Елементи програмування у середовищі Delphi
Сидоров М.В.-С.

Вступ

Структура проекту

Середовище Delphi містить у собі повний набір візуальних інструментів для швидкої розробки програм (RAD - rapid application development), що підтримує розробку інтерфейсу користувача та підключення до корпоративних баз даних. VCL - бібліотека візуальних компонентів, що містить стандартні об'єкти побудови інтерфейсу користувача, графічні об'єкти, діалоги, об'єкти мультимедіа, об'єкти керування базами даних, об'єкти керування файлами тощо.
Саму розробку програми можна поділити на дві частини: візуальну та алгоритмічну.
У Delphi візуальна частина просто "ліпиться", як скульптор з кусків глини ліпить фігуру. Центром "ліплення", полотном, де будуть розгортатись всі події програми є форма (Form), а вже на формі "ліпимо" компоненти, які є доступними у Delphi.

Найпростіша програма складається з однієї форми, складні – з 2 і більше

Проект програми

У Delphi розробляється проект програми. При цьому автоматично створюється кілька файлів:
project1.dpr – delphi project – саме ядро програми – текстовий файл де описана головна програма, що викликає всі компоненти.
unit1.pas – текстовий файл, де описаний алгоритм всіх подій, що відбуватимуться з елементами (компонентами) форми. Ця частина у Delphi носить назву модуль (unit).
unit1.dfm – файл опису розташування компонентів на полотні формі.

Таким чином, при написанні програми, алгоритмічна частина робота ведеться з pas–файлом. dpr–файл керується автоматично самим проектом при роботі з компонентами інтерфейсу.
Спочатку вважатимемо, що кожній формі відповідає один модуль. Ще раз зауважимо, що у формі створюється інтерфейс, тобто "ліпляться" компоненти, а у модулі описуються події, що можуть відбуватись з цими компонентами

При запуску або створенні нового проекту Delphi автоматично створює pas – файл форми з базовими елементами, необхідними для роботи проекту.

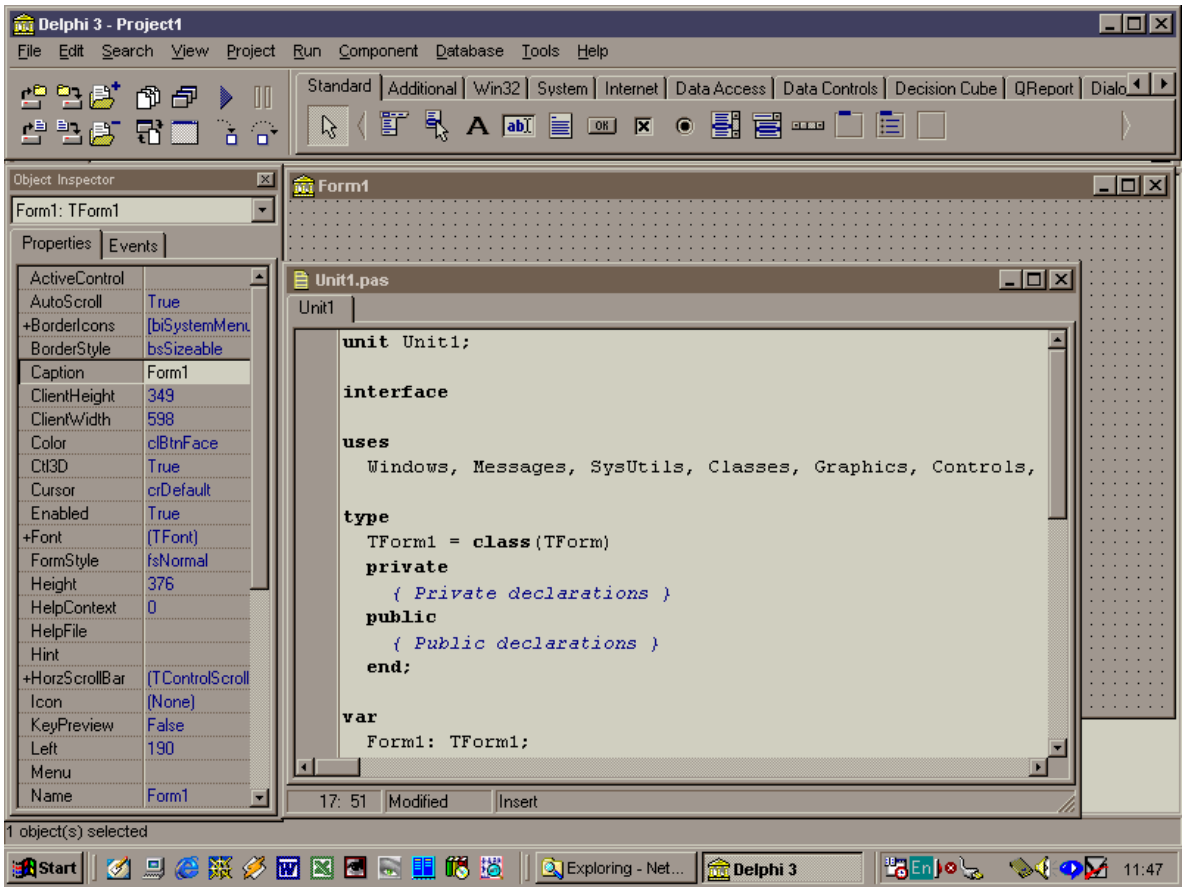
```
unit Unit1;
interface
uses
        Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
type
        TForm1 = class(TForm)
        private
        { Private declarations }
        public
```

```
        { Public declarations }
        end;

var
        Form1: TForm1;

Implementation

        {$R *.DFM}
end.
```

Слово *Unit* говорить про те, що це модуль.
Interface – блок програми, в якому визначаються типи, змінні, константи, процедури та функції, що доступні іншим модулям.
Uses – перераховані всі зовнішні (у даному випадку стандартні) модулі, які необхідні даному модулю в процесі роботи.
Type TForm1 = class(TForm) – створена форма належить до класу Tform. Tform – абстрактний об'єкт, у якому описані всі події та можливості керування всіма елементами форми. Він є спадкоємцем класу Tform, тобто, успадковує його властивості та методи, додаючи до них власні.

Середовище програмування Delphi

Розглянемо тепер середовище програмування Delphi.



Мал. 1

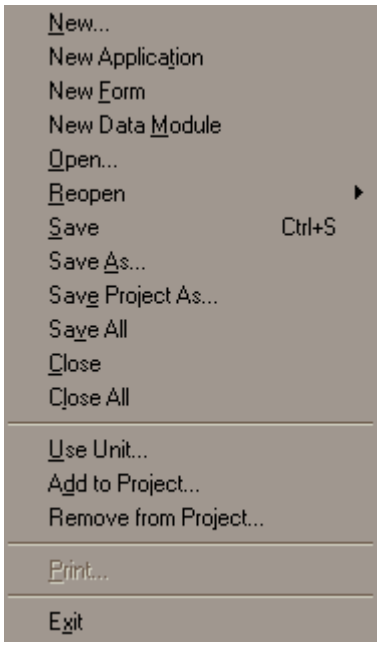
Головне меню середовища Delphi.

Головне меню Delphi розташоване у верхній частині робочого вікна середовища і має вигляд.

У ньому розташовані головні настройки та дії, що пов'язані як з проектом, так і з самим середовищем.
Розглянемо деякі компоненти меню:

File.

Зупинимось спочатку на пункті File.(див. Мал. 2).



Мал. 2

New… – Створити нові елементи: на екрані виводиться перелік елементів, які можна створити. Це нова програма, нова форма, новий модуль тощо.

New Application – Створити нову програму (цей елемент також є і у переліку New…)

New Form – Створити нову форму (цей елемент також є і у переліку New…)

Open… - Відкрити існуючий проект, модуль та ін.

Save – Зберегти зміни у поточному проекті чи модулі. Якщо до цього часу проект чи модуль не зберігався на диск, то автоматично Delphi присвоїть йому ім’я Project1 або Unit1 відповідно.

Save As… - Зберегти проект ти модуль під іншим ім’ям на диск.

Save Project As… - Зберегти проект під іншим ім’ям

Save All - Зберегти всі зміни у модулях та проектах на диск

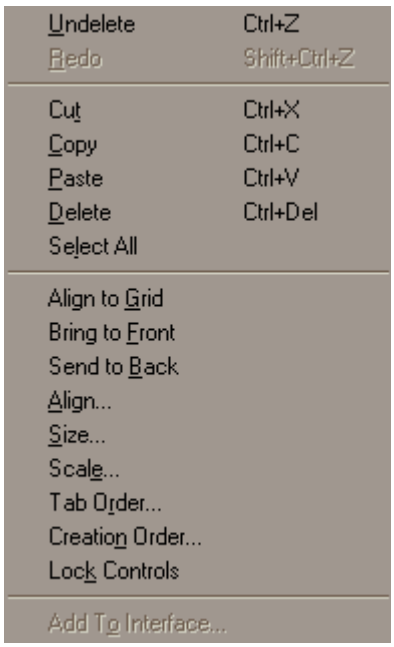
Close – Закрити поточне вікно проекту чи модуля

Close All – Закрити всі вікна з елементами проекту

Exit – Вихід з програми

Edit

Наступним пунктом є Edit:



У цьому пункті наведені головні функції стосовно редагування як форми, так і текстової частини програми.

Undelete – Відмінити видалення елементу або фрагменту тексту.

Redo – Повторити останню дію

Cut – Вирізати фрагмент або об'єкт та зберегти його у буфері обміну. Буфер обміну містить тільки останній занесений фрагмент чи об'єкт

Paste – вставити з буферу обміну

Delete – видалити відмічений фрагмент або об'єкт

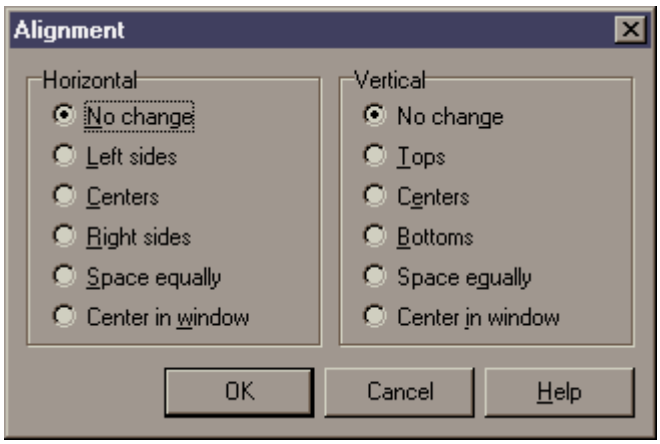
Select All – відмітити все

Align to Grid – перемістити об'єкт до найближчої точки сітки. Параметри сітки встановлюються у пункті Tools| Environment Options.

Bring to Front – Перенести об'єкт на задній план.

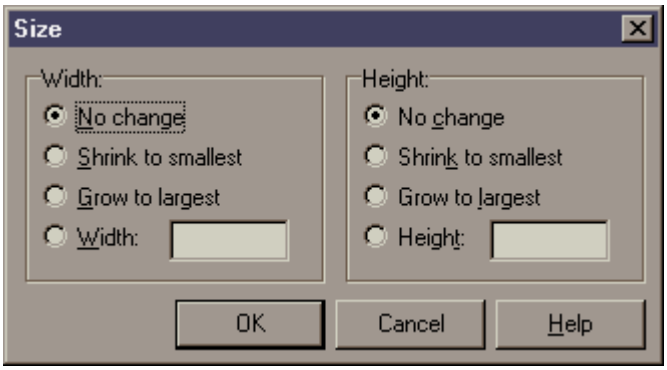
Send to Back – Перенести об'єкт на передній план.

Align… – Вирівняти розташування об'єкту. При виборі цієї функції на екран буде подане діалогове вікно вигляду:



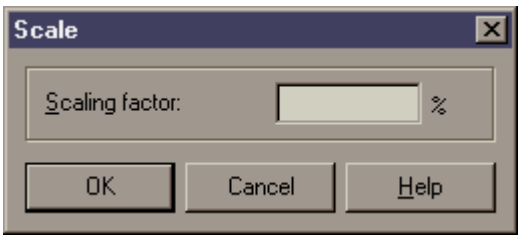
у якому пропонуються різні види вирівнювання об'єктів.

Size… – Встановлення розміру об'єкту на формі. На екран буде подане діалогове вікно

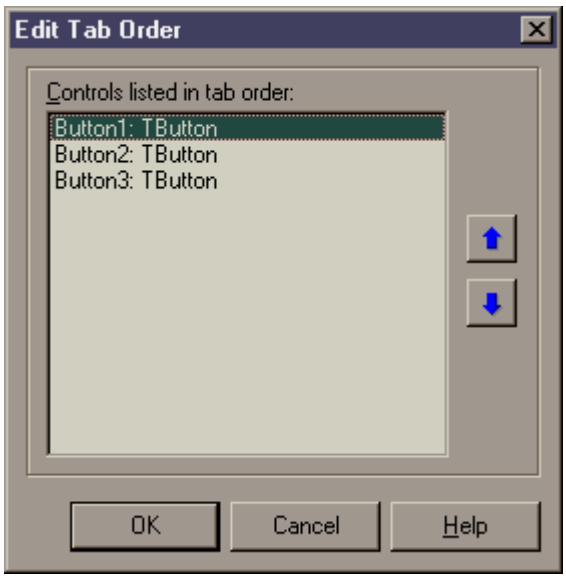


для встановлення параметрів розміру.

Scale… – пропорційна зміна розміру об'єкту (шкалювання). На екран подається вікно для встановлення відсотку зменшення або збільшення розміру.

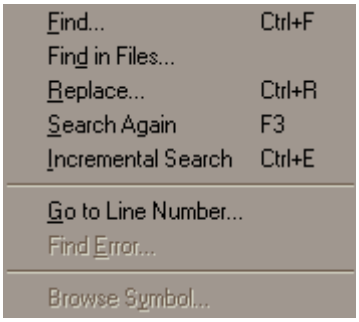


Tab Order… – Встановлення послідовності переходів між керуючими об'єктами (Button, RadioGroup, …) за використання клавіші Tab.



Search

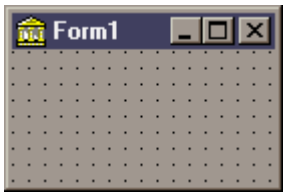
Тут наведено головні функції пошуку та заміни (здебільшого у текстовій /алгоритмічній/ частині проекту)



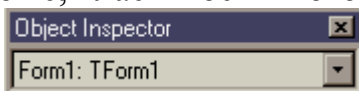
Tab Order… – Встановлення послідовності переходів між керуючими об'єктами (Button, RadioGroup, …) за використання клавіші Tab.

