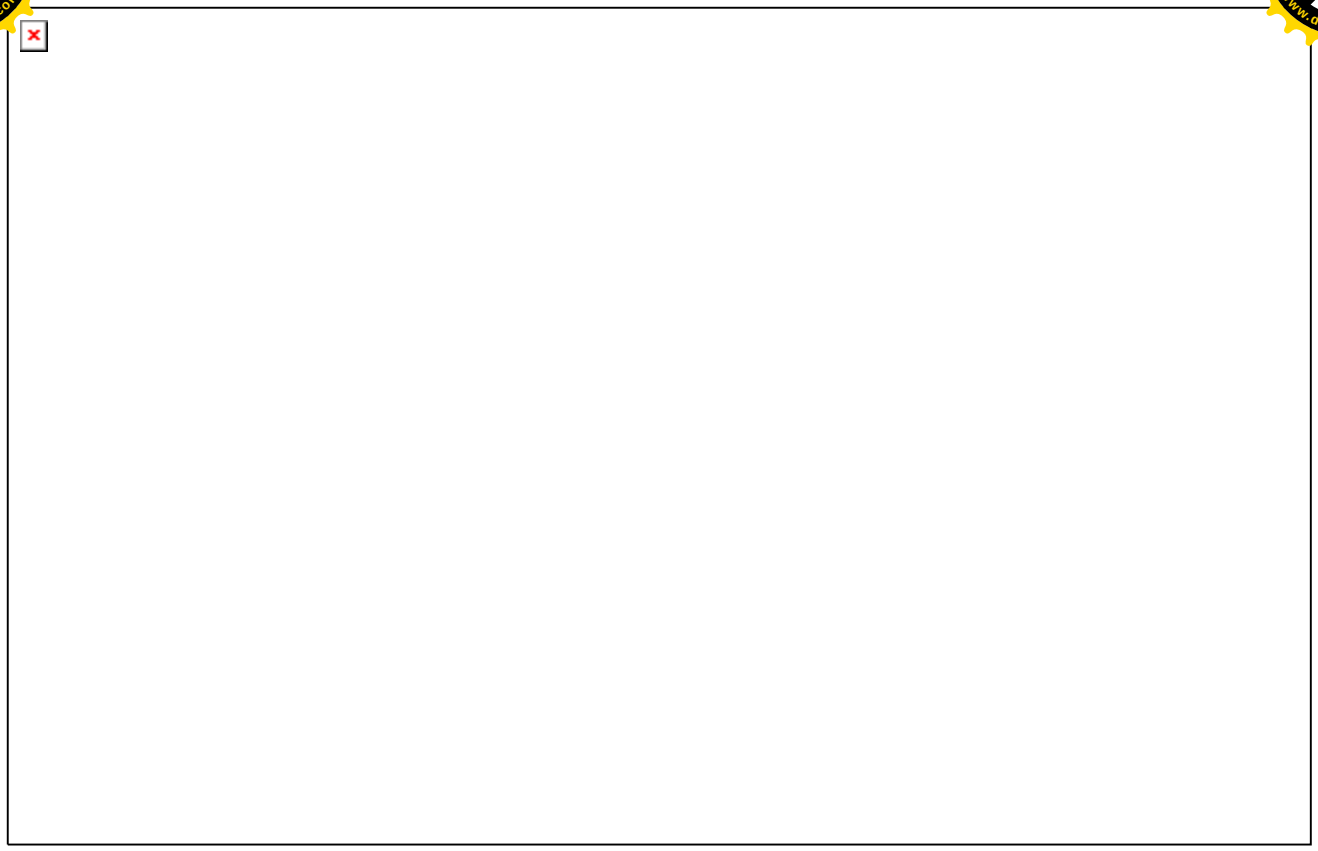
Мал. 2.45. Головна форма застосування «Електронний екзаменатор»

На малюнку 2.46 зображене діалогове вікно, що з'являється при виборі пункту **Питання 1** з головного меню. Вікно містить питання за курсом неорганічної хімії і варіанти відповідей.



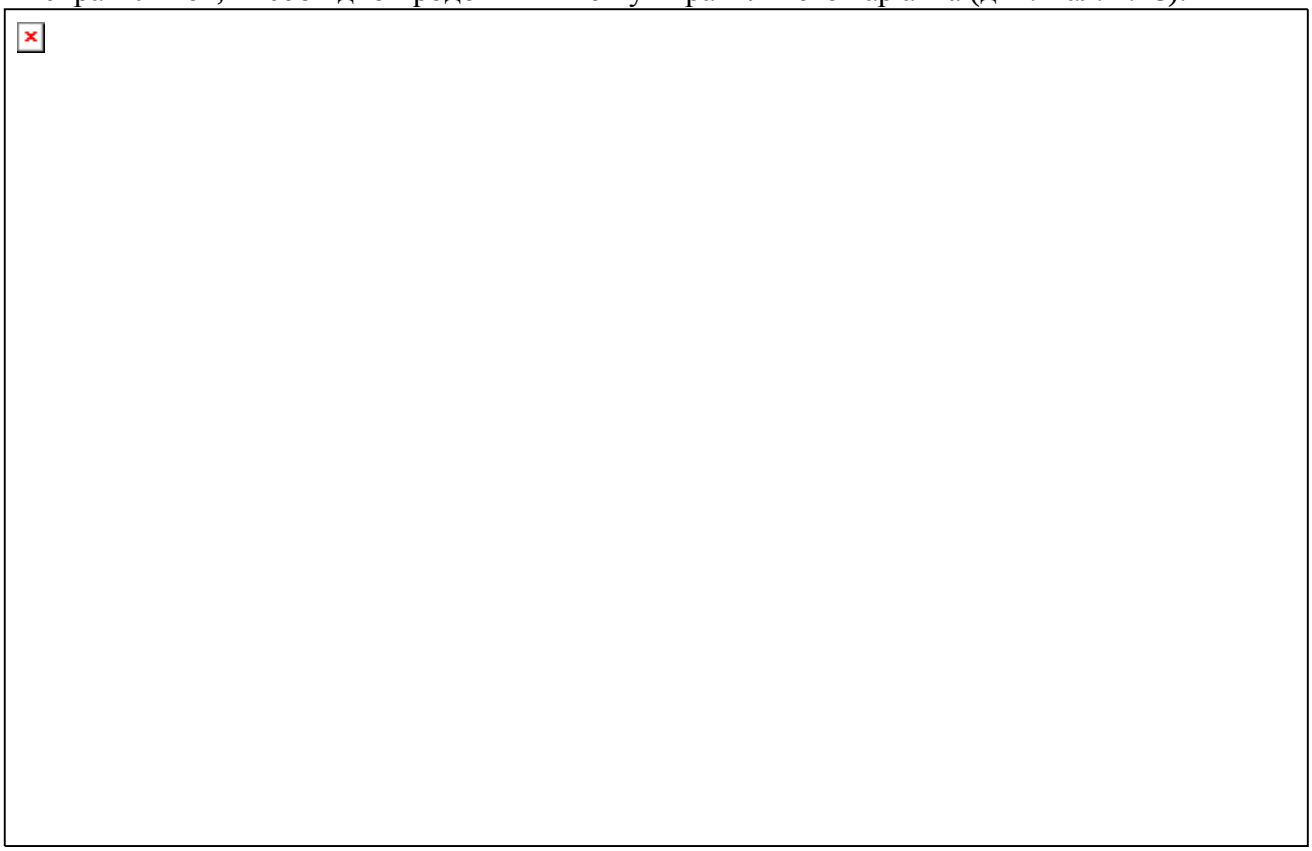
Мал. 2.46. Форма для проведення контролю знань

Розташований у нижній частині вікна список перемикачів призначений для вибору правильної відповіді. Якщо обраний правильний варіант відповіді, то список перемикачів стає недоступним, а на розташованій праворуч панелі з'являється напис – «Правильно» (див. мал. 2.47).



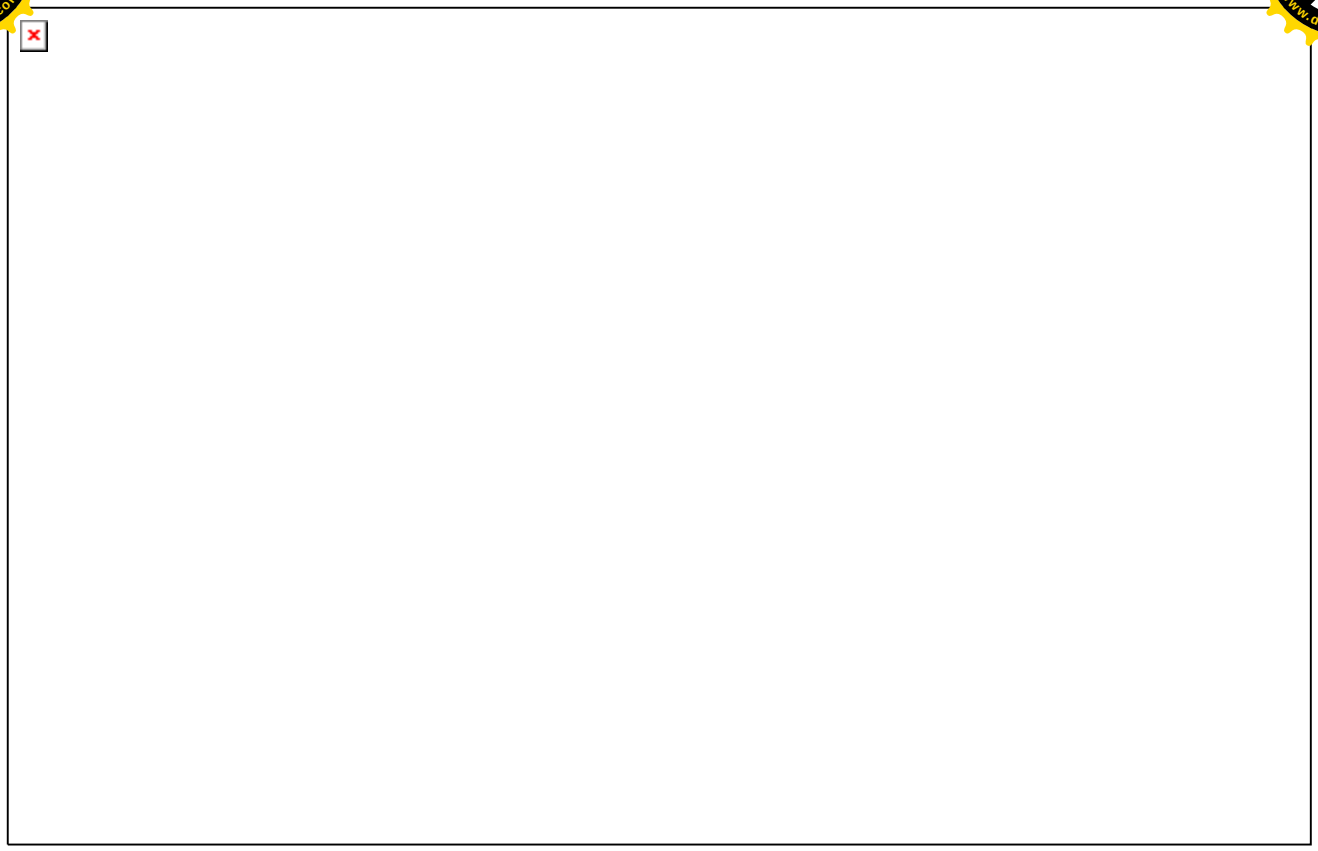
Мал. 2.47. Вид форми для проведення контролю знань після правильної відповіді

Якщо обраний невірний варіант відповіді, то на панелі з'являється напис – «Неправильно», і необхідно продовжити пошук правильного варіанта (див. мал. 2.48).