Форма

Далі, звернемо увагу на саме вікно форми

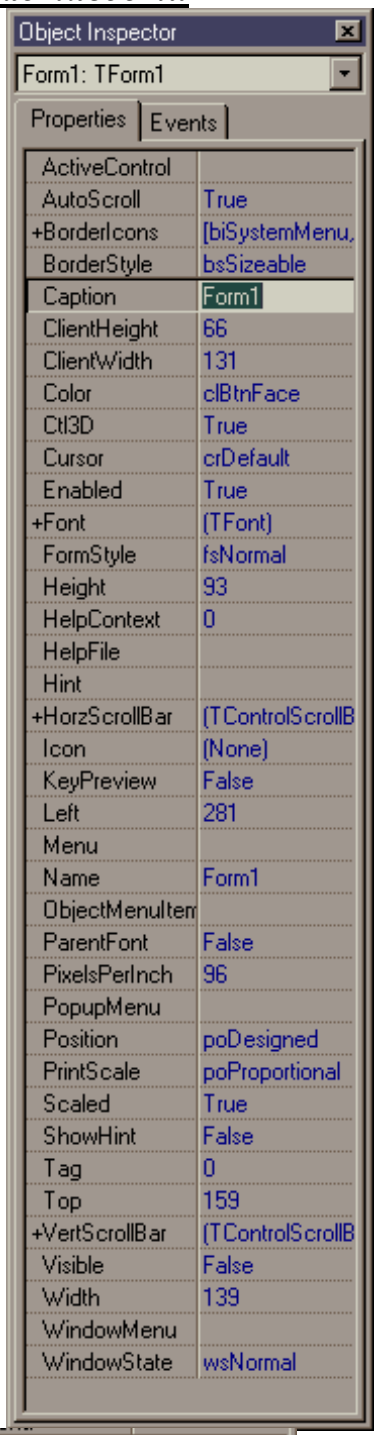


та на її властивості.
Зверху вікна властивостей вказано те, властивості якого об’єкта розглядаються.



означає, що розглядаються властивості об’єкту Form1.

Властивості.

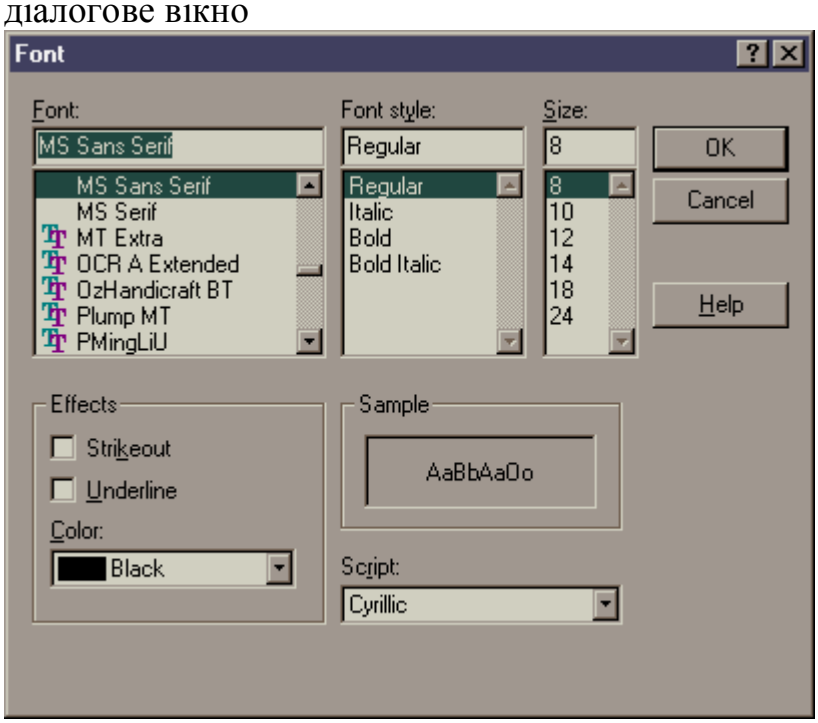


Зупинимось на деяких, найголовніших властивостях.

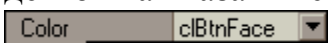
Caption. Це власне назва форми, що буде виводитись на екран у верхній частині (заголовку) форми.

Для вікна форми можна встановити шрифт (Font) та розмір.

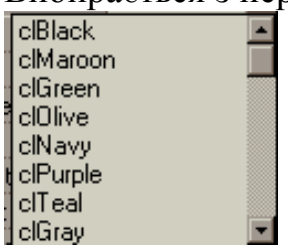
+Font. Встановлення параметрів шрифту. При натисканні на ... на екран виводиться діалогове вікно



де можна вказати всі параметри шрифту.



Color. Колір фону форми. Вибирається з переліку кольорів



Name. Ім’я форми – присвоюється автоматично. Кожен об’єкт повинен мати унікальне ім’я. Програміст може змінити ім’я форми на будь-яке більш зрозуміле та мнемонічне

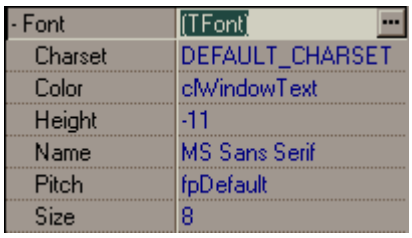
Відмітимо, що багато об’єктів можуть мати однакові параметри властивостей.

Загальні правила використанні властивостей об'єктів у Delphi.

Якщо у переліку властивостей об'єкту зустрічається властивість, у якої зліва є символ +, то це значить, що ця властивість має групу параметрів, до яких можна дістатись двічі клацнувши лівою клавішею мишки у цю властивість.

Наприклад

після подвійного клацання мишкою перетвориться на



Якщо у рядку з властивістю є піктограма ..., то, клацнувши у цю піктограму ми отримаємо діалогове меню по встановленню параметрів даної властивості.

Палітра

Наступним елементом середовища Delphi є палітра, яка складається з 13 закладок.

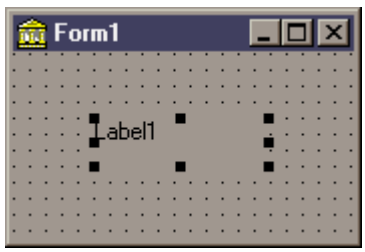
Розглянемо найпростіші елементи (об'єкти) закладки Standard. (див. Мал. 3)

Standard.

Мал. 3

Мітка Label.

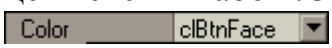
 - Вставити текст, що пояснює призначення елементів на формі (надписи чи мітки) або просто текст. Також може використовуватись для виведення інформації на екран. Досить клацнути мишкою у  і потім клацнути десь на полі форми. На екрані це відобразиться таким чином:



За допомогою мишки цей об'єкт можна переміщати та змінювати його розмір.

Головні Властивості:

- власне сам текст, що виводитиметься. Змінна, що відповідає за цей текст – Label1.Caption має тип string (див далі типи змінних).



- колір фону під текстом.

- настройки шрифту тексту

- якщо значення True (вірно), то текст виводитиметься на екран, якщо False (невірно), то ні.

- ім’я об’єкту (для кожного - унікальне)

Кнопка Button.

- командна кнопка. Використовується для запуску процесів (наприклад обчислень, змін параметрів об’єктів та ін.). На екрані має вигляд

Властивості

- текст, що виводитиметься на кнопці. Змінна, що відповідає за цей текст – Button1.Caption має тип string (див далі типи змінних).

- ім’я об’єкту (для кожного об’єкту - унікальне)

- шрифт тексту на кнопці

Але головним для кнопки є не її зовнішній вигляд, а те, що відбуватиметься при, наприклад, натисканні на неї, чи подвійному натисканні та ін. Зрозуміло, що для кожного об’єкту має бути головна подія.

Головна подія об’єкту.

Введемо поняття головної події об’єкту. Вважатимемо головною подією об’єкту той набір операцій, для якого цей об’єкт використовується взагалі.

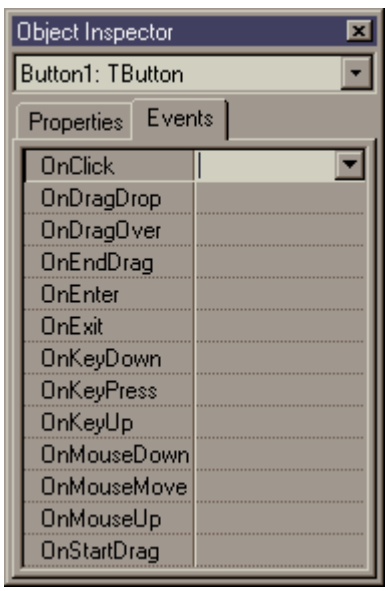
Наприклад:

- для форми головною подією є створення формою компонент та зміна параметрів як своїх, так і компонент. Всі дії відбуваються при запуску форми.
- для кнопки, як і для статичного тексту, головною подією є натискання на кнопку чи текст відповідно.

Розглянемо події, які можна використовувати для кнопки.

Події.

Вікно подій для кнопки має вигляд



Для того, щоб призначити, наприклад, події OnClick (буквально: “при клацанні”) якусь дію, досить двічі клацнути на самому об’єкту “кнопка” (Button1).
Тоді, у вікні Unit1 побачимо

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

Тобто Delphi створює пусту процедуру в програмі, й призначає її події. Всередині процедури програміст повинен написати команди, які виконаються при натисканні на кнопку (подія OnClick).
Про те, які ж команди можна використовувати, розглянемо у розділі “Програмування”.

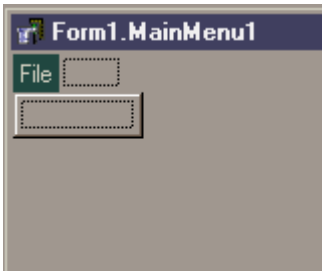
Створення меню MainMenu.

 - створення меню програми. За допомогою цього об'єкту можна створити меню у програмі. У формі Menu має вигляд

Для встановлення назви пунктів та підпунктів меню досить двічі клацнути на об'єкт лівою клавішею мишки. На екран буде виведено меню

Далі слід вказати назву головного пункту меню. Наприклад File. На екрані це виглядатиме 

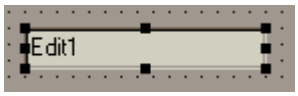
та



Далі, рухаючись униз ми додаватимемо підпункти меню (слід вказувати їх назви), і вправо – пункти меню. Якщо клацнути у будь-який з підпунктів меню двічі лівою клавішею мишки, то автоматично створиться підпрограма, у якій треба буде вказати послідовність операторів, що виконуватимуться при виборі відповідного пункту меню при роботі програми. Меню розташовується у програмі автоматично у лівому верхньому куті вікна програми.

Рядок редагування Edit.

 – рядок редагування. Використовується для введення та, іноді, для виведення інформації. На екрані має вигляд



Властивості

Частину властивостей рядок редагування має таку, яка є і у кнопки та статичного тексту. З важливих нових властивостей рядок редагування має

- текст, що виводиться у вікні рядку редагування.

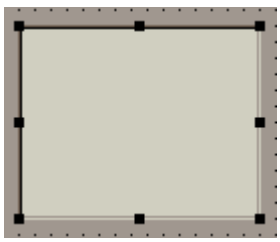
- є чи нема можливості вводити текст через рядок редагування.

- виводити чи не виводити рядок редагування на екран.

Текст, що відображений(введений) у вікні записується у змінну Edit1.Text і має тип string (див далі розділ типи змінних).

Вікно для виведення тексту ListBox.

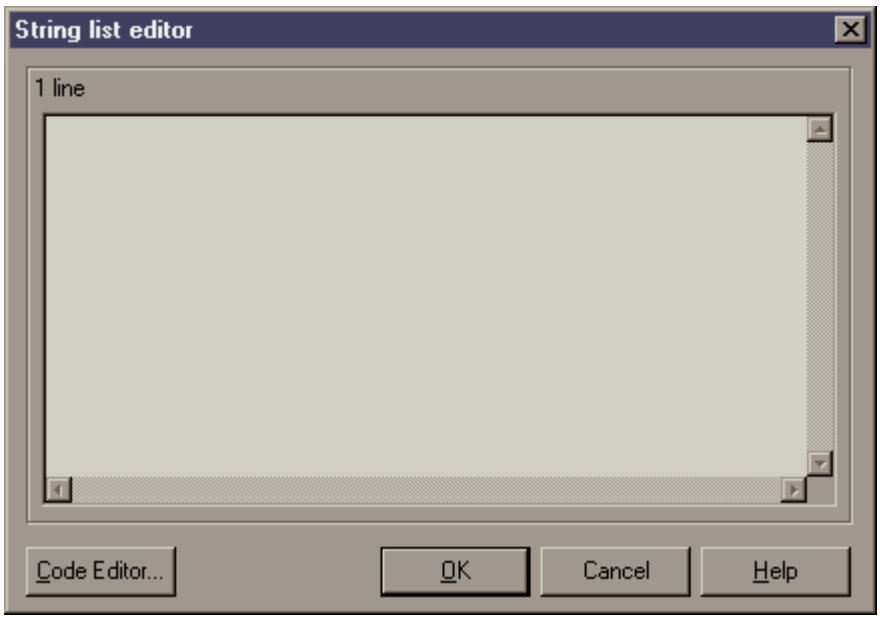
 - текстове вікно. Використовується для виведення більше за один рядок тексту у вікно з можливістю перегляду (прокрутки) тексту вверх-вниз. Видима ширина рядку залежить від ширини вікна ListBox. На екрані має вигляд



Властивості

Частину властивостей рядок редагування має таку, яка є і у кнопки та статичного тексту. З важливих нових властивостей рядок редагування має

- текст, що виводитиметься у ListBox на самому початку. При натисканні на  на екран виведеться вікно для введення тексту вигляду



<u>Додаткові процедури та функції ListBox.</u>

ListBox1.Items.Append.

Для зміни тексту у ListBox протягом роботи програми використовується процедура

ListBox1.Items.Append(s);

де параметр s є рядком. При виконанні цієї процедури, знизу до рядків дописується ще один з s.

ListBox1.Items.Clear.

Ця процедура очищає вікно ListBox від всього тексту.

ListBox1.Items.Delete.

Процедура

ListBox1.Items.Delete(i);

знищує рядок з номером i, де i – ціла змінна(див. типи змінних). Відмітимо, що нумерація рядків починається зверху з нульового номера.

ListBox1.Items.Insert.

Процедура

ListBox1.Items.Insert(i,s);

Вставляє текст s у рядок з номером i. Змінна s – рядкова, i – ціла. При цьому всі рядки після i зсуваються донизу.

Наприклад у нас є ListBox з текстом з чотирьох рядків

‘1 перший’
‘2 другий’
‘3 третій’
‘4 четвертий’

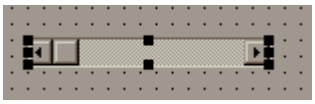
Тоді після запуску процедури *ListBox1.Items.Delete(2);* стан ListBox буде

‘1 перший’
‘2 другий’
‘4 четвертий’

Після запуску процедури *ListBox1.Items.Insert(1,’новий рядок’);* стан ListBox буде
‘1 перший’
‘новий рядок’
‘2 другий’
‘4 четвертий’

Полоса прокрутки ScrollBar

 - полоса прокрутки. Використовується для візуального вибору цілого значення з допустимого діапазону. На екрані у формі має вигляд



Властивості

Для полоси прокрутки головною подією є зміна координати курсору TForm1.ScrollBar1Change.

- позиція (координата) курсору. Залежить від візуального розташування у полосі прокрутки та мінімального з максимальним значень діапазону.

- максимальне та мінімальне значення, які може приймати координата курсору у полосі прокрутки відповідно

- одиниця градації зміни положення курсору.

Наприклад, потрібно за допомогою полоси прокрутки встановити значення цілої змінної і , що може приймати значення від 10 до 100 з кроком 5(тобто може приймати значення 10, 15, 20, 25, 30,..., 90, 95, 100).
Для цього встановимо такі початкові значення параметрів

Min	10
Max	100
LargeChange	5

і у алгоритмічні частині програми присвоювати змінній і значення ScrollBar.Position.

RadioGroup. Вибір з переліку

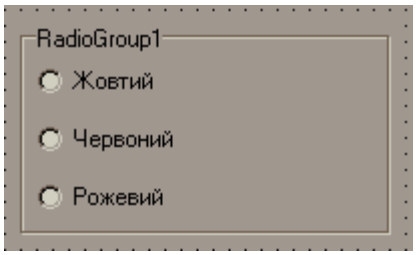
 - дозволяє вибрати одне значення з переліку. Має вигляд у формі

Для встановлення значень слід вказати початкові значення. Для цього треба зробити це або через параметр Radiogroup1.Items (

як це робилось у ListBox) або програмно. У першому випадку на екран буде виведено вікно, у якому можна ввести рядкі. Наприклад введемо 3 рядки

Жовтий
Червоний
Рожевий

Тоді у формі це матиме вигляд



Головною подією для RadioGroup є натискання на неї (яке взагалі відбувається разом з вибором значення).

Властивості

Частину властивостей RadioGroup має таку, яка є і у ListBox. З важливих нових властивостей рядок редагування має

- порядковий номер вибраного значення у переліку. За вмовчанням він має значення –1. Нумерація рядків здійснюється від 0. Тобто при виборі першого рядку значення RadioGroup1.ItemIndex буде 0.

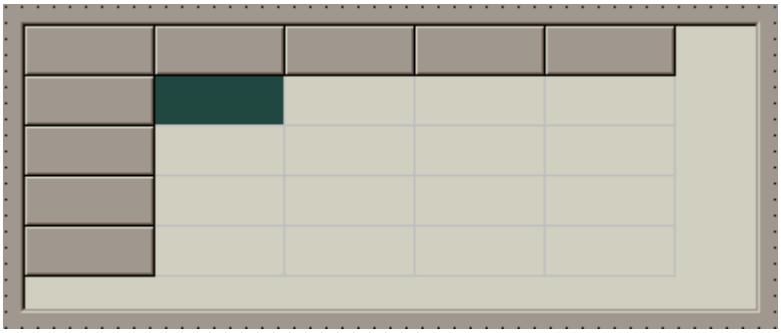
Додаткові процедури та функції RadioGroup.

Знищення, додавання та вставка рядку здійснюється так само, як і у ListBox, тільки назвою об'єкту є не ListBox, а RadioGroup. Тобто використовуються процедури RadioGroup1.Items.Append, RadioGroup1.Items.Clear, RadioGroup1.Items.Delete, RadioGroup1.Items.Insert відповідно.

Additional

Таблиця StringGrid

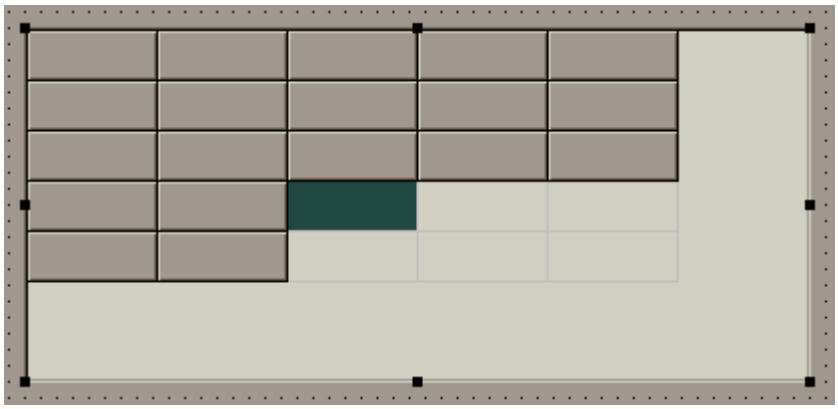
- використовується для виведення-введення даних у таблицю. Кожна клітинка таблиці має рядковий тип (див. типи змінних). SrtingGrid у формі має вигляд



Всі клітинки таблиці утворюють двовимірний масив рядків (див. типи змінних). Програмне звертання до кожної клітинки ведеться за номером стовпчика та рядку. Нумерація рядків та стовпчиків ведеться з 0. Наприклад клітинка, що розташована у 3-му рядку 2-го стовпчика має значення StringGrid1.Cells[nc,nr], де nc – номер стовпчика, nr – номер рядка.

Властивості

ColCount | 5 - кількість стовпчиків таблиці.
RowCount | 5 - кількість рядків таблиці.
FixedCols | 1 та FixedRows | 1 - кількість рядків та стовпчиків відповідно, що відводиться тільки під виведення інформації. Як правило це рядки з описом стовпчиків чи рядків. У нашому випадку обидва значення дорівнюють 1. Якщо, наприклад встановити FixedCols=2 та FixedRows=3, то таблиця у формі матиме вигляд



За вмовчанням таблиця StringGrid використовується тільки для виведення інформації на екран. Для того, щоб значення, що записані у клітинках (крім відокремлених через FixedCols та FixedRows), необхідно встановити двічі клацнувши лівою клавішею мишки у +Options | [goFixedVertL і у вікні, що відкриється, встановити для параметру goEditing | False значення True. Також можна це зробити у програмі при виконанні

```
StringGrid1.Options:=StringGrid1.Options + [goEditing];
```

Відповідно відключити можливість редагування даних у таблиці можна через

```
StringGrid1.Options:=StringGrid1.Options - [goEditing];
```

Додаткові процедури та функції StringGrid.

StringGrid1.ColCount

У програмі можна міняти кількість стовпчиків та рядків, а також розмір таблиці StringGrid. Для цього досить встановити нове значення для кількості стовпчиків

StringGrid1.ColCount:=i;

де i – ціле число.

Наприклад

StringGrid1.ColCount:=8;

При цьому слід вважати на розмір таблиці StringGrid1.Width. При потребі його можна змінити вказавши

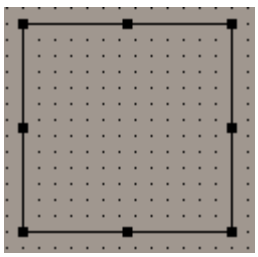
StringGrid1.Width:=600;

StringGrid1.RowCount

Функціонує аналогічно до StringGrid1.ColCount, тільки встановлює кількість рядків. За висоту таблиці відповідає змінна StringGrid1.Height.

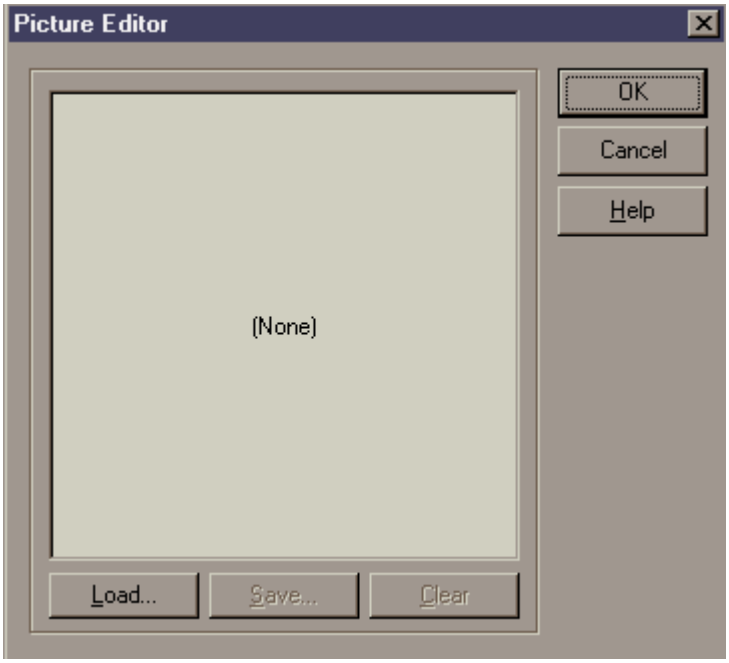
Малюнок Image

 - вставка малюнку. Дозволяє використовувати малюнки формату (з розширеннями) ico, bmp, wmf та emf. У формі до завантаження малюнку має вигляд

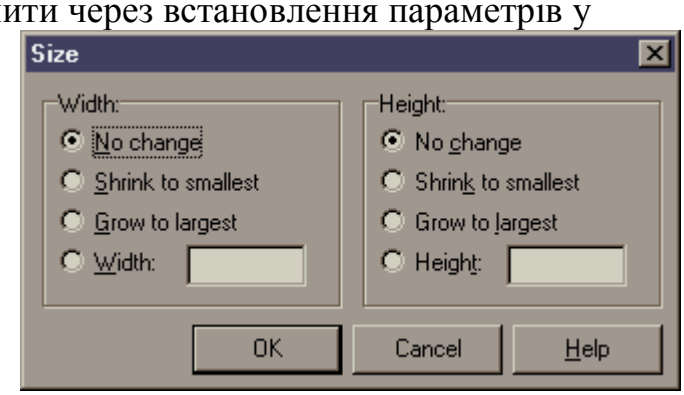


Для завантаження малюнку (встановлення) досить встановити параметр 

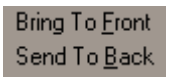
або двічі клацнути лівою клавішею мишки на вставлений у форму об'єкт Image. На екрані з'явиться діалогове вікно



де буде запропоновано через функцію Load… вибрати існуючий файл з малюнком. Далі малюнок можна зменшити через встановлення параметрів у

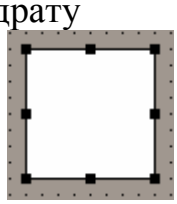


та перемістити за існуючі об'єкти, чи закрити малюнком їх через натискання на Image правою клавішею мишки та вибору з



Малюнок Shape.

 - дозволяє створювати у формі прості малюнки – прямокутник, еліпс, квадрат, круг. У формі за вмовчанням має вигляд квадрату



Властивості

- встановлення форми Shape. Можна вибрати з наведених:

stRectangle – стандартний прямокутник
stCircle – стандартний круг
stEllipse – стандартний еліпс
stRoundRect – стандартний прямокутник з закругленими кутами

stRoundSquare - стандартний квадрат з закругленими кутами
stSquare – стандартний квадрат.

- у пензлі Brush можна вибрати колір заповнення фігури Brush.Color з наведеного переліку кольорів.

- у ручці Pen можна вибрати колір контуру фігури Pen.Color з наведеного переліку кольорів.

Dialogs.

У розділі Dialogs розташовані різноманітні “заготовки” для діалогів. Розглянемо дві з них – це OpenDialog та SaveDialog.

OpenDialog

 - OpenDialog. Використовується для вибору файлу з існуючих під час роботи програми.

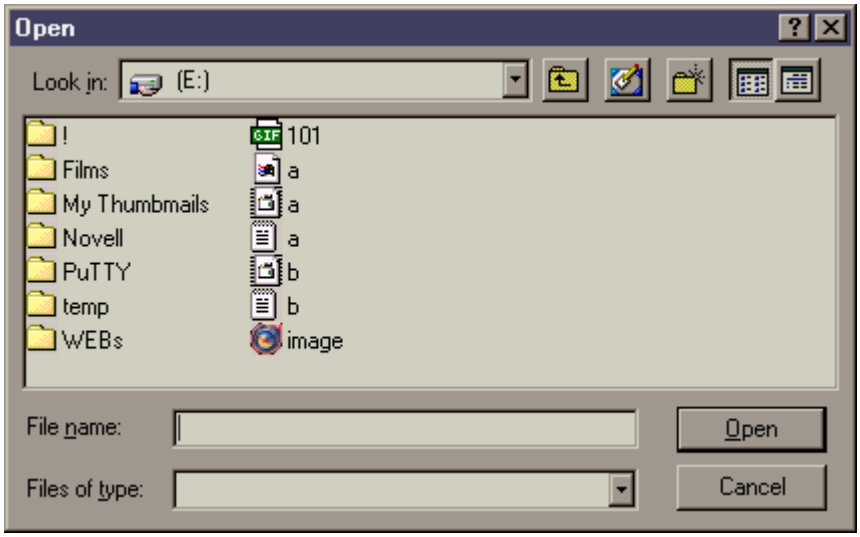
Активізації діалогу можна досягнути за допомогою перевірки параметра execute, що приймає значення True(правда), якщо файл був вибраний, і False, якщо ні. Ім'я вибраного файлу присвоюється параметру FileName, що має рядковий тип (див. Рядки).

Наприклад, якщо у нас є об'єкт OpenDialog1, то за допомогою запуску умовного оператора if (див. розділ Робота з текстовими файлами.) ми отримаємо ім'я вибраного файлу у OpenDialog1.FileName тобто:

if OpenDialog1.Execute *then* s:=OpenDialog1.FileName;

де s – рядкова змінна.

При цьому, під час роботи програми, на екрані буде подано вікно вигляду



у якому можна вибрати ім'я файлу.

Зазначимо, що головним призначенням об'єкту OpenDialog є тільки вибір файлу та присвоєння його імені та повного шляху якійсь рядковій змінній.

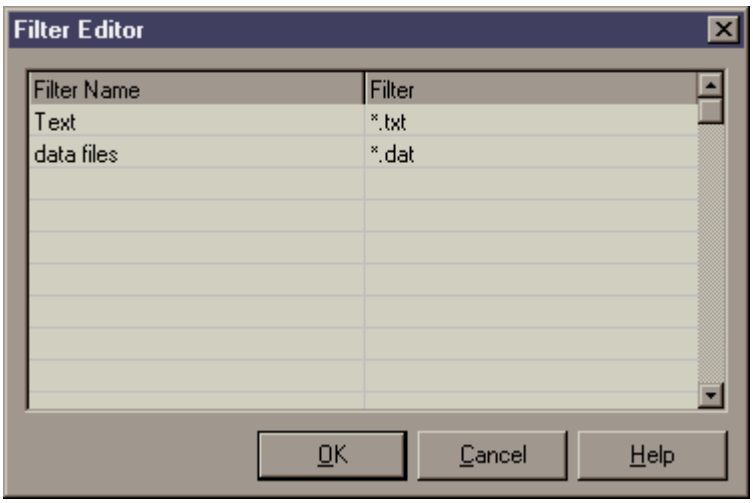
Параметри

Filter.