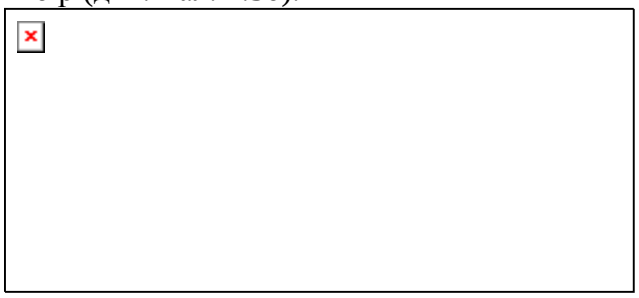
Мал. 2.48. Вид форми для проведення контролю знань після неправильної відповіді

Пункти **Питання 2** і **Питання 3** у головному меню призначені для виведення в модальному режимі того ж вікна, але яке містить інші питання (див. мал. 2.48 і 2.49).



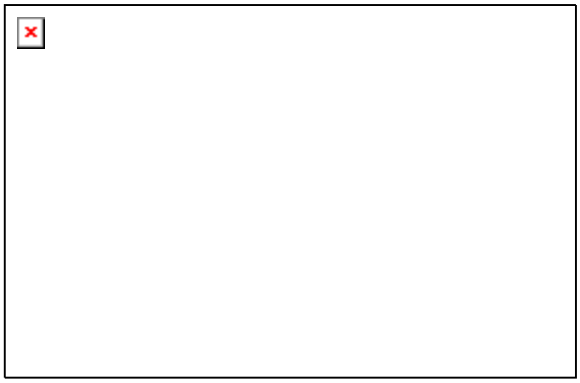
Мал. 2.49. Використання форми для відображення різних питань

Закрити вікно можна або за допомогою кнопки **Закрити**, розташованої в нижній частині вікна, або за допомогою стандартної кнопки закриття, розташованої в заголовку. В другому випадку наше застосування виводить на екран діалогове вікно, у якому необхідно підтвердити зроблений вибір (див. мал. 2.50).



Мал. 2.50. Діалогове вікно для підтвердження закриття форми проведення контролю

Пункт **Статистика** головного меню виводить на екран у модальному режимі однойменне вікно (див. мал. 2.51), у якому відображаються результати проведеного контролю.



Мал. 2.51. Діалогове вікно «Статистика»

Для завершення роботи застосування «Електронний екзаменатор» необхідно вибрати пункт **Вихід** головного меню або скористатися кнопкою закриття в заголовку головного вікна застосування.

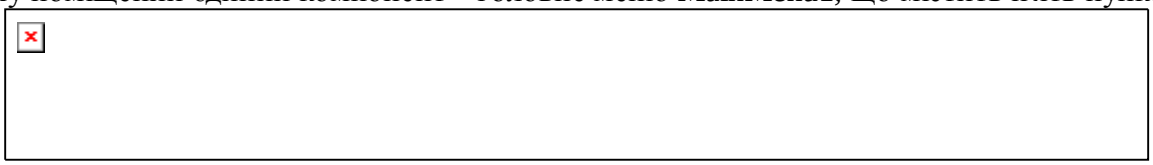
2.11.4.2. Структура застосування «Електронний екзаменатор». Головна форма застосування

Застосування «Електронний екзаменатор» містить головну форму Fmain і дві відображувані в модальному режимі форми – Fpyt і Fstat. Призначення цих форм і відповідних їм модулів приведено в таблиці 2.11.

Таблиця 2.11. Призначення форм і модулів застосування «Електронний екзаменатор»

Форма	**Модуль**	**Призначення**
Fmain	Umain	Головна форма. Модуль містить оброблювачи пунктів головного меню
Fpyt	Upyt	Форма для проведення контролю знань. Модуль містить методи для відображення документів MS Word, що містять питання, і аналізу відповідей
Fstat	Ustat	Форма для відображення статистичної інформації. Відповідний модуль містить метод для формування тексту звіту

На малюнку 2.52 зображена головна форма Fmain на етапі проектування. Призначення форми Fmain – виведення на екран форм Fpyt і Fstat у модальному режимі. З цією метою на форму поміщений єдиний компонент – головне меню MainMenu1, що містить п'ять пунктів.



Мал. 2.52. Головна форма Fmain на етапі проектування

Значення властивостей форми Fmain, змінені на етапі проектування за допомогою Інспектора Об'єктів, приведені в таблиці 2.12.

Таблиця 2.12. Значення властивостей форми Fmain

Властивість	**Значення**
Align	alTop
BorderIcons	[biSystemMenu]
BorderStyle	bsSingle
Caption	‘Електроний екзаменатор’
Height	95
Name	‘Fmain’
Width	715

Привласнивши властивості Align значення alTop, ми забезпечимо розташування головного вікна у верхній частині екрана дисплея. Значення, встановлене для властивості BorderIcons, визначає наявність у заголовку вікна тільки однієї кнопки – стандартної кнопки закриття вікна. Помітимо, що значення для властивостей Caption і Name взяті в лапки тільки для того, щоб підкреслити їхній тип – string, а при роботі з Інспектором Об'єктів лапки не потрібні.

Перейдемо до написання програмного коду.

У програмі маються дані, що використовуються різними формами. Це відбувається в наступних ситуаціях:

1) в оброблювачах пунктів меню **Питання 1** – **Питання 3** головної форми необхідно вказувати номер питання, що використовується у формі Fpyt при виведенні її на екран;

2) підрахунок кількості правильних і неправильних відповідей здійснюється в одному з методів форми Fpyt, а видача статистичних даних – у формі Fstat.

Для вирішення зазначених двох задач додамо в клас форми TFmain дві властивості – Number і Count:

```
type
  Mas = array [1..3,1..2] of integer;
  TFmain = class(TForm)
……..
  private
    FCount:Mas;
    FNumber:integer;
    function GetCount(i,j:integer):integer;
    procedure SetCount(i,j:integer; const value:integer);
  public
    property Count[i,j:integer]:integer read GetCount write SetCount;
    property Number:integer read FNumber write FNumber;
  end;
```

Властивість цілого типу Number містить номер питання, що повинний бути поміщений у форму Fpyt у момент її виведення на екран.

Властивість-масив (інша назва – масив властивостей) Count містить статистичні дані контролю знань. Перший індекс і визначає номер питання і може приймати значення від 1 до 3. Другий індекс може приймати значення 1 чи 2. Якщо Count[i,1] = 0, то на і-те питання не було дано правильної відповіді. І навпаки, якщо Count[i,1] = 1, то на і-ий питання була дана правильна відповідь. Значення Count[i,2] визначає кількість невірних відповідей на і-те питання.

Помітимо, що при оголошенні властивості-масиву існує два обмеження:
1) після імені властивості-масиву необхідно вказувати список індексних параметрів, взяті у квадратні дужки;
2) після зарезервованих слів read і write не можна вказувати ім'я поля, а потрібно вказувати імена методів читання і запису.

У розділі реалізації модуля Umain метод читання GetCount і метод запису SetCount визначені в такий спосіб:

```
function TFmain.GetCount(i,j:integer):integer;
begin
  result := FCount[i,j]
end;

procedure TFmain.SetCount(i,j:integer; const value:integer);
begin
  FCount[i,j] := value
end;
```

Крім розглянутих, у модуль Umain доданий опис ще шести методів. П'ять з них призначені для обробки пунктів головного меню. Відповідність між пунктами головного меню і їхніми оброблювачами приведено в таблиці 2.13. Імена оброблювачів сгенерированы Delphi, і при бажанні їх можна замінити більш осмисленими.

Таблиця 2.13. Відповідність між пунктами головного меню і їхніми оброблювачами

Пункт головного меню застосування	**Метод**
Питання 1	N11Click
Питання 2	N21Click
Питання 3	N31Click

PDF-XChange Click to buy NOW! www.docu-track.com

Статистика	N1Click
Вихід	N2Click

Метод N11Click призначений для виведення на екран форми Fpyt у модальному режимі:

```
Number :=1;
Fpyt.showmodal;
```

Перед виведенням форми Fpyt на екран у властивісті Number форми Fmain міститься номер потрібного питання. Методи N21Click і N31Click діють аналогічно і відрізняються від попереднього тільки номером питання, що поміщається у властивість Number.

Метод N1Click виводить на екран у модальному режимі форму Fstat. Створення і відображення форми Fstat здійснюється за допомогою наступного програмного коду :

```
if not Assigned(Fstat) then
  Fstat := TFstat.Create(Self);
Fstat.showmodal;
```

Використовувана у фрагменті функція Assigned визначена в такий спосіб:

function Assigned(const P): Boolean;

Параметр P повинний містити посилання на вказівник чи змінну процедурного типу. Функція повертає значення False, якщо P дорівнює nil, і True в інших випадках. Дія функції еквівалентна обчисленню значення виразу P<> nil, якщо P – вказівник, чи виразу @P <> nil, якщо P – змінна процедурного типу.

Приведений вище програмний код обумовлений тим, що частина форм, що входять у проект, створюється автоматично при запуску проекту, а інші створюються програмно. Визначити спосіб створення форми можна за допомогою діалогового вікна Project Options (мал. 2.53), що з'являється при виборі команди Project|Options головного меню Delphi. При додаванні нової форми в проект її ім'я, як правило, попадає в список автоматично створюваних форм (Auto-create forms). Для того щоб перемістити її в список доступних форм (Available forms) необхідно скористатися кнопкою зі стрілкою вправо. Кнопка зі стрілкою вліво дозволяє виконати переміщення форми в протилежному напрямку. Кнопки з двома стрілками виконують переміщення усіх форм одночасно в заданому напрямку.

Очевидно, що автоматично варто створювати головну форму і форму Fpyt, без яких неможлива робота всього застосування. Форму Fstat потрібно створювати в міру необхідності.

Мал. 2.53. Автоматично створювані і доступні форми проекту

Таким чином, процес створення діалогового вікна може бути описаний так: оскільки діалогове вікно Fstat не створюється автоматично, то для його створення в програмі необхідно викликати конструктор Create класу форми, а перед цим потрібно перевірити за допомогою функції Assigned, що форма не була створена раніше.

Метод N2Click закриває головну форму і, тим самим завершує роботу застосування:

```
Close
```

Метод FormCreate виконується в момент створення форми Fmain і може бути використаний для присвоєння початкових значень:

```
for i := 1 to 3 do
  for j := 1 to 2 do
    Count[i,j] := 0;
```

Приведемо повний текст сформованого модуля Umain.

Текст модуля Umain.pas.

```
unit Umain;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms,  Dialogs, Menus;

type
  Mas = array [1..3,1..2] of integer;
  TFmain = class(TForm)
    MainMenu1: TMainMenu;
    N11: TMenuItem;
    N21: TMenuItem;
    N31: TMenuItem;
    N1: TMenuItem;
    N2: TMenuItem;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

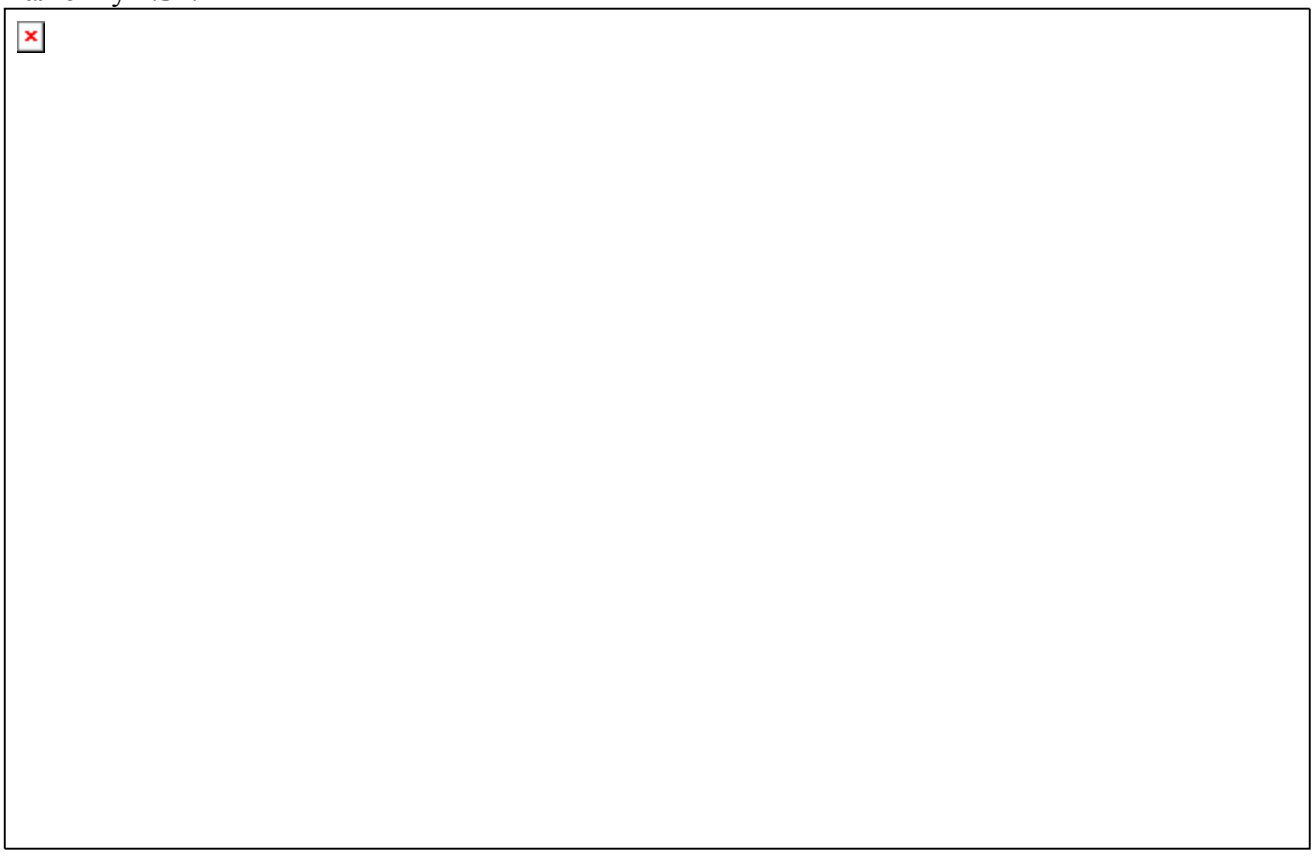
```
    procedure N11Click(Sender: TObject);
    procedure N21Click(Sender: TObject);
    procedure N31Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure N1Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
  private
    FCount:Mas;
    FNumber:integer;
    function GetCount(i,j:integer):integer;
    procedure SetCount(i,j:integer; const value:integer);
  public
    property Count[i,j:integer]:integer read GetCount write SetCount;
    property Number:integer read FNumber write FNumber;
  end;

var
  Fmain: TFmain;

implementation

uses Upyt, Ustat;

{$R *.dfm}

function TFmain.GetCount(i,j:integer):integer;
begin
  result := FCount[i,j]
end;

procedure TFmain.SetCount(i,j:integer; const value:integer);
begin
  FCount[i,j] := value
end;

procedure TFmain.N11Click(Sender: TObject);
begin
  Number :=1;
  Fpyt.showmodal;
end;

procedure TFmain.N21Click(Sender: TObject);
begin
  Number :=2;
  Fpyt.showmodal;
end;

procedure TFmain.N31Click(Sender: TObject);
begin
  Number :=3;
  Fpyt.showmodal;
end;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
procedure TFmain.FormCreate(Sender: TObject);
var i,j:integer;
begin
  for i := 1 to 3 do
    for j := 1 to 2 do
      Count[i,j] := 0;
end;

procedure TFmain.N1Click(Sender: TObject);
begin
  if not Assigned(Fstat) then
    Fstat := TFstat.Create(Self);
  Fstat.showmodal;
end;

procedure TFmain.N2Click(Sender: TObject);
begin
  Close
end;

end.
```

2.11.4.3. Проведення контролю знань

Контроль знань у застосуванні «Електронний екзаменатор» здійснюється за допомогою форми Fpyt. Ця форма виводиться на екран дисплея при виборі пунктів **Питання 1** – **Питання 3** головного меню. У залежності від номера питання на формі відображається уміст відповідного документа Word. Вид форми Fpyt на етапі проектування зображений на малюнку 2.54.



Мал. 2.54. Форма Fpyt на етапі проектування

Для додавання в проект нової форми необхідно виконати команду головного меню File|New|Form. Для того щоб у модулі Umain можна було звертатися до методів і властивостей класу форми TFpyt, необхідно переключитися на форму Fmain, використовуючи комбінацію клавіш Shift+F12, і виконати команду головного меню File|Use Unit. У результаті в операторі uses, розташованому в розділі implementation модуля Umain, додасться посилання на модуль Upyt.

Сказане вище щодо створення форми Fpyt і підключення модуля Upyt до модуля Umain відноситься також до форми Fstat і модулю Ustat, що будуть розглядатися в наступному пункті. Крім того, оскільки в модулях Upyt і Ustat будуть використовуватися властивості класу форми Fmain, те до цих модулів у свою чергу повинен бути підключений модуль Umain описаним вище способом.

Опис розташованих на формі Fpyt компонентів приведено в таблиці 2.14.

Таблиця 2.14. Опис компонентів, розташованих на формі Fpyt

Компонент	**Призначення**
Button1	Кнопка закриття вікна
OleContainer1	Відображає документ MS Word
Panel1	Групує усі компоненти на формі, крім кнопки Button1
Panel2	Відображає текст «Правильно» або «Неправильно»
RadioGroup1	Список перемикачів для вибору правильного варіанта відповіді

У таблиці 2.15 приведені змінені на етапі проектування значення властивостей форми Fpyt і компонентів, розташованих на ній.

Таблиця 2.15. Значення властивостей форми Fpyt і компонентів, розташованих на ній

Компонент	**Властивість**	**Значення**
Button1	Caption	‘Закрити’
	Font.Size	10
OleContainer1	Align	alTop
	AutoActivate	aaManual
	AutoVerbMenu	False
	Ctl3D	False
Panel1	Align	alTop
	Caption	‘’
Panel2	BevelInner	bvSpace
	BevelOuter	bvLowered
	Caption	‘’
RadioGroup1	Caption	‘Виберіть правильний варіант відповіді’
	Columns	5
	Align	alLeft
Fpyt	BorderIcons	[biSystemMenu]
	BorderStyle	bsDialog
	Caption	‘Неорганічна хімія’
	Name	‘Fpyt’
	Position	poScreenCenter

У клас форми TFpyt додано чотири методи:

1) FormShow;
2) RadioGroup1Click;
3) Button1Click;
4) FormCloseQuery.

PDF-XChange Click to buy NOW! www.docu-track.com

Розглянемо їх докладніше.

Метод FormShow виконується перед кожною появою форми Fpyt на екрані. Призначення методу – настроювання зовнішнього вигляду форми в залежності від конкретної ситуації. При виведенні форми Fpyt на екран у панелі перемикачів RadioGroup1 не повинно бути обраної кнопки, а панель Panel2 не повинна містити текст:

```
RadioGroup1.ItemIndex := -1;
Panel2.Caption := '';
```

Якщо на обране питання вже була дана правильна відповідь, то компонент RadioGroup1 буде недоступний, у противному випадку можна продовжити пошук правильної відповіді:

```
if Fmain.Count[Fmain.Number,1] = 1
  then
  RadioGroup1.Enabled := false
  else
  RadioGroup1.Enabled := true;
```

Для відображення на формі потрібного документа Word необхідно уміти формувати повне ім'я файлу. Умовимося для визначеності, що використовувані документи Word знаходяться в тій же папці, де і виконуваний файл нашого проекту (тобто файл з розширенням exe). Для рішення зазначеної задачі нам знадобляться функції ExtractFilePath і ParamStr, тому приведемо їхній опис:

function ExtractFilePath(const FileName: string): string;
Повертає шлях до файлу, включаючи ім'я диска і слеш перед ім'ям файлу, вирізаний з повного імені файлу, що міститься в параметрі FileName.

function ParamStr(Index: Integer): string;
Функція повертає параметр командного рядка, що відповідає значенню параметра Index. Наприклад, якщо параметр Index дорівнює трьом, то буде повернутий третій параметр, зазначений після імені виконуваного файлу програми. Якщо параметр Index дорівнює нулю, то функція поверне повне ім'я виконуваного файлу програми, що включає шлях і власне ім'я файлу. Для нашого приклада повне ім'я файлу має вид:

```
D:\MyProject\ModalForms\examenator.exe .
```

Нижче приведений фрагмент, у якому в залежності від номера обраного питання (властивість Number класу Fmain) формується ім'я документа Word, а потім цей документ міститься в компонент OleContainer1:

```
s := ExtractFilePath(ParamStr(0));
case Fmain.Number of
  1:OleContainer1.CreateObjectFromFile(s+'zapyt1.doc',false);
  2:OleContainer1.CreateObjectFromFile(s+'zapyt2.doc',false);
  3:OleContainer1.CreateObjectFromFile(s+'zapyt3.doc',false);
end;
```

У методі RadioGroup1Click здійснюється аналіз зробленої відповіді. Кожному питанню відповідає свій номер правильної відповіді. Так, наприклад, для першого питання правильною буде третя відповідь (RadioGroup1.ItemIndex = 2), а для другого питання правильною буде четверта відповідь (RadioGroup1.ItemIndex = 3). Якщо вибрана правильна відповідь, то:

1) панель перемикачів стає недоступною;
2) у панелі Panel2 з'являється текст 'Правильно';
3) елементу властивості-масиву Count, розташованому в рядку, номер якого дорівнює номеру питання, і першому стовпці, привласнюється значення 1.

Якщо вибрана неправильна відповідь, то:

PDF-XChange Click to buy NOW! www.docu-track.com

1) у панелі Panel2 з'являється текст 'Неправильно';
2) елемент властивості-масиву Count, розташований у рядку, номер якого дорівнює номеру питання, і другому стовпці, збільшує своє значення на 1, тобто на 1 збільшилася кількість неправильних відповідей на дане питання:

```
n := Fmain.Number;
m := RadioGroup1.ItemIndex;
if (n = 1)and(m = 2) or
   (n = 2)and(m = 3) or
   (n = 3)and(m = 4)
then
  begin
  RadioGroup1.Enabled := false;
  Panel2.Caption := 'Правильно';
  Fmain.Count[n,1] := 1;
  end
else
  begin
  Panel2.Caption := 'Неправильно';
  Fmain.Count[n,2] := Fmain.Count[n,2]+1;
  end;
```

Метод Button1Click призначений для закриття форми Fpyt, виведеної на екран дисплея в модальному режимі:

```
ModalResult := 1;
```

Метод FormCloseQuery виконується при закритті форми Fpyt. У цьому методі аналізується, якою кнопкою було закрите вікно. При аналізі використовується функція MessageBox. Приведемо її опис.

function MessageBox(const Text, Caption: PChar; Flags: Longint = MB_OK): Integer;
Функція MessageBox є методом глобального об'єкта Application класу TApplication. Цей об'єкт створюється автоматично для кожного застосування в момент його запуску і инкапсулирует у собі властивості і методи програми як такий. Функція MessageBox відображає діалогове вікно з заданими кнопками, повідомленням і заголовком і дозволяє проаналізувати відповідь користувача. Параметр Text містить текст виведеного повідомлення. У параметрі Caption задається текст заголовка вікна. Зовнішній вигляд діалогового вікна визначається за допомогою прапорів, заданих у параметрі Flags. Нижче приведена відповідність між прапорами і відображуваними в діалоговому вікні кнопками:

Прапор	**Кнопки**
MB_ABORTRETRYIGNORE	Abort (Стоп), Retry (Повтор) і Ignore (Пропустити)
MB_OK	OK. Цей прапор прийнятий за замовчуванням
MB_OKCANCEL	OK і Cancel (Скасування)
MB_RETRYCANCEL	Retry (Повтор) і Cancel (Скасування)
MB_YESNO	Yes (Так) і No (Немає)
MB_YESNOCANCEL	Yes (Так), No (Немає) і Cancel (Скасування)

Наступна група прапорів визначає вид піктограми в діалоговому вікні:

Прапор	**Піктограма**
MB_ICONEXCLAMATION, MB_ICONWARNING	Знак оклику – зауваження чи попередження
MB_ICONINFORMATION, MB_ICONASTERISK	Буква і у колі – підтвердження
MB_ICONQUESTION	Знак питання – очікується відповідь
MB_ICONSTOP,	Хрест у червоному колі – помилка чи заборона

PDF-XChange Click to buy NOW! www.docu-track.com

MB_ICONERROR,
MB_ICONHAND

Значенням параметра Flags, найчастіше, є сума вищевказаних прапорів, по одному з кожної групи, наприклад MB_ABORTRETRYIGNORE+MB_ICONEXCLAMATION. Функція повертає нуль, якщо їй не вистачає пам'яті для створення діалогового вікна, у противному випадку повертається один з наступних результатів:

Результат, що повертається	**Числовий еквівалент**	**Значення**
IDOK	1	Обрана кнопка OK
IDCANCEL	2	Обрана кнопка Cancel (Скасування)
IDABORT	3	Обрана кнопка Abort (Стоп)
IDRETRY	4	Обрана кнопка Retry (Повтор)
IDIGNORE	5	Обрана кнопка Ignore (Пропустити)
IDYES	6	Обрана кнопка Yes (Так)
IDNO	7	Обрана кнопка No (Немає)

Тепер сформулюємо алгоритм, реалізований у методі FormCloseQuery. Якщо форма Fpyt була закрита кнопкою **Закрити** (Button1), то відповідно до методу Button1Click властивість ModalResult форми прийме значення 1. Якщо ж форма Fpyt була закрита за допомогою стандартної кнопки закриття вікна, розташованої в заголовку вікна, то за замовчуванням властивості ModalResult буде привласнене значення 2. В другому випадку з'явиться діалогове вікно, у якому треба підтвердити закриття чи відмовитися від нього:

```
if ModalResult = 2 then
  case Application.MessageBox('Ви дійсно хочете закрити вікно?',
                'Підтвердите ваш вибір',
                MB_YESNO+MB_ICONQUESTION) of
    IDYES:;
    IDNO:CanClose := false;
  end;
```

Помітимо, що розглянутий метод FormCloseQuery є скоріше прикладом деякого шаблона, оскільки в ньому відсутня обробка даних. Звичайно такий підхід використовується для збереження даних чи інших допоміжних операцій при закритті форм застосування.

Приведемо повний текст сформованого модуля Upyt.

Текст модуля Upyt.pas.

```
unit Upyt;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, OleCtnrs, ExtCtrls, StdCtrls;

type
  TFpyt = class(TForm)
    Panel1: TPanel;
    OleContainer1: TOleContainer;
    Button1: TButton;
    RadioGroup1: TRadioGroup;
    Panel2: TPanel;
    procedure RadioGroup1Click(Sender: TObject);
```

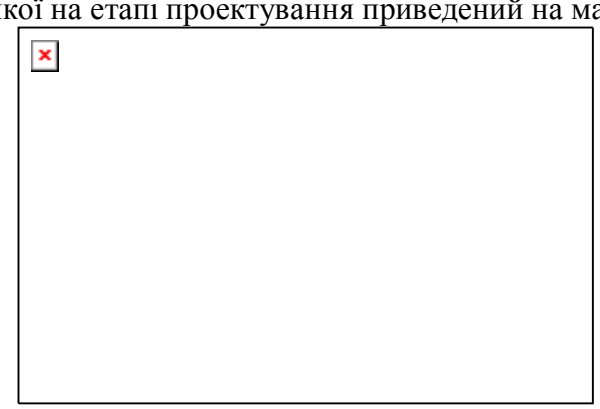
PDF-XChange
Click to buy NOW!
www.docu-track.com

```
    procedure Button1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Fpyt: TFpyt;

implementation

uses Umain;

{$R *.dfm}

procedure TFpyt.FormShow(Sender: TObject);
var s:string;
begin
  RadioGroup1.ItemIndex := -1;
  Panel2.Caption := '';
  if Fmain.Count[Fmain.Number,1] = 1
    then
    RadioGroup1.Enabled := false
    else
    RadioGroup1.Enabled := true;
  s := ExtractFilePath(ParamStr(0));
  case Fmain.Number of
    1:OleContainer1.CreateObjectFromFile(s+'zapyt1.doc',false);
    2:OleContainer1.CreateObjectFromFile(s+'zapyt2.doc',false);
    3:OleContainer1.CreateObjectFromFile(s+'zapyt3.doc',false);
  end;
end;

procedure TFpyt.RadioGroup1Click(Sender: TObject);
var n,m:integer;
begin
   n := Fmain.Number;
   m := RadioGroup1.ItemIndex;
   if (n = 1)and(m = 2) or
      (n = 2)and(m = 3) or
      (n = 3)and(m = 4)
   then
     begin
     RadioGroup1.Enabled := false;
     Panel2.Caption := 'Правильно';
     Fmain.Count[n,1] := 1;
     end
   else
     begin
```

```
    Panel2.Caption := 'Неправильно';
    Fmain.Count[n,2] := Fmain.Count[n,2]+1;
    end;
end;

procedure TFpyt.Button1Click(Sender: TObject);
begin
  ModalResult := 1;
end;

procedure TFpyt.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if ModalResult = 2 then
    case Application.MessageBox('Ви дійсно хочете закрити вікно?',
                  'Підтвердите ваш вибір',
                  MB_YESNO+MB_ICONQUESTION) of
     IDYES:;
     IDNO:CanClose := false;
   end;
end;

end.
```

2.11.4.4. Ведення статистичного обліку

Вікно **Статистика** з'являється на екрані при виборі в головному меню однойменного пункту. У цьому вікні відображається інформація з кожного питання: чи була дана правильна відповідь і кількість невдалих спроб. Вікно **Статистика** реалізоване за допомогою форми Fstat, зовнішній вигляд якої на етапі проектування приведений на малюнку 2.55.



Мал. 2.55. Форма Fstat на етапі проектування

Перелік компонентів, розташованих на формі Fstat і їхнє призначення приведені в таблиці 2.16.

Таблиця 2.16. Опис компонентів, розташованих на формі Fstat

Компонент	**Призначення**
Button1	Кнопка закриття вікна
Memo1	Відображає результати проведення контролю знань

У таблиці 2.17 приведені значення властивостей компонентів і форми Fstat, які були змінені на етапі проектування.

Таблиця 2.17. Значення властивостей форми Fstat і компонентів, розташованих на ній

Компонент	**Властивість**	**Значення**
Button1	Caption	‘Закрити’

	Font.Size	10
Memo1	Align	alTop
	ReadOnly	True
Fstat	BorderIcons	[biSystemMenu]
	BorderStyle	bsDialog
	Caption	‘Статистика’
	Name	‘Fstat’
	Position	poScreenCenter

У клас форми TFstat додані два методи – FormShow і Button1Click. Розглянемо їх докладніше.

Метод FormShow виконується з кожною появою форми Fstat на екрані дисплея і призначений для виведення статистичної інформації. Нагадаємо, що необхідна інформація про результати контролю знань зберігається у властивості-масиві Count, визначеному в класі Fmain. Алгоритм роботи методу FormShow досить простій:

1) очищаємо багаторядковий редактор Memo1 і організуємо цикл із параметром і для перебору наявних трьох питань;

2) якщо Count[i,1] містить одиницю, те це означає, що на і-те питання була дана правильна відповідь; у редактор Memo1 додається рядок з повідомленням, у якому використовується інформація про кількість неправильних відповідей, що міститься в Count[i,2];

3) якщо Count[i,1] не містить одиницю (початковим значенням є нуль), то на і-те питання не була дана правильна відповідь, і в редактор Memo1 додається рядок з відповідним повідомленням:

```
Memo1.Clear;
for i := 1 to 3 do
  if Fmain.Count[i,1] = 1
  then
    Memo1.Lines.Add('Відповідь на питання №'+IntToStr(i)+' дана з '+
    IntToStr(Fmain.Count[i,2]+1)+'-ої спроби')
  else
    Memo1.Lines.Add('Відповідь на '+IntToStr(i)+' питання не дана')
```

Метод Button1Click обробляє щиглик по кнопці Button1 і призначений для закриття вікна:

```
ModalResult := 1;
```

Приведемо повний текст сформованого модуля Ustat.

Текст модуля Ustat.pas.

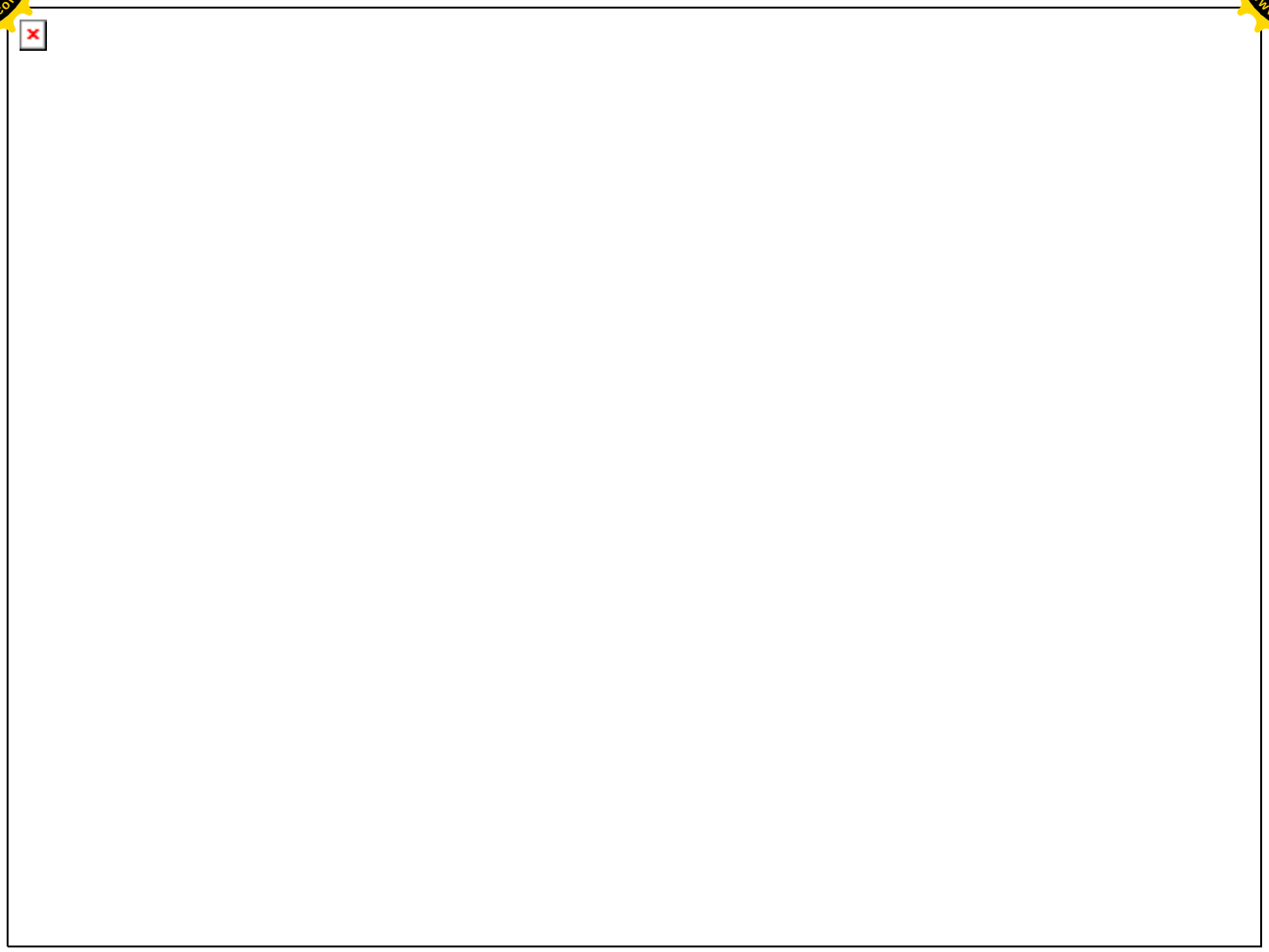
```
unit Ustat;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, StdCtrls;

type
  TFstat = class(TForm)
    Memo1: TMemo;
    Button1: TButton;
    procedure FormShow(Sender: TObject);
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
  public
    { Public declarations }
  end;

var
  Fstat: TFstat;

implementation

uses Umain;

{$R *.dfm}

procedure TFstat.FormShow(Sender: TObject);
var i:integer;
begin
  Memo1.Clear;
  for i := 1 to 3 do
    if Fmain.Count[i,1] = 1
    then
      Memo1.Lines.Add('Відповідь на питання №'+IntToStr(i)+' дана з '+
      IntToStr(Fmain.Count[i,2]+1)+'-ої спроби')
    else
      Memo1.Lines.Add('Відповідь на '+IntToStr(i)+' питання не дана')
end;

procedure TFstat.Button1Click(Sender: TObject);
begin
  ModalResult := 1;
end;

end.
```

2.11.5. Приклад створення SDI-застосування

У пункті 2.11.4 ми розглянули застосування, усі вікна якого, крім головного, виводяться в модальному режимі. Тобто, якщо відкрите деяке діалогове вікно, те друге вікно можна відкрити тільки після закриття першого. Іноді зручніше працювати відразу з декількома відкритими вікнами. На малюнку 2.56 зображене застосування «Електронний екзаменатор», виконане в стилі SDI.



Мал. 2.56. Застосування «Електронний екзаменатор», виконане в стилі SDI

У верхній частині екрана розташовується головне вікно. Воно містить лінійку меню і призначено для виведення на екран інших вікон. Вибираючи пункти **Питання1** – **Питання3** меню можна відкрити усі вікна з контрольними питаннями. Переключення між вікнами здійснюється або за допомогою миші, або вибором відповідного пункту меню. Положення будь-якого вікна може бути змінено довільним образом. Вікно **Статистика** будемо як і раніше виводити в модальному режимі, підкреслюючи, таким чином, умовність розбивки на стилі. Кожне вікно може бути закрите за допомогою наявних на ньому кнопок. При закритті головного вікна всі немодальні вікна закриваються автоматично.

Розглянемо відмінності, що з'явилися при модифікації застосування.

Візьмемо за основу раніше створене застосування. З цією метою скопіюємо його в нову папку, наприклад D:\myprog\SDI.

Внесемо невелику зміну у властивості форм. Установимо властивість Positions для форм Fpyt і Fmain рівною poDefault. Тим самим ми одержимо можливість керувати положенням форм на екрані програмно.

Виконавши команду Project|Options головного меню Delphi, відкриємо вікно опцій проекту. Перенесемо форму Fpyt з панелі Auto-create forms на панель Available forms (мал. 2.57). Це означає, що екземпляри форми Fpyt ми будемо створювати програмно, у міру необхідності.

Мал. 2.57. Автоматично створювані і доступні форми застосування «Електронний екзаменатор», виконаного в стилі SDI

Тепер перейдемо до розгляду відмінностей у програмному коді. Почнемо з модуля Umain.

Основна відмінність від попереднього варіанта застосування полягає в тім, що нам необхідно відображати на екрані дисплея кілька форм із контрольними питаннями одночасно. Це означає, що ми повинні створити кілька об'єктів – екземплярів класу TFpyt. З цією метою об’явимо необхідні об'єкти в розділі implementation:

```
var Fpyt1,Fpyt2,Fpyt3:TFpyt;
```

Створення і виведення на екран форм-об'єктів здійснюється як і раніше в оброблювачах N11Click, N21Click, N31Click, зв'язаних відповідно з пунктами **Питання 1**, **Питання 2** і **Питання 3** головного меню. Так, наприклад, створення і виведення на екран у немодальному режимі форми-об'єкта Fpyt1 виконується в такий спосіб:

```
if not Assigned(Fpyt1) then
  begin
  Fmain.Tag :=1;
  Fpyt1 := TFpyt.Create(Self);
  end;
Fpyt1.Show;
```

Об'єкти Fpyt2 і Fpyt3 створюються і відображаються аналогічно. Властивість цілого типу Tag визначено у всіх спадкоємцях базового класу TComponent і надається програмісту для використання їм по особистому розсуді. Очевидно, що ця властивість цілком еквівалентна властивості Number, визначеної нами раніше в класі форми Fmain. Можна вважати, що властивість Number було введено в навчальних цілях для ілюстрації додавання властивості простого типу в клас форми. У застосуванні, що розглядається, ми вилучимо властивість Number із класу TFmain, і будемо використовувати властивість Tag, передаючи через неї об'єктам класу Fpyt номер контрольного питання.

Приведемо повний текст модифікованого модуля Umain.

Текст модуля Umain.pas.