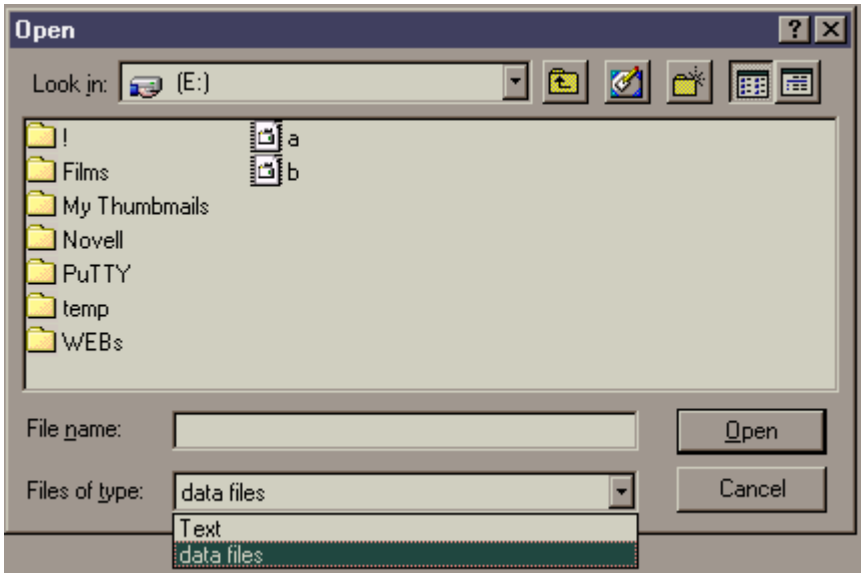
Перш за все OpenDialog дозволяє обмежити файли за розширеннями, які можна вибирати у OpenDialog. Цей параметр носить назву Filter 

.

Після клацання у  на екран буде виведено вікно



де пропонується ввести розширення та описати їх. Зауважимо, що розширення треба вводити, використовуючи маску *.Після цього при активізації діалогу, на екран вже виводитиметься вікно вигляду



де у переліку типів файлів, що дозволено відкривати, будуть тільки ті типи, які описані у Filter.

FilterIndex.

- вказується номер елементу у переліку фільтрів, що буде виводитись у першу чергу. У нашому випадку встановлено 2, тому у першу чергу виводиться другий рядок (data files).

InitialDir

- вказується каталог, з якого буде починатись вибір файлу. У нашому випадку – це просто диск E:.

SaveDialog

- SaveDialog. Використовується для встановлення імені файлу під час роботи програми.

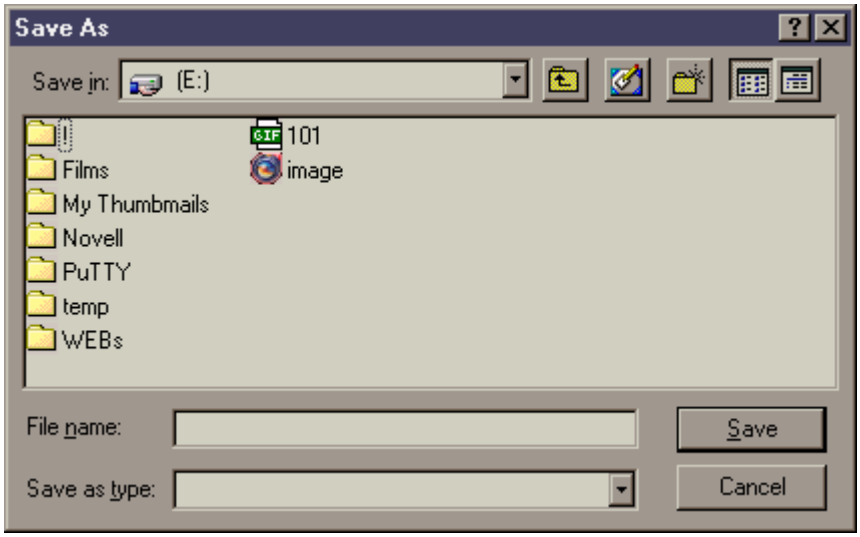
Принцип роботи SaveDialog аналогічний до OpenDialog. Відмінністю є те, що нам не обов'язково вибирати існуючий файл, а можна вказати нове ім'я файлу.

У програмі це матиме вигляд

```
if SaveDialog1.Execute then s:=SaveDialog1.FileName;
```

де s – рядкова змінна.

При цьому, під час роботи програми, на екрані буде подано вікно вигляду



у якому можна вказати ім'я файлу.

Зазначимо, що головним призначенням об'єкту SaveDialog є тільки визначення імені файлу та присвоєння його та повного шляху якійсь рядковій змінній.

Основи програмування.

Введемо спочатку поняття алгоритму.

Алгоритмом називатимемо скінченну послідовність команд, яка виконується у строгій послідовності від першої до останньої, причому жодна команда не виконується, поки попередня не закінчила виконання.

Елементи мови

Звичайна розмовна мова складається з чотирьох головних елементів: *символів, слів, словосполучень та речень.*

Алгоритмічна мова теж базується на схожих елементах, тільки *слова називають елементарними конструкціями, словосполучення – виразами, речення – операторами*

Описом мови є опис цих чотирьох елементів.

Під описом символів розуміють перелік усіх припустимих символів, під описом елементарних конструкції – правила їх створення, опис виразів – правила створення виразів, що мають сенс у конкретній мові, а опис операторів є розглядом усіх типів операторів, що припустимі у даній мові. Опис кожного елементу мови задається її синтаксисом та семантикою.

Під *синтаксисом* розумітимемо перелік правил побудови елементів мови, а під *семантикою* – зміст та правила використання тих елементів мови, для яких визначено синтаксичні визначення.

Символи мови – це головні неподільні знаки, у термінах яких і пишуться всі тексти мови.

Елементарні конструкції – мінімальні одиниці мови, що мають самостійний зміст і утворюються з символів мови.

Вираз у алгоритмічній мові складається з елементарних конструкцій та символів та вказує правило обчислення деякого значення.

Оператор визначає повний опис деякої дії, яку необхідно виконати. Для опису складних дій може знадобитись навіть група операторів. У цих випадках оператори об’єднуються у складні оператори та блоки.

Дії, вказані операторами, виконуються над даними.

Речення алгоритмічної мови, у яких подана інформація про типи даних, називаються *описовими* або операторами, що не виконуються.

Об’єднання єдиним алгоритмом сукупності операторів та описів і утворює *програму*

Синтаксичні визначення можуть задаватись як формальним, так і неформальним чином.

Існують три формальних методи:

- металінгвістична символіка, що називається формулами Бекуса-Наура;
- синтаксичні діаграми;
- дужкові конструкції;

Ми використовуватимемо неформальний метод

Головні символи

Перерахуємо тепер припустимі символи мови Delphi Pascal.

1. Літери латиниці – великі та маленькі (pascal при виконанні програми не розрізняє велика літера чи мала – розмір літер використовується лише для зручності написанні програми)

2. Цифри від 0 до 9
3. Знаки операцій + – * / = < > <= >= <> := & @
4. Знак підкреслення _ та знак пропуск
5. Обмежувальні символи , . ‘ ( ) [ ] (. .) { } (* *) .. ; :
6. Специфікатори ^ # $
7. Службові зарезервовані слова

До елементарних конструкцій мови Delphi Pascal віднесемо імена, числа та рядки.

Іменами (ідентифікаторами) називають елементи мови – константи, мітки, типи, змінні, процедури, функції, модулі, об’єкти.

Ім’я – це послідовність літер та цифр, що починається завжди з літери. В іменах допускається символ підкреслення _ . Ім’я може складатись з довільної кількості символів, хоча значущою є тільки послідовність з не більш ніж 63 символів. У Pascal заборонено використовувати службові слова та стандартні константи, типи, процедури, функції як імена файлів.

Для покращення наглядності програми у Pascal можна вставляти пропуски та порожні рядки. Принаймні один пропуск необхідно вставляти між двома послідовними іменами, числами або стандартними чи службовими іменами. Пропуски не допускається вставляти всередині імен та чисел.

Приклади імен мови Pascal

A aa asd h1 z_1 p23_127 ResultOnScreen

Числа у Pascal звичайно записуються у десятковій системі числення. Вони можуть бути цілими (1 2 67 -124) та дійсними (1.2 -0.12 345.6). Цілі числа записують у формі без десяткової точки.

Дійсні числа можуть записуватись і у формі запису десяткового порядку, що позначається літерою E.

Наприклад

1.79E10 33.1E-8 -123.6E5

Pascal допускає запис цілих чисел у шістнадцятирічній системі числення

Наприклад

$FF $12 $A9B0

Рядок у Pascal – це послідовність символів, що записана між апострофами. Якщо у рядку треба використати апостроф, як змістовний символ, то його слід записати, як два апострофи ‘’.

Наприклад

‘Цей рядок пов’‘язаний з наступним’

‘Просто рядок’

Концепція типу даних.

У математиці змінні класифікуються у відповідності до їх головних характеристик. Здійснюється строге розмежування між дійсними, комплексними та логічними змінними, між змінними, що є окремою змінною та множиною змінних і т.і.

При обробці даних на ЕОМ така класифікація є ще більш суттєвою. У кожній алгоритмічній мові кожна константа, змінна, вираз чи функція є певного типу.

У Pascal існує правило: тип явно вказується при описі змінної чи функції, який передує їх використанню.

Концепція типу змінних мови Pascal має такі головні властивості:

- будь-який тип даних визначає множину значень, до якого належить константа, які можуть приймати змінні та функції, або генерувати операція чи функція;
- тип значення, що задається константою, змінною чи виразом можна визначити за їх виглядом чи описом;
- кожна операція чи функція вимагає аргументів фіксованого типу і видає результат фіксованого типу.

Тип визначає:

- можливі значення змінних, констант, функцій, виразів, що належать даному типу;
- внутрішню форму подання даних в ЕОМ;
- операції та функції, які можуть виконуватись з величинами даного типу;

Класифікація типів даних.

У мові Pascal існують скалярні та структуровані типи даних. До скалярних типів стосуються стандартні типи і типи, що визначаються користувачем.

Стандартні

- цілі
- дійсні
- символьний
- логічний
- адресний

Типи, що визначаються користувачем

- перерахований
- інтервальний

Структуровані типи

- масиви
- множини
- записи
- файли
- рядки символів

Крім перерахованих є ще два типи даних:

- процедурний
- об’єктний

З групи скалярних типів можна виділити порядкові типи, які характеризуються такими властивостями:

- всі можливі значення порядкового типу є обмеженою впорядкованою множиною
- до довільного порядкового типу можна застосувати стандартну функцію Ord, яка, як результат, повертає порядковий номер конкретного значення у даному типі
- до довільного порядкового типу можна застосувати стандарті функції Pred та Succ, які повертають, як результат, наступне або попереднє значення відповідно
- до довільного порядкового типу можна застосувати стандартні функції Low та High, які повертають найбільше та найменше значення величин даного типу

Стандартні типи даних.

Розглянемо детальніше найбільш вживані стандартні типи змінних. До них стосуються цілі типи, дійсні, символьний та адресний.

Цілі типи.

Визначають константи, змінні та функції, значення яких реалізується множиною цілих чисел, що припустимі у даній ЕОМ.

Тип	Діапазон значень
Shortint	-128 ... 127
Integer	-32768 ... 32767
Longint	-2147483648 ... 2147483647
Byte	0 ... 255
Word	0 ... 65535

Арифметичні операції

Над цілими операндами можна виконувати такі арифметичні операції:

Операція	Знак операції	Тип результату
Додавання	+	цілий
Віднімання	–	цілий
Множення	*	цілий
Ділення	/	дійсний
залишок від ділення	Div	цілий
Ціла частина від ділення	Mod	цілий

Наприклад:

2+2=4 5 - 3=2 12 div 5 =2 13 mod 3 = 4 12 div 4 = 0

Операції відношення

Операції відношення, що застосовані до цілих операндів дають результат логічного типу True або False (істина - неправда)

Існують такі операції відношення:

Операція	Знак
Дорівнює	=
Більше	>

Менше	<
більше або дорівнює	>=
менше або дорівнює	<=
не дорівнює	<>

Стандартні функції.

До аргументу x цілого типу можуть застосовуватись стандартні функції:

Функція	Оператор	Тип результату
Модуль x	Abs(x)	Цілий
Квадрат x	Sqr(x)	Цілий
Наступне значення x (або x+1)	Succ(x)	Цілий
Попереднє значення x (або x-1)	Pred(x)	Цілий
Синус x	Sin(x)	Дійсний
Косинус x	Cos(x)	Дійсний
Тангенс x	Tan(x)	Дійсний
Логарифм натуральний x	Ln(x)	Дійсний
Експонента x	Exp(x)	Дійсний
Корінь квадратний x	Sqrt(x)	Дійсний
Арксинус x (x задано у радіанах)	ArcSin(x)	Дійсний
Арккосинус x (x задано у радіанах)	ArcCos(x)	Дійсний
Арктангенс x (x задано у радіанах)	ArcTan(x)	Дійсний
Перевірка парності x	Odd(x)	Логічний
Переведення цілого аргумента у рядковий	IntToStr(x)	Рядковий

Наприклад

X=10 Odd(x)=False, X=5 Odd(x)=True,

Дійсні типи.

Дійсні типи визначають ті дані, які реалізуються підмножиною дійсних чисел, що припускається у даній ЕОМ. Всі дійсні числа можуть бути як від'ємними, так і додатніми.

Тип	Діапазон значень	Кількість цифр мантиси (кількість цифр до символу E)
Real	2.9e-39 .. 1.7e+38	11
Single	1.5e-45 .. 3.4e+38	7
Double	5.0e-324 .. 1.7e+308	15
Extended	3.4e-4932 .. 1.1e+4932	19
Comp	-9.2e+18 .. 9.2e+18	19

Арифметичні операції.

Над дійсними операндами можна виконувати такі арифметичні операції

Операція	Знак операції	Тип результату
Додавання	+	Дійсний
Віднімання	–	Дійсний
Множення	*	Дійсний

Ділення	/	Дійсний

До величин дійсного типу застосовні всі операції відношення, що дають логічний (булевський) результат. Причому один з операндів, що бере участь у цих операціях, може бути цілим.

Стандартні функції.

Функція	Оператор	Тип результату
Модуль x	Abs(x)	Дійсний
Квадрат x	Sqr(x)	Дійсний
Синус x	Sin(x)	Дійсний
Косинус x	Cos(x)	Дійсний
Тангенс x	Tan(x)	Дійсний
Логарифм натуральний x	Ln(x)	Дійсний
Експонента x	Exp(x)	Дійсний
Корінь квадратний x	Sqrt(x)	Дійсний
Арксинус x (x задано у радіанах)	ArcSin(x)	Дійсний
Арккосинус x (x задано у радіанах)	ArcCos(x)	Дійсний
Арктангенс x (x задано у радіанах)	ArcTan(x)	Дійсний
Дробова частина x	Frac(x)	Дійсний
Ціла частина x	Int(x)	Дійсний
Число пі	Pi	Дійсний
Ціла наближена частина x	Round(x)	Цілий
Ціла частина x (методом видалення дробової)	Trunc(x)	Цілий
Переведення дійсного аргументу у рядковий	FloatToStr(x)	Рядковий

Логічний тип.

Логічний тип (або Boolean – булевський) визначає ті дані, які можуть приймати логічні значення True або False.

До булевських операндів можна застосовувати логічні операції:

not and or xor

Логічний тип визначений таким чином, що False < True. Це дозволяє застосовувати до булевських операндів всі операції відношення.

Символьний тип.

Символьний тип, або тип Char, визначає впорядковану сукупність символів, припустимих у даній ЕОМ. Значення символьної змінної або константи – це один символ з припустимого набору. Символьну константу можна записати у програмі трьома методами:

- як один символ, розміщений між апострофами (‘F’ ‘a’ ‘3’ ‘!’)
- за допомогою конструкції #N, де N – код відповідного символу, причому значення коду має бути у межах 0..255 (#48 #55 #128)
- за допомогою конструкції вигляду ^M, де M – код відповідного керуючого символу + 64

До величин символьного типу можна застосовувати всі операції відношення.

Стандартні функції
Для величин символьного типу визначені дві функції перетворення:
Ord(C) – повертає порядковий номер символа С в наборі символів (Ord (‘a’) = 97)
Chr(K) – повертає символ з порядковим номером К (Chr (97) = ’a’)

До аргументів символьного типу можна застосовувати функції, які визначають:
Pred (C) – попередній символ (Pred (‘c’) = ’b’)
Succ (C) - наступний символ (Succ (‘3’) = ’2’)
За відсутності попереднього та наступного символу значення цих функцій не визначено.

Для літер з інтервалу ‘a’ .. ‘z’ можна застосовувати функцію
UpCase(C), котра переводить маленькі літери у великі (UpCase (‘f’) = ‘F’)

Типи констант
Тип констант визначається за їх виглядом: Константи цілого типу – це цілі числа, що не мають десяткової точки, константи дійсного типу – дійсні числа, логічні константи – True або False, символьні – або рядки довжиною у один символ, або конструкції вигляду #K або ^K та ін.

Переліковий тип

Переліковий тип є обмеженою впорядкованою послідовністю однотипних скалярних констант, що і складають цей тип. Значення кожної константи задається її ім’ям. Імена окремих констант відокремлюють одне від одного комами, а вся сукупність констант, що складає переліковий тип, обмежується круглими дужками.
Програміст об’єднує до однієї групи за якоюсь ознакою всю сукупність значень, що складають переліковий тип.
Наприклад, переліковий тип turn (поворот) об’єднує скалярні значення left, right; cube (кубик) об’єднує значення c1, c2, c3, c4, c5, c6; move (рух) об’єднує значення left, right, backward, forward.
Переліковий тип описується у блоці опису типів, що починається зі службового слова **type**. Наприклад:

```
Type
        turn = ( left, right );
        cube = ( c1, c2, c3, c4, c5, c6 );
```

Кожне значення є константою свого типу і може належати тільки одному з перерахованих типів, що задані у програмі. Наприклад типи turn не може бути визначеними у одній програмі з типом move через те, що обидва типи містять однакові константи left та right.

Далі, після опису типу, його можна використовувати, як звичайний стандартний тип для опису змінних.
Впорядкована послідовність значень, що становить переліковий тип, автоматично нумерується, починаючи з 0 і далі через 1. Звідси випливає, що до перелікових змінних та констант можна застосовувати стандартні функції Pred, Succ та Ord.
Змінні перелікового типу не можуть бути елементами списків входу або виходу.

Інтервальний тип.

Відтинок довільного порядкового типу може бути визначеним як інтервальний або обмежений тип. Відтинок задається діапазоном від найменшого до найбільшого значення констант, розділених двома крапками "..". Константи можуть використовуватись цілого, символьного, логічного (булевського) та перелікового типів. Скалярний тип, на якому базується відтинок, називається базовим типом.
Найменше та найбільше значення констант називають нижньою та верхньою границями відтинка, що визначають інтервальний тип. Нижня границя має бути меншою за верхню.
Наприклад:

```
Type
    interval = 1..100;
```

Над змінними інтервального типу можуть виконуватись всі операції та застосовуватись всі стандартні функції стосовні відповідного базового типу.
При застосуванні у програмах інтервальний типів даних можу здійснюватись контроль над тим, щоб значення змінних не виходили за межі, встановлені у описі інтервального типу.

Ініціалізація змінних.

У відкомпільованій програмі (тобто робочій програмі, файл якої має розширення exe) для всіх змінних виділено місце у пам'яті та всім змінним присвоєно нульове значення.

Для задання початкових значень (ініціалізація змінних) Delphi Pascal, як і всі попередні версії Pascal, дозволяє присвоювати значення змінним одночасно з їх описом. Для цього використовується конструкція

```
Ім'я_змінної : тип = значення;
```

Наприклад

```
i : integer = 5;
```

Зауважимо, що тип змінної та тип значення має співпадати.
Змінні перелікового типу можуть бути описані у блоці (розділі) опису змінних. Наприклад:

```
Var
    Turn : ( left, right );
```

При цьому імена типів відсутні, а змінні визначаються сукупність значень, що складають даний переліковий тип. До змінних перелікового типу можна застосовувати оператор присвоєння.

```
Turn := left;
```

Масиви

Масив – це обмежена впорядкована сукупність однотипних величин. Кожна окрема величина називається компонентою масиву. Тип компонент масиву може бути довільним, з тих, що застосовні у Delphi, крім файлового типу. Тип компонент називається базовим типом. Вся сукупність компонент визначається одним ім'ям. Для визначення окремих компонент використовується конструкція, що називається змінною з індексом або індексами.
Масив описується таким чином:

```
var
    a: array [1..100] of integer;
    b: array [1..20,1..30] of real;
```

У залежності від розмірності масиву (одновимірний /рядок/, двовимірний /таблиця/ і т.д.) звертання до його елементу має вигляд.

```
a[30], b[2,4]
```

Для двовимірних (та більше) масивів першим індексом є ціле число – номер рядку, другий – номер стовпчика.

Відмітимо, що при роботі з масивами треба уважно слідкувати, щоб у програмі не відбувалось звернення до неіснуючого елементу масиву. Наприклад, якщо масив A у нас складається зі 100 елементів з номерами від 1 до 100, то звертання до a[0] чи a[101] є недопустимим.

Ініціалізація масиву.

Ініціалізація масиву (присвоєння початкових значень компонентам масиву) може здійснюватись двома шляхами:
Перший – послідовне присвоєння елементам масиву відповідних значень.

```
A[3]:=5;
B[10,20]:=3.5;
```

Другий – з використанням типізованих констант
Наприклад:

```
type ar = array [1..10] of real;

const
ab : ar = (1, 4.6, 3.3, 1.6, 5.5, 3.2, 0, 444.5, 0.23, 1024.2 );
```

Двовимірні масиви зберігаються у пам'яті ЕОМ по рядкам.

Індекс(и) масиву – число або вираз. Тип у індексу може бути переліковим, інтервальним, цілим, булевським та символьним.

Наприклад:

```
Type
        S1 = 1..100;
        S2 = ( left, right, back, forward );
Var
        A : array [1..100] of real;
        B : array [s1,s2] of integer;
        C : array [s2] of char;
        CC : array [s2] of char;
        D : array ['a'..'z'] of integer;
```

У операторній (алгоритмічній) частині програми один масив може бути присвоєний іншому, якщо їх типи та розмірність є ідентичною. Наприклад, у нашому випадку:

```
C := CC;
```

Кожен елемент масиву можна використовувати як окрему змінну типу, який вказаний при описі масиву.

Рядки

Особливе місце у Delphi займають масиви символів, або рядки – String. Стандартні змінні елементів форми, що використовуються для введення чи виведення інформації, такі як Label.Caption, Form.Caption, Edit.Text, Button.Caption та інші мають рядковий тип.
Символьний рядок може бути або рядковою константою, або рядковою змінною. Рядкова константа, або рядок, є сукупністю символів, обмеженою апострофами. Рядок – елементарна конструкція мови Delphi Pascal.
Рядкові константи можуть входити до складу виразів. Як і числові константи, вони можуть бути описаними у блоці опису констант.
Наприклад, якщо рядок міститиме не більше за 50 символів, то його тип буде визначений, як

```
Type
        S : string [50];
```

Довжина рядку не може перевищувати 255 символів. Тому, якщо ми не вказуємо максимальну довжину, то вона автоматично вважається 255. Тобто при описі

```
Type
        S : string [255];
        AS : string ;
```

типи S та AS будуть повністю ідентичними.
Якщо розмір текстового рядку при присвоєнні перевищує допустиму описану кількість символів для даної змінної, то Delphi збереже тільки стільки перших символів рядку, яка кількість вказана при описі змінної (типу).

Особливістю змінних типу String є те, що до них можна звертатись, як до скалярних змінних, так і як до масивів. Наприклад:

```
var
        SA,S: string;
…
begin
…
        S := ’Перша спроба’;
        SA := ‘роботи з Delphi’;
        SA := S+SA;
…
        S[2] := ’a’;
        S[3] := SA[5];
…
```

Зазначимо, що при використанні рядкової змінної, як масиву символів, нижня границя індексу дорівнює 1. Крім того 0-й елемент рядку містить інформацію про довжину всього рядка у вигляді конструкції #N, де N – число у діапазоні від 0 до 255.

Операції та функції.

Для отримання інформації про довжину рядку, до змінних рядкового типу можна застосовувати функцію **length (s)**, де s – рядкова змінна.

Також, крім операції присвоєння, для рядків визначені операції порівняння та приєднання (конкатенації).

= < > <> >= <= +

Для порівняння рядків можна застосовувати всі операції відношення. Порівняння рядків здійснюється посимвольно, починаючи з першого символу. Рядки вважаємо однаковими, якщо вони мають однакову довжину та співпадають посимвольно.

Рядки можуть виступати частинами списку вводу-виводу, при цьому використовується ім’я рядку без індексів.

При введенні рядкових змінних, кількість символів у рядку може бути як більше, ніж описана довжина рядку (розглянуто вище), так і менше. У останньому випадку символи, що були введені, розташуються на початку, решта ж рядку буде заповнена пропусками.

Ініціалізація.

Ініціалізація рядків може здійснюватись за допомогою типізованих констант:

```
Const
        CompName: string[9] = ‘Dual Xeon’;
```

Оператор присвоювання.

Цей оператор використовується тоді, коли змінній треба присвоїти якесь значення як фіксоване, так і значення іншої змінної чи виразу. Він має вигляд:

ім'я_змінної **:=** *значення;*

знак “:=” – це лексема, яку не слід плутати зі знаком =, який використовується для порівняння.

Оператор присвоювання “працює” таким чином:

- Обчислюється значення виразу правої частини оператору
- Змінній лівої частини присвоюється обчислене значення

Наприклад

```
i := 7;
```

Процедури та функції.

Ще одним дуже важливим компонентом програми на Delphi є процедури та функції. Як вже зазначалось вище, головний файл проекту просто запускає на виконання форму з усіма її подіями, тому тіла програми, як такого (як це було у попередніх версіях мови Pascal) вже немає.

Всі події та алгоритмічна реалізація програми описуються у процедурах та функціях, як стандартних, так і написаних автором програми.

Підпрограма.

Підпрограма – це поіменована (тобто кожна підпрограма має своє унікальне для даної програми чи модуля ім’я) послідовність операторів, які визначені та записані тільки у одному місці програми (модуля) так, що їх можна використовувати при виконанні одного або кількох фрагментів програми (модуля). Це робиться у двох випадках:

1. Однакова послідовність операторів використовується у багатьох моментах реалізації алгоритму. Тоді для економії місця та обсягу програми цю послідовність операторів “виносять” до підпрограми, а потім просто викликають цю підпрограму у відповідних моментах.
2. Для виділення фрагменту алгоритму у окремий блок. Цим досягається краща візуалізація виконання алгоритму.

У Delphi існує два типи підпрограм: процедури та функції.

При використанні процедур та функції у Delphi відповідний модуль має містити у собі текст процедури або функції та звертання (сам виклик) до неї.

Послідовність текстів процедур та функцій має йти у порядку зростання вкладеності. Тобто спочатку описуються процедури, які не використовують нестандартних процедур та функцій, потім ті, що можуть використовувати написані автором підпрограми, описані вище у програмі.

Підпрограма може містити такі ж блоки опису модулів, констант, типів, змінних та ін., що і сам модуль.

Змінні, константи, та інші описові блоки підпрограми “працюють” тільки у цій підпрограмі (локальні змінні, локальні типи та ін). Крім того підпрограма може використовувати усі змінні, типи та ін., описані у самому модулі (перед текстом підпрограми).

Функції

Головним результатом дії функції є значення. На відміну він процедури, кожна функція обов’язково має фіксований тип результату.

Наприклад, якщо звернутись до стандартних функцій, то функція sin має дійсне значення, strtoint – ціле, floattostr – рядкове.

Розглянемо загальний вигляд функції:

function *ім'я_функції*(*опис параметрів*) : *тип*;
блок опису локальних констант, змінних, типів;
begin
послідовність операторів;
ім'я_функції **:=** *остаточне_значення_функції;*
end;

Зазначимо, що остаточне_значення_функції має бути того ж типу, що і сама функція.

Наприклад:

```
function extent (a:real; b:integer; var bb:boolean): real;
var
        s:real;
begin

        s:=exp(b*ln(a));
        bb:=true;
        extent:=s;

end;
```

Функція extent має 3 параметри, які передаються їй керуючою програмою, модулем чи іншою підпрограмою. Перші два параметри a та b (дійсний та цілий) передаються у функцію для використання, причому їх значення функцією не повертаються у батьківську підпрограму (тобто підпрограму, звідки викликається ця функція). Третій же параметр (булевського типу) також використовується у функції, але його значення повертається до батьківської підпрограми.
Тому допустимими є виклики цієї функції

```
var
        i, j: integer;
        ab : boolean;
        s1, s2, s3 : real;

…
ab:=false;
s1:=extent (1.5, 3, ab);
s2:=4.5;
i:=3;
s1:=extend(s2, i, ab);
...
```

і недопустимими

```
...
s1:= extent (1.5, 3, false);
...
```

Недопустимість виклику виникає з того, що після виконання усіх операторів функції extent у батьківську підпрограму передається крім самого значення функції ще й

значення параметру (у нашому випадку це третій параметр bb:boolean), яке не може передатись через те, що третім параметром виступає не змінна, а її значення.

Локальні та глобальні змінні.

Змінні, що описані у програмі (модулі) можна використовувати в усіх функціях та процедурах, що містяться у ній. Ці змінні називають глобальними змінними.
Змінні, які описуються у підпрограмі поширюють свою дію тільки на цю підпрограму і називаються локальними.
Глобальні змінні, імена яких співпадають з локальними підпрограм та функцій, не "діють" у цих підпрограмах та функціях. Усі ж локальні змінні там можна використовувати.