```
unit Umain;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, Menus, StdCtrls;

type
  Mas = array [1..3,1..2] of integer;
  TFmain = class(TForm)
    MainMenu1: TMainMenu;
    N11: TMenuItem;
    N21: TMenuItem;
    N31: TMenuItem;
    N1: TMenuItem;
    N2: TMenuItem;
    procedure N11Click(Sender: TObject);
    procedure N21Click(Sender: TObject);
    procedure N31Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure N1Click(Sender: TObject);
    procedure N2Click(Sender: TObject);
  private
    FCount:Mas;
    function GetCount(i,j:integer):integer;
    procedure SetCount(i,j:integer; const value:integer);
  public
    property Count[i,j:integer]:integer read GetCount write SetCount;
  end;

var
  Fmain: TFmain;

implementation

uses Upyt, Ustat;

{$R *.dfm}

var Fpyt1,Fpyt2,Fpyt3:TFpyt;

function TFmain.GetCount(i,j:integer):integer;
begin
  result := FCount[i,j]
end;

procedure TFmain.SetCount(i,j:integer; const value:integer);
begin
  FCount[i,j] := value
end;

procedure TFmain.N11Click(Sender: TObject);
begin
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
  if not Assigned(Fpyt1) then
    begin
    Tag :=1;
    Fpyt1 := TFpyt.Create(Self);
    end;
  Fpyt1.Show;
end;

procedure TFmain.N21Click(Sender: TObject);
begin
  if not Assigned(Fpyt2) then
    begin
    Tag :=2;
    Fpyt2 := TFpyt.Create(Self);
    end;
  Fpyt2.Show;
end;

procedure TFmain.N31Click(Sender: TObject);
begin
  if not Assigned(Fpyt3) then
    begin
    Tag :=3;
    Fpyt3 := TFpyt.Create(Self);
    end;
  Fpyt3.Show;
end;

procedure TFmain.FormCreate(Sender: TObject);
var i,j:integer;
begin
  for i := 1 to 3 do
    for j := 1 to 2 do
      Count[i,j] := 0;
end;

procedure TFmain.N1Click(Sender: TObject);
begin
  if not Assigned(Fstat) then
    Fstat := TFstat.Create(Self);
  Fstat.showmodal;
end;

procedure TFmain.N2Click(Sender: TObject);
begin
  Close
end;

end.
```

У модулі Upyt доданий метод FormCreate, що виконується при створенні чергової форми-об'єкта класу TFpyt. У цьому методі в залежності від номера контрольного питання,

що міститься у властивості Tag форми Fmain, для об'єктів Fpyt1, Fpyt2 і Fpyt3, об’явлених у модулі Umain, визначаються наступні характеристики:
1) документ Word, що поміщається в компонент OleContainer1;
2) положення форми на екрані – властивості Left і Top;
3) номер контрольного питання – міститься у властивісті Tag класу TFpyt.
Крім цього в методі FormCreate властивості ModalResult класу TFpyt привласнюється в якості початкового нульове значення. Ця властивість буде використана надалі при закритті форми:

```
ModalResult := 0;
s := ExtractFilePath(ParamStr(0));
case Fmain.Tag of
 1:begin
  OleContainer1.CreateObjectFromFile(s+'zapyt1.doc',false);
  Left := 100;
  Top := 90;
  Tag :=1;
  end;
 2:begin
  OleContainer1.CreateObjectFromFile(s+'zapyt2.doc',false);
  Left := 150;
  Top := 145;
  Tag :=2;
  end;
 3:begin
  OleContainer1.CreateObjectFromFile(s+'zapyt3.doc',false);
  Left := 200;
  Top := 200;
  Tag :=3;
  end;
```

У методі FormShow залишилися тільки загальні для всіх трьох об'єктів – Fpyt1, Fpyt2 і Fpyt3 – операції настроювання зовнішнього вигляду форм. У порівнянні з попереднім застосуванням замість властивості Number класу TFmain використовується властивість Tag класу TFpyt:

```
RadioGroup1.ItemIndex := -1;
Panel2.Caption := '';
if Fmain.Count[Tag,1] = 1
 then
 RadioGroup1.Enabled := false
 else
 RadioGroup1.Enabled := true;
```

У методі RadioGroup1Click також замість властивості Number класу TFmain використовується властивість Tag класу TFpyt:

```
……
 n := Tag;
 m := RadioGroup1.ItemIndex;
 if (n = 1)and(m = 2) or
   (n = 2)and(m = 3) or
   (n = 3)and(m = 4)
 then
……...
```

В оброблювачі Button1Click для того, щоб зробити форму невидимою використовується метод Hide. Властивості ModalResult привласнюється значення 1 для того, щоб надалі можна було визначити, якою кнопкою закрите вікно:

```
ModalResult := 1;
Hide;
```

Метод FormCloseQuery також піддався змінам. На відміну від попереднього застосування закриття немодального вікна за допомогою стандартної кнопки закриття, розташованої в заголовку вікна, не змінює значення властивості ModalResult. Тому в розглянутому застосуванні ця властивість приймає значення 1, якщо форма закрита кнопкою **Закрити**, чи залишається рівної 0, якщо використовується стандартна кнопка закриття. Діалогове вікно для підтвердження закриття вікна з'являється в другому випадку:

```
if ModalResult <> 1 then
  case Application.MessageBox('Ви дійсно хочете закрити вікно?',
                'Підтвердите ваш вибір',
                MB_YESNO+MB_ICONQUESTION) of
    IDYES:;
    IDNO:CanClose := false;
  end;
```

Приведемо повний текст модифікованого модуля Upyt.

Текст модуля Upyt.pas.

```
unit Upyt;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, OleCtnrs, ExtCtrls, StdCtrls;

type
  TFpyt = class(TForm)
    Panel1: TPanel;
    OleContainer1: TOleContainer;
    Button1: TButton;
    RadioGroup1: TRadioGroup;
    Panel2: TPanel;
    procedure RadioGroup1Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormCloseQuery(Sender: TObject; var CanClose: Boolean);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Fpyt: TFpyt;

implementation
```

```
uses Umain;

{$R *.dfm}

procedure TFpyt.FormCreate(Sender: TObject);
var s:string;
begin
  ModalResult := 0;
  s := ExtractFilePath(ParamStr(0));
  case Fmain.Tag of
    1:begin
      OleContainer1.CreateObjectFromFile(s+'zapyt1.doc',false);
      Left := 100;
      Top := 90;
      Tag :=1;
      end;
    2:begin
      OleContainer1.CreateObjectFromFile(s+'zapyt2.doc',false);
      Left := 150;
      Top := 145;
      Tag :=2;
      end;
    3:begin
      OleContainer1.CreateObjectFromFile(s+'zapyt3.doc',false);
      Left := 200;
      Top := 200;
      Tag :=3;
      end;
  end;
end;

procedure TFpyt.FormShow(Sender: TObject);
begin
  RadioGroup1.ItemIndex := -1;
  Panel2.Caption := '';
  if Fmain.Count[Tag,1] = 1
    then
    RadioGroup1.Enabled := false
    else
    RadioGroup1.Enabled := true;
end;

procedure TFpyt.RadioGroup1Click(Sender: TObject);
var n,m:integer;
begin
   n := Tag;
   m := RadioGroup1.ItemIndex;
   if (n = 1)and(m = 2) or
      (n = 2)and(m = 3) or
      (n = 3)and(m = 4)
   then
     begin
```

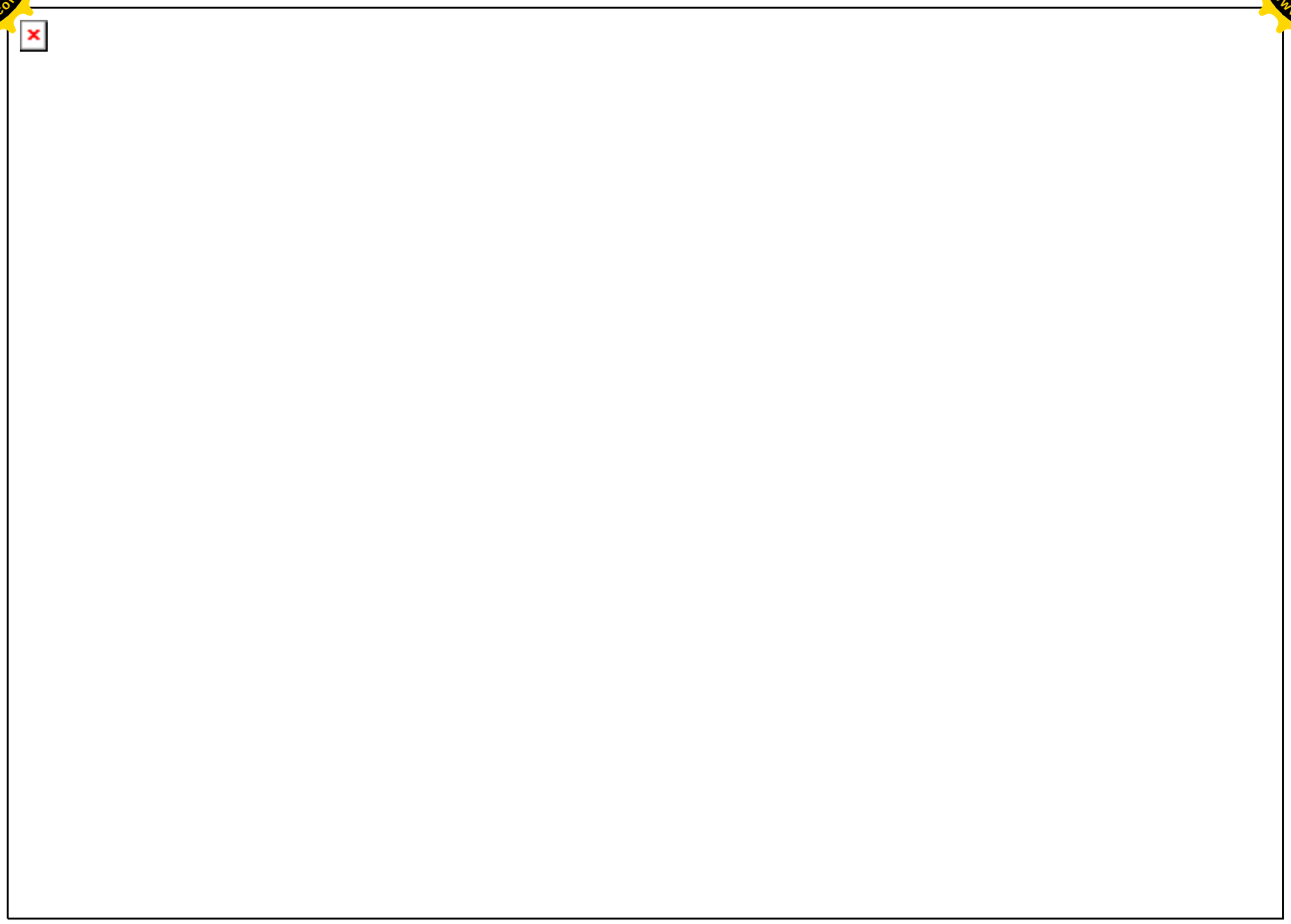
PDF-XChange
Click to buy NOW!
www.docu-track.com