Процедури

Результатом виконання процедури є не якесь значення, а зміна існуючих значень, запуск інших процедур, функцій та процесів.
Розглянемо загальний вигляд процедури:

procedure *ім'я_процедури*(*опис параметрів*);
блок опису локальних констант, змінних, типів;
begin
послідовність операторів;
end;

На відміну від функції, в її заголовку немає імені типу для значень, породжуваних у результаті виклику, тому що ніякі значення не породжуються. За цією ж причиною в тілі процедури не може бути операторів присвоювання з її ім'ям у лівій частині. Виклик процедури складається з імені й аргументів у дужках і записується як окремий оператор.
Наприклад

```
Reset(Input);
```

Процедури доцільно використовувати у тих випадках, коли у різних місцях програми необхідно виконати однакову послідовність операцій з різними параметрами. Наприклад, у багатьох алгоритмах сортування часто використовується послідовність операторів які міняють значення двох змінних місцями. У цьому випадку доцільно винести цю послідовність операторів у окрему процедуру. Це матиме вигляд:

```
procedure repl (var a,b : real);
var
        t:real;
begin
        t:=a;
        a:=b;
        b:=t;
end;
```

і викликається ця процедура, наприклад для перестановки значень a[4] та a[5] масиву, що описаний як

```
var
        a: array [1..1000] of real;
```

таким чином

```
…
repl (a[4],a[5]);
…
```

var у описі процедури носить таке саме навантаження, як і у описі функції.

Структура програми.

Проект

Як вже зазначалось вище, програма на Delphi складається з трьох файлів, але алгоритмічна частина записується у одному файлі, за вмовчанням який називається unit1.pas. Цей файл не є самостійною програмою, а модулем, який викликається головною програмою, текст якої розташований у файлі, який за вмовчанням носить ім'я Project1.dpr. Наведемо текст файлу Project1.dpr.

```
program Project1;

uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};

{$R *.RES}

begin
  Application.Initialize;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

Отже, головна програма (сам проект) на Delhpi складається з заголовку, описових розділів, та розділу операторів, де і реалізується алгоритм програми.
Заголовок програми містить ім'я програми. У нашому випадку це

```
program Project1;
```

Описи можуть містити розділ підключення бібліотек (модулів), розділ опису констант, типів, змінних, процедур та функцій.

Розділ опису модулів визначається службовим словом USES та містить імена модулів, що підключаються, як тих, що входять до складу середовища Delphi, так і тих, що написані користувачем. Розділ опису модулів має передувати всім описовим розділам програми.

Модуль.

Модуль (Unit) у Delphi є алгоритмічною частиною проекту. Текст його записується у окремий файл, якому за вмовчанням присвоюється ім'я **unit1.pas**. Модуль, на відміну від головної частини проекту, яка записана у файлі **project1.dpr**, не може виконуватись самостійно, він може тільки брати участь у виконанні програми. Але головна частина проекту **project1.dpr** передає керування подіями проекту модулю. Модулі дозволяють будувати персональні бібліотеки процедур та функцій, що дає можливість створювати програми будь-якого розміру та складності. Крім того у Delphi можна створювати зовнішні модулі (грім головного, що автоматично створюється), які потім під'єднуються до проекту і можуть використовуватись кількома проектами одночасно.
У загальному випадку модуль – це сукупність програмних ресурсів, призначених для використання іншими програмами (у нашому випадку – для використання проектами та іншими модулями, адже кожен модуль використовує інші модулі, як стандартні, так і написані користувачем). Наприклад, модуль **unit1.pas** за вмовчанням вже використовує набір стандартних модулів. Це легко бачити з рядку ініціалізації модулів файлу **unit1.pas**

```
uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
```

Під програмними ресурсами мається га увазі довільні події мови Delphi Pascal.
Всі програмні елементи модуля можна розбити на дві частини:

- програмні елементи, що призначені для використання іншими програмами та модулями (проектами та модулями). Такі елементи називають видимими поза модулем.
- програмні елементи, що є необхідними тільки для роботи самого модуля. Їх називають невидимими або скритими.

У відповідності до цього, модуль, крім заголовка, містить дві головні частини, що називаються інтерфейсом та реалізацією.
У загальному випадку, модуль має таку структуру:

```
unit < ім.'я_модуля>;                {Заголовок модуля}

interface

uses
    Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;
        {Опис зовнішніх модулів, що використовуються у програмі}

type            {Опис типів}
    TForm1 = class(TForm

private
{Опис приватних (тільки для цього модуля) властивостей, процедур та функцій об'єкту TForm1}
public
{Опис опублікованих (загальних) властивостей, процедур та функцій об'єкту TForm1}
end;
```

```
var                {опис глобальних змінних модуля}
    Form1: TForm1;

{Опис видимих програмних елементів модуля}

implementation

{$R *.DFM}   {підключення файлу опису розташування компонентів проекту на полотні формі}

{алгоритмічна реалізація усіх програмних елементів модуля – процедур та функцій, як скритих, так і видимих}

end.
```

Процедури та функції у модулі.

Використання у модулі процедур та функцій має свої особливості. Заголовок підпрограми містить всі відомості, що необхідні для її виклику: ім'я, перелік та тип параметрів, тип результату для функції. Якщо до підпрограми буде вестись звертання з інших модулів, то ця інформація має бути розташована до implementation, тобто у інтерфейсну частину модуля, а якщо ж підпрограма використовуватиметься тільки підпрограмами поточного модуля, то її опис слід розміщати після implementation. Саме тіло підпрограми (її текст), де реалізований її алгоритм, записується у частині реалізації модуля.

Інтерфейсна частина модуля містить тільки видимі (доступні для інших модулів) заголовки процедур та функцій.

У випадку, коли імена змінних інтерфейсної частини модуля та у самій програмі співпадатимуть, звертання вестиметься до змінних, що описані у програмі. Для звертання до змінної модуля слід використовувати складене ім'я, що складається з імені модуля та імені змінної, відокремлених крапкою.

Складені імена застосовуються при звертанні не тільки до імен змінних, а і до всіх імен, що описані у інтерфейсній частині модуля.

Рекурсивне використанні модуля заборонене.

Якщо у модулі є блок ініціалізації, то оператори з цього блоку виконуватимуться до початку виконання програми, у якій використовується цей модуль.

Для більшої наглядності розглянемо текст модуля програми – файл unit1.pas.

Як вже говорилось вище, за вмовчанням Delphi створює цей файл з базовими настройками та компонентами.

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

У модулі заголовком є

```
unit Unit1;
```

як це вже про це говорилось вище.

Опис констант дозволяє використовувати імена, як синоніми констант, їх необхідно визначити у розділі опису констант.

Наприклад, додамо до існуючого тексту програми опис констант:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs;

const
  k=1024;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

Ми додали константу k, яка має ціле значення 1024.

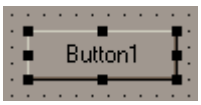
У розділі описі змінних необхідно визначити всі глобальні змінні, що використовуватимуться у програмі.

Наприклад доповнимо блок опису змінних двомі цілими змінними i та j.

```
var
  Form1: TForm1;
  i, j : integer;
```

Моделювання форми.

Перейдемо тепер безпосередньо до створення програми. Спочатку створимо форму, тобто візуальну частину програми. На відміну від більш старих версій мов програмування, Delphi значно полегшує роботу з інтерфейсною частиною. Для того, щоб додати кнопку до форми досить мишкою вибрати у стандартній частині (Standard) палітри компонентів кнопку  і клацнути потім на формі. На палітрі форми з'явиться



У модулі автоматично додастся до розділу опису типів Button1: TButton;

```
type
  TForm1 = class(TForm)
    Button1: TButton;
```

Згадаємо властивості кнопки. По-перше, надпис на кнопці, за який відповідає змінна caption або, якщо бути точнішим, Button1.Caption. Тип цієї змінної – string, початкове значення – 'Button1'. Це значення можна міняти як у самій програмі, так і у властивостях об'єкту. Розглянемо приклад, коли при натисканні на кнопку, напис на ній буде змінюватись на 'Pressed'.

Як розглядалось вище, кожен об'єкт форми має перелік допустимих подій. Також для кожного об'єкта є і головна подія. Наприклад для кнопки головною подією є натискання на неї.

Для того, щоб описати послідовність дій, які відбуватимуться при натісканні на кнопку, досить або двічі клацнути у неї мишкою, або вибрати у Object Inspector кнопки Events, і далі OnClick.

У тексті модуля буде створена нова процедура

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

причому її заголовок буде розташований у інтерфейсній частині модуля

```
type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
```

Далі, вся послідовність дій, що відбуватиметься при натисканні на цю кнопку, має бути розташована у алгоритмічній частині процедури між службовими словами begin та end;, що утворюють складний оператор.

У нашому випадку, у процедурі повинен змінюватись надпис на клавіші. Це матиме такий вигляд

```
procedure TForm1.Button1Click(Sender: TObject);
begin
        Button1.Caption:=' Pressed ';
end;
```

Весь текст модуля матиме вигляд

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
     Button1.Caption:=' Pressed ';
end;

end.
```

Відмітимо, що частина “заготовок” для підпрограм створюється автоматично при встановленні якоїсь події (наприклад події, що відбувається при натисканні на клавішу) для об'єкту форми. У нашому прикладі червоним кольором відмічена частина, що написана “вручну” програмістом, решта створюється автоматично. Крім того, для використання подій, наприклад кнопок, спочатку їх потрібно розташувати у формі.

Оператори та функції.

Оператор begin end.

Першим оператором, без якого жодна програма на Delphi не буде працювати – це оператор, який складається з двох службових слів

```
begin        end
```

Алгоритмічна частина кожної програми чи підпрограми знаходиться між begin та end. У кінці програми після цього оператору ставиться крапка “.”, у підпрограмах – крапка з комою “;”.

Виходячи з того, що пара begin – end є одним оператором, то досить часто його ще застосовують у циклах, умовних операторах та ін., коли синтаксично виконується тільки один оператор, а за алгоритмом необхідно виконати кілька. Тоді перелік операторів розташовують між begin та end. Формально виконується у цьому випадку тільки один оператор begin-end, а насправді для виконання цього оператору “доводиться” виконувати усі, що розташовані між begin та end.

Оператори відділяються один від одного символом “;”.
Крім того програма чи модуль може містити у собі коментарі, які обмежуються символами “{” “}”.

Робота з текстовими файлами.

Delphi, як і попередні версії Pascal, дозволяє читати та записувати у файли інформацію. Це стосується і текстових файлів. Delphi дозволяє відкривати файл або для читання, або для запису. Якщо файл відкривається для читання з нього інформації, то вся інформація у ньому зберігається. Якщо ж файл відкривається (створюється) для запису, то інформація у ньому знищується, якщо такий файл вже існує.

Для зручності, вважатимемо, що у нас для прикладів роботи з файлами, описані такі змінні

```
var
    s, s1: string;
    f: textfile;
```

Процедура assignfile.

Перш ніж починати роботу з текстовим файлом у Delphi, треба підготувати його до цього. Спочатку необхідно описати змінну типу textfile у блоці опису змінних і поставити у відповідність цій змінній ім'я файлу на диску. Це можна зробити за допомогою процедури assignfile. Синтаксис цієї процедури має вигляд:

assignfile*(текстова_змінна, ім'я_файлу);*

де *ім'я_файлу* – рядкова змінна, у якій вказується повний шлях до файлу та його ім'я. Після цього – готуємо файл для читання або запису.

Приклад.

```
s:='c:\a.txt';
assignfile(f,s);
```

або

```
assignfile(f,'c:\a.txt');
```

Процедура reset

Підготовка файлу для читання. Синтаксис процедури має вигляд:

reset*(текстова_змінна);*

Відмітимо, що файл, який відкривається для читання, має існувати на диску.

Приклад.

```
s:='c:\a.txt';
assignfile(f,s);
reset(f);
```

або

```
assignfile(f,'c:\a.txt');
reset(f);
```

Процедура rewrite

Підготовка файлу до запису. Синтаксис процедури має вигляд:

rewrite*(текстова_змінна);*

Відмітимо, що файл, який відкривається(створюється) для запису, може не існувати на диску. У цьому випадку він буде створений.

Приклад.

```
s:=’c:\a.txt’;
assignfile(f,s);
rewrite(f);
```

або

```
assignfile(f,’c:\a.txt’);
rewrite(f);
```

Процедура closefile

Закривання файлу після використання rewrite або reset. Цю процедуру обов'язково потрібно встановлювати у програмі, коли ми працюємо з файлами. У противному випадку, особливо коли ми записуємо інформацію у файл, може бути втрата інформації або файл неможливо буде відкрити другий раз для читання або запису.
Синтаксис процедури має вигляд:

closefile*(текстова_змінна);*

Приклад.

```
s:=’c:\a.txt’;
assignfile(f,s);
rewrite(f);
closefile(f);
```

або

```
assignfile(f,’c:\a.txt’);
rewrite(f);
closefile(f);
```

Оператори readln та writeln

При роботі з файлами як для читання, так і для запису, необхідно застосовувати оператори, які і здійснюють ці дії. До таких і відносяться оператори readln та writeln.

Readln

Зчитування рядкової інформації з файлів. Синтаксис оператора:

readln(*текстова_змінна, рядкова_змінна*);

Приклад.

```
s:=’c:\a.txt’;
assignfile(f,s);
```

```
reset(f);
readln(f,s1);
close(f);
```

або

```
assignfile(f,'c:\a.txt');
reset(f);
readln(f,s1);
close(f);
```

У цьому прикладі відкривається для читання файл a.txt, що розташований на диску С, зчитується з нього перший рядок у змінну s1 та закривається файл.
Більш повний приклад описаний у розділі "Приклад читання тексту з файлу з використанням repeat - until.".

Writeln

Запис рядкової інформації у файли. Синтаксис оператора:

writeln(*текстова_змінна, рядкова_змінна*);

Приклад.

```
s1:='Текстовий рядок';
s:='c:\a.txt';
assignfile(f,s);
rewrite(f);
writeln(f,s1);
close(f);
```

або

```
s1:='Текстовий рядок';
assignfile(f,'c:\a.txt');
rewrite(f);
writeln(f,s1);
close(f);
```

У цьому прикладі відкривається (або створюється якщо такого файлу на диску не існує) для запису файл a.txt з диску С, записується текстовий рядок s1 (у нашому випадку значення s1 є 'Текстовий рядок', тобто у файлі a.txt буде записано один рядок з текстом *Текстовий рядок*) та закривається файл.

Більш повний приклад описаний у розділі "Приклад використання умовного оператору if для запису інформації у текстовий файл.".

Умовний оператор IF.

Оператор if використовується, якщо у алгоритмі у залежності від стану якогось процесу потрібно виконати ту чи іншу дію.
Повна версія оператору має вигляд:

if умова *then* оператор1 *else* оператор2 ;

Тобто "якщо виконуються умова, то виконувати оператор1, якщо ні – то оператор2".
Бувають випадки, коли у алгоритмі вимагається умовний оператор "якщо виконуються умова, то виконувати оператор1". Тоді його вигляд буде:

if умова *then* оператор1;

Зазначимо, що перед else символ ";" ніколи не ставиться.

Виходячи з вигляду оператору слід відзначити, що за виконання умови чи ні може виконуватись тільки один оператор (оператор1 чи оператор2). Якщо необхідним є виконання кількох операторів, то слід використовувати складний оператор begin – end. Тоді всі необхідні команди будуть розташовуватись між begin та end, як це зазначалось вище.