```
    RadioGroup1.Enabled := false;
    Panel2.Caption := 'Правильно';
    Fmain.Count[n,1] := 1;
    end
  else
    begin
    Panel2.Caption := 'Неправильно';
    Fmain.Count[n,2] := Fmain.Count[n,2]+1;
    end;
end;

procedure TFpyt.Button1Click(Sender: TObject);
begin
  ModalResult := 1;
  Hide;
end;

procedure TFpyt.FormCloseQuery(Sender: TObject; var CanClose: Boolean);
begin
  if ModalResult <> 1 then
    case Application.MessageBox('Ви дійсно хочете закрити вікно?',
                  'Підтвердите ваш вибір',
                  MB_YESNO+MB_ICONQUESTION) of
      IDYES:;
      IDNO:CanClose := false;
    end;

end;

end.
```

Третій модуль – Ustat – у порівнянні з попереднім застосуванням не змінився.

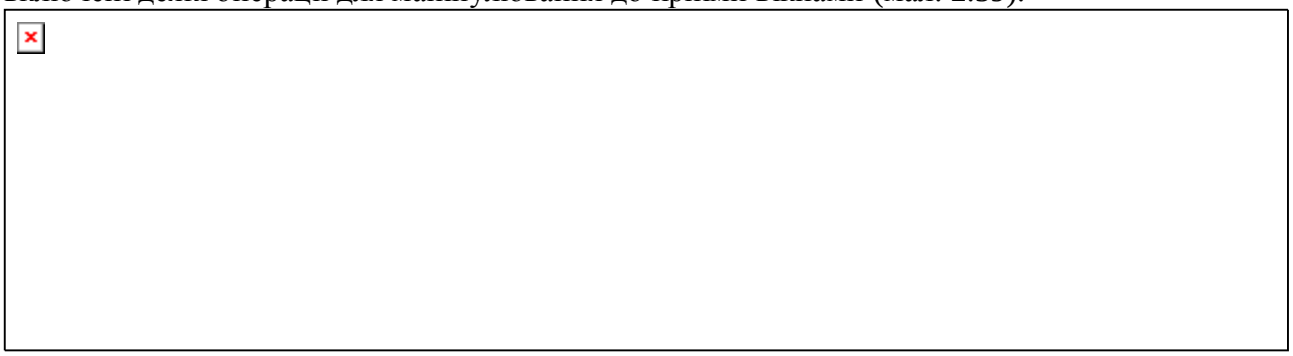
2.11.6. Приклад створення MDI-застосування

На малюнку 2.58 зображене застосування «Електронний екзаменатор», виконане в стилі MDI.



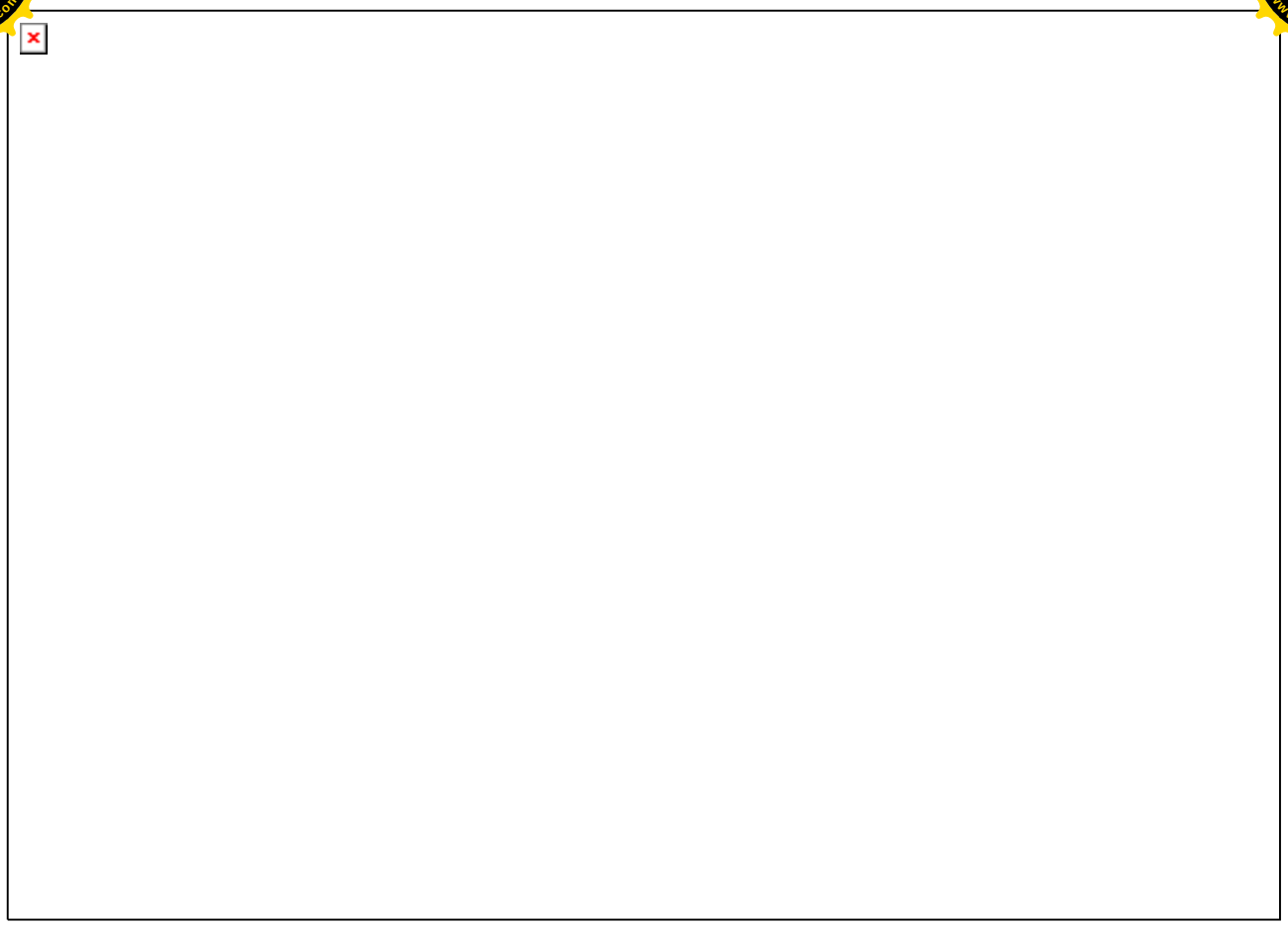
Мал. 2.58. Застосування «Електронний екзаменатор», виконане в стилі MDI

Характерною рисою MDI-застосувань є можливість керування дочірніми формами з батьківської форми. Для ілюстрації цієї можливості в головне меню нашого застосування включені деякі операції для маніпулювання дочірніми вікнами (мал. 2.59).



Мал. 2.59. Операції для роботи з дочірніми вікнами

Розташування дочірніх вікон каскадом приведено на малюнку 2.58, а горизонтальне розташування зображене на малюнку 2.60.



Мал. 2.60. Горизонтальне розташування дочірніх вікон

Для того щоб краще усвідомити відмінності між SDI і MDI стилями, створимо нове застосування шляхом внесення змін у застосування з попереднього пункту. З цією метою скопіюємо папку D:\myprog\SDI у нову папку, наприклад D:\myprog\MDI.

Насамперед, необхідно внести деякі зміни у властивості форм Fmain і Fpyt (див. табл. 2.18).

Таблиця 2.18. Значення властивостей форм Fmain і Fpyt

Форма	**Властивість**	**Значення**
Fmain	Align	alNone
	BorderIcons	[biSystemMenu,biMinimize,biMaximize]
	BorderStyle	bsSizeable
	FormStyle	fsMDIForm
	Height	600
	Width	900
	WindowState	wsMaximized
Fpyt	BorderIcons	[biSystemMenu,biMinimize,biMaximize]
	BorderStyle	bsSizeable
	FormStyle	fsMDIChild
	Position	poDefaultPosOnly

Центральною властивістю, що забезпечує створення MDI-стилю, є FormStyle. У батьківській формі вона повинна мати значення fsMDIForm, а в дочірній – fsMDIChild. Для властивостей Align, BorderIcons і BorderStyle форми Fmain установлені значення, прийняті за замовчуванням. З огляду на змінені значення властивостей WindowState, Height і Width, поводження форми Fmain можна описати в такий спосіб:

1) при запуску застосування батьківська форма займає весь екран;

2) крім стандартної кнопки закриття вікна в заголовку батьківської форми присутні кнопки мінімізації і максимізації;
3) розміри форми Fmain, що знаходиться в не максимізованому стані, можна змінювати за допомогою миші.

Для форми Fpyt також залишена можливість максимізації і мінімізації. Помітимо, що при збільшенні розмірів форми розташовані на ній компоненти також повинні змінювати своє положення і розміри. Одним зі способів забезпечення цього є «прикріплення» компонентів до країв форми чи групуючих компонентів. Нагадаємо, що для рішення цієї задачі варто використовувати властивість Anchors.

Присвоєння властивості Position значення poDefaultPosOnly є змушеним заходом, оскільки дочірні форми при виведенні на екран в інших режимах злегка «обрізуються» знизу.

Розглянемо зміни, зроблені в програмному коді. Почнемо, як звичайно, з модуля Umain.

У класі форми TFmain доданий властивість-масив IsCreate з елементами логічного типу:

```
property IsCreate[i:integer]:Boolean read GetIsCreate write SetIsCreate;
```

Якщо IsCreate[i] містить true, то це означає, що форма з i-тим питанням була створена і відображається на екрані в даний момент. Якщо ж IsCreate[i] містить false, то форма з i-тим питанням закрита. На відміну від попередніх двох прикладів закриття форми в даному випадку буде означати її знищення і звільнення займаної нею пам'яті.

Тепер перш ніж створювати форму з i-тим питанням варто перевірити значення, що міститься в IsCreate[i]. Наприклад, для форми з першим питанням маємо:

```
if not IsCreate[1] then
  begin
  Tag :=1;
  Fpyt := TFpyt.Create(Self);
  end;
Fpyt.Show;
```

Форми з другим і третім питанням створюються аналогічно.

У клас форми TFmain додано кілька методів, що упорядковують розташування дочірніх вікон і змінюють активну форму серед них. Ці методи обробляють пункти головного меню, зображені на малюнку 2.59. Керувати дочірніми вікнами з батьківського вікна досить просто:

1) зробити наступну дочірню форму активної:

```
Next;
```

2) зробити попередню дочірню форму активної:

```
Previous;
```

3) упорядкувати форми горизонтально (див. мал. 2.60):

```
TileMode := tbHorizontal;
Tile;
```

4) упорядкувати форми вертикально:

```
TileMode := tbVertical;
Tile;
```

5) упорядкувати форми каскадом (мал. 2.58):

```
Cascade;
k := MDIChildCount;
for i := 1 to k do
  begin
  ActiveMDIChild.Height := 487;
  Previous;
  end;
```

В останньому фрагменті після розташування форм каскадом, їхня висота «вручну» установлюється рівної 487 пикселей. Це зв'язано з тим, що розмір клієнтської області батьківської форми може бути недостатнім для розміщення дочірніх форм, тому розміри останніх можуть бути автоматично зменшені.

Приведемо повний текст модифікованого модуля Umain.

Текст модуля Umain.pas.

```
unit Umain;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, Menus, StdCtrls;

type
  Mas = array [1..3,1..2] of integer;
  Vec = array [1..3] of Boolean;
  TFmain = class(TForm)
    MainMenu1: TMainMenu;
    N11: TMenuItem;
    N21: TMenuItem;
    N31: TMenuItem;
    N1: TMenuItem;
    N2: TMenuItem;
    N3: TMenuItem;
    N7: TMenuItem;
    N8: TMenuItem;
    N9: TMenuItem;
    N4: TMenuItem;
    N5: TMenuItem;
    N6: TMenuItem;
    N10: TMenuItem;
    procedure N11Click(Sender: TObject);
    procedure N21Click(Sender: TObject);
    procedure N31Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure N1Click(Sender: TObject);
    procedure N3Click(Sender: TObject);
    procedure N5Click(Sender: TObject);
    procedure N6Click(Sender: TObject);
    procedure N8Click(Sender: TObject);
    procedure N9Click(Sender: TObject);
    procedure N10Click(Sender: TObject);
  private
    FCount:Mas;
    FIsCreate:Vec;
    function GetCount(i,j:integer):integer;
    procedure SetCount(i,j:integer; const value:integer);
    function GetIsCreate(i:integer):Boolean;
    procedure SetIsCreate(i:integer; const value:Boolean);
  public
    property Count[i,j:integer]:integer read GetCount write SetCount;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
    property IsCreate[i:integer]:Boolean read GetIsCreate write SetIsCreate;
  end;

var
  Fmain: TFmain;

implementation

uses Upyt, Ustat;

{$R *.dfm}

function TFmain.GetCount(i,j:integer):integer;
begin
  result := FCount[i,j]
end;

procedure TFmain.SetCount(i,j:integer; const value:integer);
begin
  FCount[i,j] := value
end;

function TFmain.GetIsCreate(i:integer):Boolean;
begin
  result := FIsCreate[i]
end;

procedure TFmain.SetIsCreate(i:integer; const value:Boolean);
begin
  FIsCreate[i] := value
end;

procedure TFmain.N11Click(Sender: TObject);
begin
  if not IsCreate[1] then
    begin
    Tag :=1;
    Fpyt := TFpyt.Create(Self);
    end;
  Fpyt.Show;
end;

procedure TFmain.N21Click(Sender: TObject);
begin
  if not IsCreate[2] then
    begin
    Tag :=2;
    Fpyt := TFpyt.Create(Self);
    end;
  Fpyt.Show;
end;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
procedure TFmain.N31Click(Sender: TObject);
begin
  if not IsCreate[3] then
    begin
    Tag :=3;
    Fpyt := TFpyt.Create(Self);
    end;
  Fpyt.Show;
end;

procedure TFmain.FormCreate(Sender: TObject);
var i,j:integer;
begin
  for i := 1 to 3 do
  begin
    IsCreate[i] := false;
    for j := 1 to 2 do
      Count[i,j] := 0;
  end;
end;

procedure TFmain.N1Click(Sender: TObject);
begin
  if not Assigned(Fstat) then
    Fstat := TFstat.Create(Self);
  Fstat.showmodal;
end;

procedure TFmain.N3Click(Sender: TObject);
begin
    Close;
end;

procedure TFmain.N5Click(Sender: TObject);
var i,k:integer;
begin
  Cascade;
  k := MDIChildCount;
  for i := 1 to k do
    begin
    ActiveMDIChild.Height := 487;
    Previous;
    end;
end;

procedure TFmain.N6Click(Sender: TObject);
begin
  TileMode := tbVertical;
  Tile;
end;

procedure TFmain.N8Click(Sender: TObject);
```

PDF-XChange Click to buy NOW! www.docu-track.com

```
begin
  Next;
end;

procedure TFmain.N9Click(Sender: TObject);
begin
  Previous;
end;

procedure TFmain.N10Click(Sender: TObject);
begin
  TileMode := tbHorizontal;
  Tile;
end;

end.
```

У модулі Upyt.pas зміни торкнулися таких операцій як створення і знищення форми-об'єкта класу TFpyt.

Створення форми-об'єкта відбувається за наступним алгоритмом:

1) у властивість Tag форми-об'єкта поміщаємо номер питання, переданий за допомогою властивості Tag батьківської форми;
2) у заголовок форми-об'єкта додаємо номер питання, що міститься в ньому;
3) елементу з номером Tag властивості-масиву IsCreate батьківської форми привласнюємо значення true; це означає, що дочірня форма, яка містить питання з номером Tag створена і відображається на екрані;
4) поміщаємо в дочірню форму документ Word, який відповідає значенню, що знаходиться в її властивості Tag:

```
Tag := Fmain.Tag;
Caption := Caption + ' - Питання №' + IntToStr(Tag);
Fmain.IsCreate[Tag] := true;
s := ExtractFilePath(ParamStr(0));
case Tag of
  1:OleContainer1.CreateObjectFromFile(s+'zapyt1.doc',false);
  2:OleContainer1.CreateObjectFromFile(s+'zapyt2.doc',false);
  3:OleContainer1.CreateObjectFromFile(s+'zapyt3.doc',false);
end;
```

Закриття дочірньої форми здійснюється за допомогою методу

```
Close;
```

який міститься в оброблювачі Button1Click, або за допомогою стандартної кнопки закриття вікна.

У порівнянні з попередніми двома застосуваннями в модулі Upyt з метою спрощення вилучений оброблювач FormCloseQuery, але зате доданий оброблювач FormClose, у якому вказується, що варто мати на увазі під закриттям дочірньої форми:

```
Fmain.IsCreate[Tag] := false;
Action := caFree;
```

З приведеного коду випливає, що закриття форми означає її знищення і звільнення займаної нею пам'яті. Попередньо елементу з номером Tag властивості-масиву IsCreate батьківської форми привласнюємо значення false. Це означає, що дочірня форма, яка містить питання з номером Tag, знищена.

Приведемо повний текст модифікованого модуля Upyt.

Текст модуля Upyt.pas.

```
unit Upyt;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls,
Forms, Dialogs, OleCtnrs, ExtCtrls, StdCtrls;

type
  TFpyt = class(TForm)
    Panel1: TPanel;
    OleContainer1: TOleContainer;
    Button1: TButton;
    RadioGroup1: TRadioGroup;
    Panel2: TPanel;
    procedure RadioGroup1Click(Sender: TObject);
    procedure Button1Click(Sender: TObject);
    procedure FormShow(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure FormClose(Sender: TObject; var Action: TCloseAction);
  private
     { Private declarations }
  public
    { Public declarations }
  end;

var
  Fpyt: TFpyt;

implementation

uses Umain;

{$R *.dfm}

procedure TFpyt.FormCreate(Sender: TObject);
var s:string;
begin
  Tag := Fmain.Tag;
  Caption := Caption + ' - Питання №' + IntToStr(Tag);
  Fmain.IsCreate[Tag] := true;
  s := ExtractFilePath(ParamStr(0));
  case Tag of
    1:OleContainer1.CreateObjectFromFile(s+'zapyt1.doc',false);
    2:OleContainer1.CreateObjectFromFile(s+'zapyt2.doc',false);
    3:OleContainer1.CreateObjectFromFile(s+'zapyt3.doc',false);
  end;
end;

procedure TFpyt.FormShow(Sender: TObject);
begin
  RadioGroup1.ItemIndex := -1;
```

PDF-XChange
Click to buy NOW!
www.docu-track.com

```
  Panel2.Caption := '';
  if Fmain.Count[Tag,1] = 1
    then
    RadioGroup1.Enabled := false
    else
    RadioGroup1.Enabled := true;
end;

procedure TFpyt.RadioGroup1Click(Sender: TObject);
var n,m:integer;
begin
  n := Tag;
  m := RadioGroup1.ItemIndex;
  if (n = 1)and(m = 2) or
     (n = 2)and(m = 3) or
     (n = 3)and(m = 4)
  then
    begin
    RadioGroup1.Enabled := false;
    Panel2.Caption := 'Правильно';
    Fmain.Count[n,1] := 1;
    end
  else
    begin
    Panel2.Caption := 'Неправильно';
    Fmain.Count[n,2] := Fmain.Count[n,2]+1;
    end;
end;

procedure TFpyt.Button1Click(Sender: TObject);
begin
  Close;
end;

procedure TFpyt.FormClose(Sender: TObject; var Action: TCloseAction);
begin
  Fmain.IsCreate[Tag] := false;
  Action := caFree;
end;

end.
```

Література

1. Архангельский А.Я. Программирование в Delphi 5. – М.: ЗАО «Издательство БИНОМ», 2000. – 1072 с.: ил.
2. Дарахвелидзе П.Г., Марков Е.П. Delphi 4. – СПб.:БХВ – Санкт-Петербург, 1999. – 816 с.: ил.
3. Калверт Ч. Delphi 4. Энциклопедия пользователя: Пер. с англ./ Чарлз Калверт. – К.: Издательство «ДиаСофт», 1998. – 800 с.
4. Кандзюба С.П., Громов В.Н. Delphi 6. Базы данных и приложения. Лекции и упражнения. – К.: Издательство «ДиаСофт», 2001. – 576 с.
5. Конопка Р. Создание оригинальных компонент в среде Delphi: Пер. с англ./ Рэй Конопка. – НИПФ «ДиаСофт Лтд.», 1996. – 512 с.
6. Кэнту М. Delphi 4 для профессионалов. – СПб., Издательство «Питер», 1999. – 1120 с.: ил.
7. Тейксейра С., Пачеко К. Delphi 5. Руководство разработчика, том1. Основные методы и технологии программирования: Пер. с англ.: Уч. пос. – М.: Издательский дом «Вильямс», 2000. – 832 с.: ил. – Парал. тит. англ.
8. Тейксейра С., Пачеко К. Delphi 5. Руководство разработчика, том2. Разработка компонентов и программирование баз данных: Пер. с англ.: Уч. пос. – М.: Издательский дом «Вильямс», 2000. – 992 с.: ил. – Парал. тит. англ.
9. Фаронов В.В. Delphi 5. Руководство программиста.– М.: «Нолидж», 2001. – 880 с.: ил.

Навчальне видання

Алексеєв Михайло Олександрович
Кандзюба Сергій Павлович
Коротенко Леонід Михайлович
Шевцова Ольга Сергіївна

ОСНОВИ ПРОГРАМУВАННЯ

DELPHI 6

Навчальний посібник

Авторська редакція

Підп. до друку 08.04.2013. Формат 30х42/4.
Папір офсетний. Ризографія. Ум. друк. арк.15,2.
Обл.-вид. арк. 15,2. Тираж 5 прим. Зам. № 270 .

Державний ВНЗ «Національний гірничий університет»
Свідоцтво про внесення до Державногог реєстру ДК № 1842 від 11.06.2004.

49005, м. Дніпропетровськ, просп. К. Маркса, 19.