Наприклад

```
if  a>max  then  max:=a  else  max:=b;

if a[i]>a[i+1] then
      begin
            a[i]:=a[i]+a[i+1];
            a[i+1]:=a[i]-a[i+1];
            a[i]:=a[i]-a[i+1];
      end;
```

У наведених прикладах використовується проста умова, тобто умова, що використовує тільки одне порівняння. Оператор if допускає ще і складні умови. Тоді вони записуються у дужках ( ). Наприклад:

```
if  ( a>b and b>c)  then  midd:=b ;
```

Нехай нам потрібно написати програму, яка при натисканні на клавішу Button1, зчитує з рядку редагування Edit1 текст і, якщо цим текстом є рядок "автотекст", то надпис на клавіші міняється на "введіть свій текст", у противному разі – на клавішу виводиться текст, введений користувачем у Edit1, і у Edit1 виводиться текст "автотекст".

Для цього нам спочатку треба на формі розташувати два об'єкти: кнопку Button1 та рядок редагування Edit1. Після цього встановити головну подію для кнопки Button1 подвійним натисканням на неї. Далі, у автоматично згенерованій процедурі
procedure TForm1.Button1Click

дописати алгоритмічну частину, яка матиме вигляд

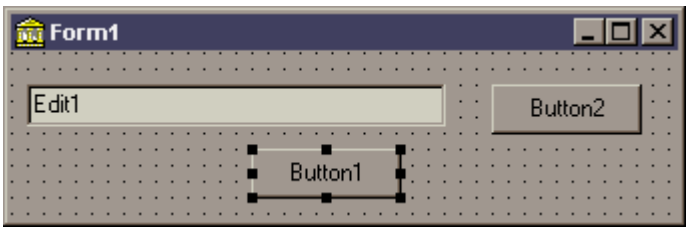
```
procedure TForm1.Button1Click(Sender: TObject);
begin
        if Edit1.Text=’автотекст’
                then   Button1.Caption:=' введіть свій текст '
                else
                        begin
                                Button1.Caption:=Edit1.Text;
                                Edit1.Text:= ‘автотекст’;
                        end;
end;
```

Приклад використання умовного оператору if для запису інформації у текстовий файл.

Написати програму, що створює на диску C файл a.txt і заповнює його рядками, введеними з Edit1 поки не буде введено рядок END.

Для написання програми потрібно розташувати на формі об'єкти Button1, Button2 та Edit1.
Так як ім'я файлу нам відомо завчасно, то не будемо використовувати об'єкт SaveDialog.

Форма матиме вигляд:



Визначимо головну подію для клавіші Button1. Ця клавіша буде розпочинати роботу програми, тобто ми організовуємо роботу програми таким чином, щоб спочатку у робочому вікні програми була видимою лише клавіша Button1, а по натисканню її Button1 зникала з екрану, а Button2 та Edit1 – виводились. Спочатку встановимо початкові значення параметрів visible для об’єктів форми. Для цього звернемось до Object inspector і послідовно для всіх об’єктів (Button1, Button2 та Edit1) встановимо значення таким чином:
Button1.Visible – true
Button2.Visible – false
Edit1.Visible - false

Для створення відповідної процедури досить двічі клацнути на Button1. Отримаємо

```
procedure TForm1.Button1Click(Sender: TObject);
begin
```

```
end;
```

Далі потрібно відкрити для запису (створити) файл c:\a.txt. Для цього опишемо глобальні (тобто ті, що можуть використовуватись у будь-якій підпрограмі даної програми) змінні таким чином:

```
var
  f:textfile;
  s:string;
```

та опишемо оператори, що відкривають файл на запис. Фрагмент програми матиме вигляд:

```
var
  f:textfile;
  s:string;
procedure TForm1.Button1Click(Sender: TObject);
begin
  assignfile(f,'c:\a.txt');
  rewrite(f);
end;
```

У цій же ж підпрограмі має зникати з екрану клавіша Button1, Button2 та Edit1 – з’являтись. Тому текст підпрограми матиме вигляд:

```
var
  f:textfile;
  s:string;
procedure TForm1.Button1Click(Sender: TObject);
begin
  assignfile(f,'c:\a.txt');
  rewrite(f);
  button2.visible:=true;
  edit1.visible:=true;
  button1.visible:=false;
end;
```

Наступним кроком є встановлення головної події для клавіші Button2, де має відбуватись зчитування з Edit1 тексту та запису його у файл. Це виглядає таким чином:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  s:=Edit1.Text;
  writeln(f,s);
  closefile(f);
end;
```

Але зчитування повинно проводитись поки не буде набрано рядок END, тому необхідним є використання умовного оператору if, і текст процедури матиме вигляд

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  s:=Edit1.Text;
  if s<>'END' then
        writeln(f,s)
  else
      closefile(f);

end;
```

тобто поки введений рядок тексту не є END, то здійснюється запис цього рядка у файл, а у противному випадку – файл закривається.
Логічно додати сюди ще дії, у яких клавіша Button2 та рядок Edit1 стають невидимими, а клавіша Button1 - видимою:

```
procedure TForm1.Button2Click(Sender: TObject);
begin
  s:=Edit1.Text;
  if s<>'END' then
        writeln(f,s)
else
  begin
      closefile(f);
      button1.visible:=true;
      edit1.visible:=false;
      button2.visible:=false;
  end;
end;
```

Таким чином, при запуску написаної програми, на екран виводиться вікно

Далі, після натискання на клавішу Button1, отримаємо

і вводимо текст у рядок Edit1, підтверджуючи натисканням на клавішу Button2. Як тільки буде введено рядок END і натиснуто на Button2, файл c:\a.txt буде створено і на екран виведеться вікно

Повний текст програми має вигляд:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    Edit1: TEdit;
    Button2: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
var
   f:textfile;
   s:string;
procedure TForm1.Button1Click(Sender: TObject);

begin
   assignfile(f,'c:\a.txt');
   rewrite(f);
   button2.visible:=true;
```

```
    edit1.visible:=true;
    button1.visible:=false;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
 s:=Edit1.Text;
   if s<>'END' then writeln(f,s)
   else
      begin
        closefile(f);
        button1.visible:=true;
        edit1.visible:=false;
        button2.visible:=false;
      end;

end;

end.
```

Оператор вибору CASE.

Синтаксис:

```
case змінна of
        значення1_змінної :     оператор1;
        значення2_змінної :     оператор2;
        значення3_змінної :     оператор3;
        значення4_змінної :     оператор4;
        значення5_змінної :     оператор5;
        else    оператор6;
end;
```

де *значення1_змінної* є значенням(и) змінної *змінна* і може бути як одним значенням, так і кількома, перерахованими через кому “,”.

Оператор IF зручний, коли перевіряється кілька множинних умов, тобто умов, коли порівнюються змінні. У випадку, коли у залежності від значення однієї змінної (виразу) треба виконати ту чи іншу дію, особливо якщо треба перевірити кілька значень цієї змінної, логічно використовувати оператор вибору CASE. Наприклад виходячи зі значення цілої змінної і, де і змінюється від 1 до 5, вивести на екран у Label1 відповідний кожному значенню текст.
З використанням оператору IF цей фрагмент програми матиме вигляд

```
if  i=1  then  label1.caption:=’Текст 1’
      else  if  i=2  then label1.caption:=’Текст 2’
            else  if  i=3  then label1.caption:=’Текст 3’
                  else  if  i=4  then label1.caption:=’Текст 4’
                        else  if  i=5  then label1.caption:=’Текст 5’
```

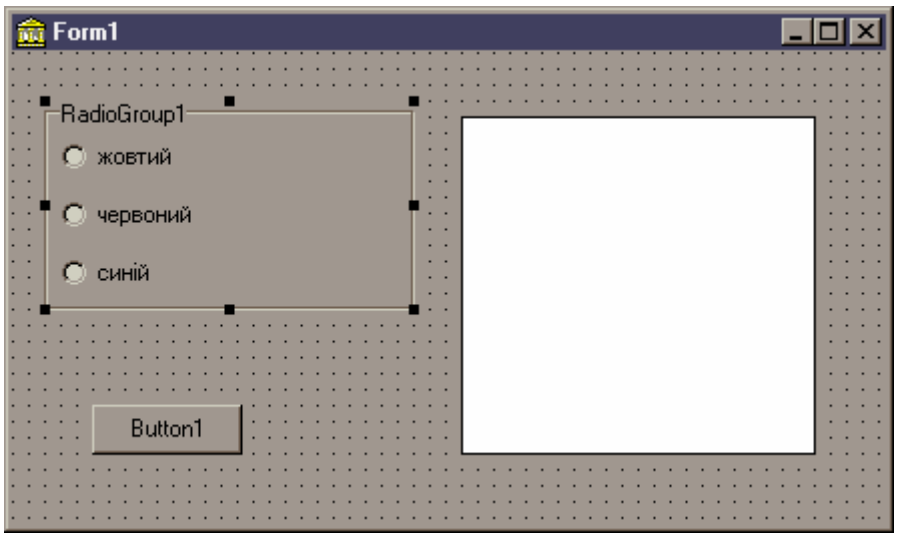
З оператором CASE цей фрагмент має вигляд

```
case i of
        1: label1.caption:='Текст 1';
        2: label1.caption:='Текст 2';
        3: label1.caption:='Текст 3';
        4: label1.caption:='Текст 4';
        5: label1.caption:='Текст 5';
end;
```

Приклад використання Case та об'єкту RadioGroup для встановлення кольору Shape.

Напишемо програму, яка у залежності від вибору кольору у об'єкті RadioGroup1, при натисканні на клавішу Button1 встановлює колір прямокутника Shape1.

Спочатку створимо форму і розташуємо на ній об'єкти RadioGroup1, Button1, Shape1 та заповнимо RadioGroup1 значеннями “ жовтий, червоний та синій ”таким чином:



Текст програми має вигляд:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
```

```
    RadioGroup1: TRadioGroup;
    Shape1: TShape;
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

end.
```

Відмітимо, що цей текст створений Delphi автоматично виходячи з присутності усіх об'єктів на формі.

Так як подія встановлення кольору відбувається при натисканні на клавішу Button1, то вкажемо для об'єкту Button1 головну подію двічі клацнувши лівою клавішею мишки на об'єкт Button1 у формі. Отримаємо:

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    RadioGroup1: TRadioGroup;
    Shape1: TShape;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation
```

```
{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin

end;

end.
```

Тобто автоматично була оголошена та створена порожня процедура *procedure TForm1.Button1Click*, у якій і буде алгоритмічно реалізована подія, що відбудеться при натисканні на клавішу Button1 при роботі програми. Опишемо цю подію.

procedure TForm1.Button1Click(Sender: TObject);	заголовок процедури
begin	початок тіла процедури
case Radiogroup1.Itemindex of	вибрати значення Itemindex з
0: Shape1.Brush.Color:=clYellow;	якщо 0, то колір пензля – жовтий
1: Shape1.Brush.Color:=clRed;	якщо 1, то червоний
2: Shape1.Brush.Color:=clNavy;	якщо 2, то синій
end;	кінець вибору case
end;	кінець процедури

Повний текст програми має вигляд

```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ExtCtrls, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    RadioGroup1: TRadioGroup;
    Shape1: TShape;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;
```

```
implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
begin
  case Radiogroup1.Itemindex of
    0: Shape1.Brush.Color:=clYellow;
    1: Shape1.Brush.Color:=clRed;
    2: Shape1.Brush.Color:=clNavy;
  end;
end;

end.
```

Робоче вікно програми має вигляд

Функція random

Функція random використовується для вибору значення цілого випадкової змінної, розподіленого за рівномірним розподілом. Синтаксис:

ціла_змінна:=random(*ціле_число*)

де *ціла_змінна* – змінна цілого типу, у яку буде записано випадкове ціле значення, *ціле_число* – верхня границя випадкових значень.

Після запуску цієї функції цілій змінній буде присвоєно випадкове значення у діапазоні від 0 до *ціле_число*-1. Детальніше використання цієї функції розглянуто у “Приклад використання циклу for to do для сортування числового масиву (бульбашкове сортування)”.

Алгоритм перестановки значень двох змінних.

Нехай маємо дві однотипні змінні (наприклад типу real). Нам потрібно переставити їх значення місцями. Для цього існує два шляхи: з використанням додаткової змінної аналогічного типу та без.

Перестановка двох значень змінних з використанням додаткової змінної.

Нехай у нас є дві змінні i,j:real, причому i=2.456, j=3.1416. Для того, що переставити значення цих двох змінних задіємо ще одну змінну k теж типу real. Тоді сам алгоритм матиме вигляд:

Дія	i	j	k
	2.456	*3.1416*	0
k:=i	2.456	3.1416	2.456
i:=j	3.1416	3.1416	2.456
j:=k	*3.1416*	**2.456**	2.456

Таким чином ми переставили значення двох змінних місцями.

Перестановка двох значень змінних без використання додаткової змінної.

Цей метод можна виклористовувати, якщо сума значень змінних не перевищує допустиме значення для даного типу змінних. Наприклад, якщо міняємо значення двох цілих змінних i=30000 та j=25000, то їх сума буде 55000, що перевищує максимально допустиме значення 32767, то використання цього алгоритму є неможливим.

Розглянемо сам алгоритм.
Нехай у нас є дві змінні i,j:real, причому, як і у попередньому прикладі, i=2.456, j=3.1416.

Дія	i	j
	2.456	*3.1416*
i:=i+j	5,5976	3.1416
j:=i-j	5,5976	2.456
i:=i-j	*3.1416*	**2.456**

Цикли.

Цикли використовуються у випадку, коли якась дія або група дій повторюється певну кількість раз. Кожне виконання усіх дій, що вказані у циклі називається ітерацією. Кожен цикл повинен мати умову, за якої він закінчує свою роботу. Цикли можуть бути з фіксованою кількістю ітерацій та ні. До циклів з фіксованою кількістю ітерацій (тобто зразу вказується їх кількість) відноситься цикл For – to – do. До інших відносяться цикли while – do та repeat – until.

Відмітимо, що при роботі з циклами слід “вручну” перевіряти принаймні початкову та кінцеву ітерації циклу.

Цикл for - to - do.

Як вже зазначалось вище, цикл for – to - do відноситься до циклів зі скінченною фіксованою кількістю ітерацій. Внаслідок цього для роботи циклу необхідна змінна, яка буде "відповідати" за номер ітерації, тобто індикатор циклу. Індикатором циклу може бути довільна перелікова, інтервальна, ціла, булевська чи символьна змінна. Найчастіше для індексу циклу використовується змінна цілого типу. Загальна структура циклу for - to - do має вигляд

for *інд_циклу:=початкове_значення_інд* to *кінцеве_значення_інд* do *оператор*;

де *інд_циклу* – індикатор циклу, - початкове значення індексу циклу, - кінцеве значення індексу циклу. Причому *початкове_значення_інд* > *кінцеве_значення_інд.*
У цьому випадку значення індикатору зростає від *початкове_значення_інд* кожної ітерації автоматично зростаючи на одиницю поки не досягне *кінцеве_значення_інд*. Тобто є **початкова умова циклу**, де змінній *інд_циклу* примусово присвоюється початкове значення *початкове_значення_інд* та **умова завершення роботи циклу** коли *інд_циклу* досягає значення *кінцеве_значення_інд*.

Не рекомендується у тілі циклу міняти самостійно значення *інд_циклу*.

Ще одна модифікація циклу for – to – do має вигляд for – downto – do. У цій модифікації індекс циклу

for *інд_циклу:=початкове_значення_інд* downto *кінцеве_значення_інд* do *оператор*;

де *інд_циклу* – індикатор циклу, - початкове значення індексу циклу, - кінцеве значення індексу циклу. Причому *початкове_значення_інд* < *кінцеве_значення_інд.*

Приклад 1.
Потрібно порахувати суму усіх цілих чисел у діапазоні від 1 до 50.
Для цього нам потрібно 2 цілих змінних (наприклад і та s): одна (i) відповідатиме за індекс циклу, а друга (s) - за суму.
Початкове значення суми 0. Фрагмент програми матиме вигляд:

```
s:=0;
for i:=1 to 50 do
        s:=s+i;
```

Зазначимо, що ці змінні і та s мають бути описаними у блоці опису змінних var.

Приклад 2.
Потрібно вивести у ListBox1 послідовні суми чисел від 1 до 50. Тобто 1, 1+2, 1+2+3,..., 1+2+3+...+50.
Для цього потрібно також 2 цілих змінних (наприклад і та s): одна (i) відповідатиме за індекс циклу, а друга (s) - за суму.

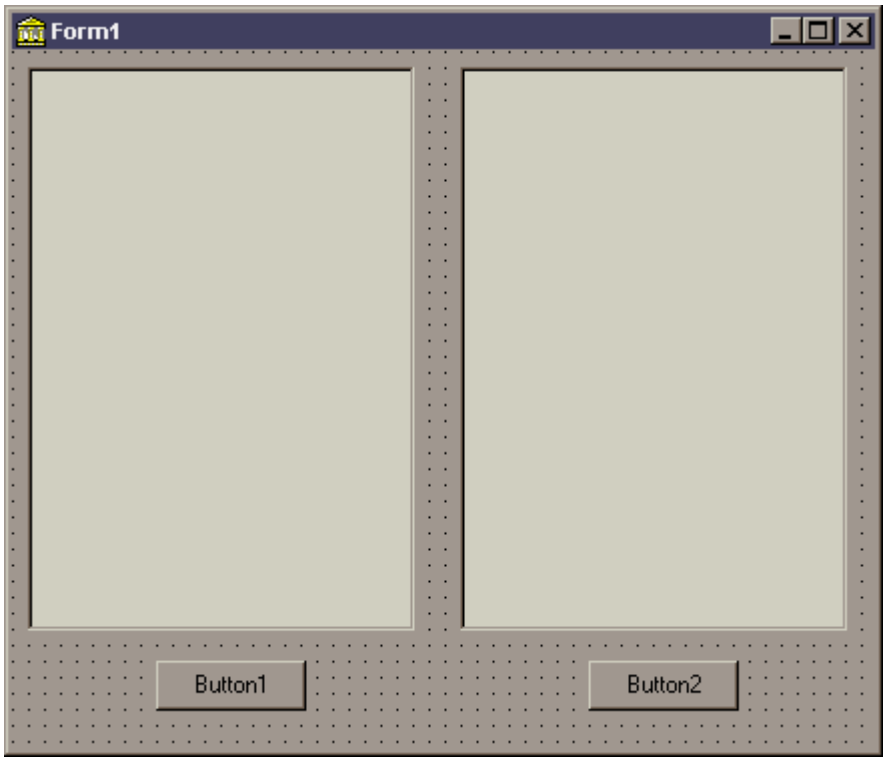
У тілі циклу повинна обраховуватись сума та додаватись рядок до ListBox1, тому ми вимушені будемо використати оператор begin end. Початкове значення суми 0. Фрагмент програми матиме вигляд:

```
s:=0;
for i:=1 to 50 do
        begin
                s:=s+i;
                ListBox1.Items.Append(inttostr(s));
        end;
```

Приклад використання циклу for to do для сортування числового масиву (бульбашкове сортування)

Потрібно згенерувати випадковим чином значення масиву зі 100 цілих чисел, кожне число має бути у діапазоні від 0 до 16, вивести максив на екран у ListBox1, відсортувати значення елементів масиву за зростанням і вивести їх у ListBox2.

Спочатку розташуємо об'єкти на формі таким чином:



Далі, розіб'ємо виконання алгоритму на дві частини: клавіша Button1 відповідатиме за заповнення масиву значення і виведення його на екран не сортованим у ListBox1. Клавіша Button2 відповідатиме за сортування масиву та виведення його сортованим у ListBox2.

Крім того нам знадобляться такі глобальні змінні:

```
var
  a:array [1..100] of integer;
  i,j:integer;
```

Сформулюємо спочатку головну подію для клавіші Button1. По натисканню цієї клавіші спочатку у нас має заповнюватись масив цилими випадковими чисоами у діапазоні від 0 до 16, тому ця частина підпрограми матиме вигляд:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  for i:=1 to 100 do
     begin
        a[i]:=random(17);
     end;
end;
```

Крім того значення елемнтів масиву мають виводитись у ListBox1. Отримаємо:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  for i:=1 to 100 do
     begin
        a[i]:=random(17);
        ListBox1.Items.Append(inttostr(a[i]));
     end;
end;
```

У ListBox1.Items.Append(inttostr(a[i])); ми використовуємо функцію inttostr через те, що елемент масиву а має цілий тип, а додати до ListBox можна тільки рядок (або інакше рядкову змінну). Тому нам і доводиться конвертувати ціле значення у рядкове, використовуючи inttostr.

Далі опишемо головну подію для клавіші Button2. Спочатку тут масив має сортуватись за зростанням (тобто елементи масиву мають бути переставлені таким чином, щоб на початку масиву знаходилось найменше значення з елементів масиву, потім більше і т.д. аж до останнього елементу, де знаходиться найбільше значення). Сформулюємо алгоритм бульбашкового сортування.

Алгоритм бульбашкового сортування.

Нехай маємо послідовність, наприклад, для простоти викладення, з 5 чисел

2 5 1 0 6

записану у масив А, тобто a[1]=2, a[2]=5, a[3]=1, a[4]=0 і a[5]=6.
Нашою задачею є переставити ці числа за зростанням, тобто таким чином, щоб на першому місці було найменше число, за ним більше і т.д. останнім має стояти найбільше число.

Алгоритм бульбашкового сортування діє таким чином: послідовно порівнюються сусідні елементи послідовності (масиву) і, якщо те, що розташоване лівіше, є більшим за те, що

розташоване правіше – їх значення переставляють місцями. Так треба перевірити всі послідовні сусідні пари у масиві. У нашому випадку це матиме вигляд:

Стан масиву до перевірки					Елементи, які порівнюються	Результат порівняння	Стан масиву після перевірки та перестановки				
2	***5***	1	0	6	a[1] > a[2]	ні	2	5	1	0	6
2	***5***	***1***	0	6	a[2] > a[3]	так	2	**1**	**5**	0	6
2	1	***5***	***0***	6	a[3] > a[4]	так	2	1	**0**	**5**	6
2	1	0	***5***	***6***	a[4] > a[5]	ні	2	1	0	5	6

Таким чином на першому етапі застосування бульбашкового методу сортування ми "виставили" найбільший елемент масиву на своє місце (тобто у хвіст масиву). Наступним кроком є перевірка вже не всіх чотирьох сусідніх послідовніх пар елементів, а лише перших трьох (останній елемент вже є найбільшим і перевіряти його нема потреби). Отже:

Стан масиву до перевірки					Елементи, які порівнюються	Результат порівняння	Стан масиву після перевірки та перестановки				
2	***1***	0	5	6	a[1] > a[2]	так	**1**	**2**	0	5	6
1	***2***	***0***	5	6	a[2] > a[3]	так	1	**0**	**2**	5	6
1	0	***2***	***5***	6	a[3] > a[4]	ні	1	0	2	5	6

Тобто на другому етапі ми визначили передостанній елемент масиву.
Відповідно на наступному кроці отримаємо

0 1 2 5 6

і на останньому

0 1 2 5 6

тепер формально запишемо перший етап, вважаючи, що у нас є ще ціла змінна i:

```
for i:=1 to 4 do
      if a[i]>a[i+1] then

        begin
          a[i]:=a[i]+a[i+1];
          a[i+1]:=a[i]-a[i+1];
          a[i]:=a[i]-a[i+1];
        end;
```

Якщо значення поточної змінної, більше за значення наступної, то переставити значення місцями

Тут є перестановка значень двох змінних місцями. Див. "Перестановка двох значень змінних без використання додаткової змінної."

Наступна ітерація матиме вигляд

```
for i:=1 to 3 do
```

```
if a[i]>a[i+1] then

  begin
    a[i]:=a[i]+a[i+1];
    a[i+1]:=a[i]-a[i+1];
    a[i]:=a[i]-a[i+1];
  end;
```

Тобто всього ітерацій нам треба зробити чотири – для i=1..4, i=1..3, i=1..2 та i=1. Досягти цього можна ввівши ще одну цілу змінну, наприклад j і використати її для вкладеності циклу

```
for j:=1 to 4
        for i:=1 to 5-j do
                if a[i]>a[i+1] then

                  begin
                    a[i]:=a[i]+a[i+1];
                    a[i+1]:=a[i]-a[i+1];
                    a[i]:=a[i]-a[i+1];
                  end;
```

Відмітимо, що у прикладі у нас масив складається з 5 елементів а цикл для проходу по масиву ми організовуємо від 1 до 4. Це робиться через те, що ми порівнюємо на кожній ітерації поточний елемент масиву з наступним. Якби ми організували цикл від 1 до 5, то на останньому кроці відбувалося б порівнянн 5-го елементу з 5+1-м, тобто 6-м, а такого немає.

Алгоритм бульбашкового сортування маству за спаданням має такий самий вигляд що і за зростанням, тільки знак > у операторі if слід поміняти на <.

<u>Продовження прикладу використання циклу for to do.</u>

У нашому випадку сам фрагмент програми з реалізацією алгоритму бульбашкового сортування, має вигляд

```
for j:=1 to 99 do
   for i:=1 to 100-j do
      if a[i]>a[i-1] then
        begin
            a[i]:=a[i]+a[i+1];
            a[i+1]:=a[i]-a[i+1];
            a[i]:=a[i]-a[i+1];
        end;
```

і останнім кроком роботи програми є виведення відсортованого масиву у ListBox2. Цей фрагмент програми матиме вигляд

```
    for i:=1 to 100 do
     Listbox2.Items.Append(inttostr(a[i]));
```

Отже вся процедура (головна подія для Button2) має вигляд

```
procedure TForm1.Button2Click(Sender: TObject);
begin
    for j:=1 to 99 do
       for i:=1 to 100-j do
          if a[i]>a[i-1] then
            begin
               a[i]:=a[i]+a[i+1];
               a[i+1]:=a[i]-a[i+1];
               a[i]:=a[i]-a[i+1];
            end;

    for i:=1 to 100 do
     Listbox2.Items.Append(inttostr(a[i]));
end;
```

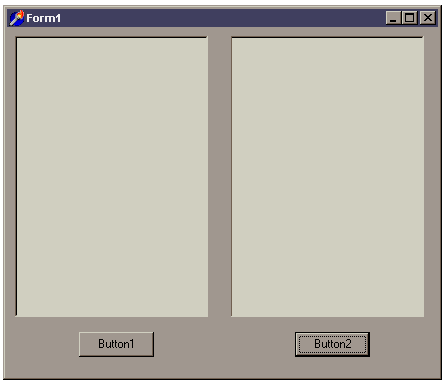
І нарешті, виходячи з того, що під час роботи програми ми можемо генерувати масив A довільну кількість разів і він буде виводитись у ListBox1 та сортований у ListBox2, то щоразу новозгенерований масив буде дописуватись у кінець ListBox1 та ListBox2 відповідно, тобто з часом у ListBox1 та ListBox2 почнеться плутанина. Щоб запобігти цьому досить просто очищати вміст ListBox1 та ListBox2 перед кожним записом масиву у них. Здійснити це легко за допомогою процедури ListBox1.Items.Clear та ListBox2.Items.Clear.

Весь текст програми матиме вигляд

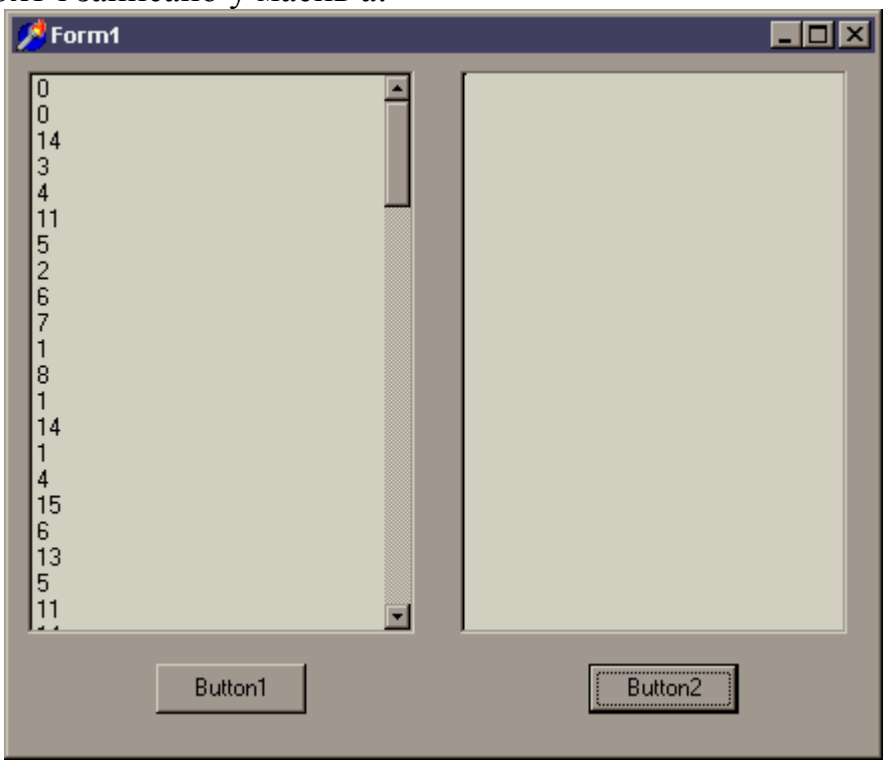
```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    ListBox1: TListBox;
    Button2: TButton;
    ListBox2: TListBox;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
```

```
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

var
   a:array [1..100] of integer;
   i,j:integer;

procedure TForm1.Button1Click(Sender: TObject);
begin
   ListBox1.Items.Clear;
   ListBox2.Items.Clear;
   for i:=1 to 100 do
      begin
          a[i]:=random(17);
          ListBox1.Items.Append(inttostr(a[i]));
      end;
end;

procedure TForm1.Button2Click(Sender: TObject);
begin
    ListBox2.Items.Clear;
     for j:=1 to 99 do
        for i:=1 to 100-j do
           if a[i]>a[i-1] then
              begin
                  a[i]:=a[i]+a[i+1];
                  a[i+1]:=a[i]-a[i+1];
                  a[i]:=a[i]-a[i+1];
              end;

     for i:=1 to 100 do
      Listbox2.Items.Append(inttostr(a[i]));
end;

end.
```

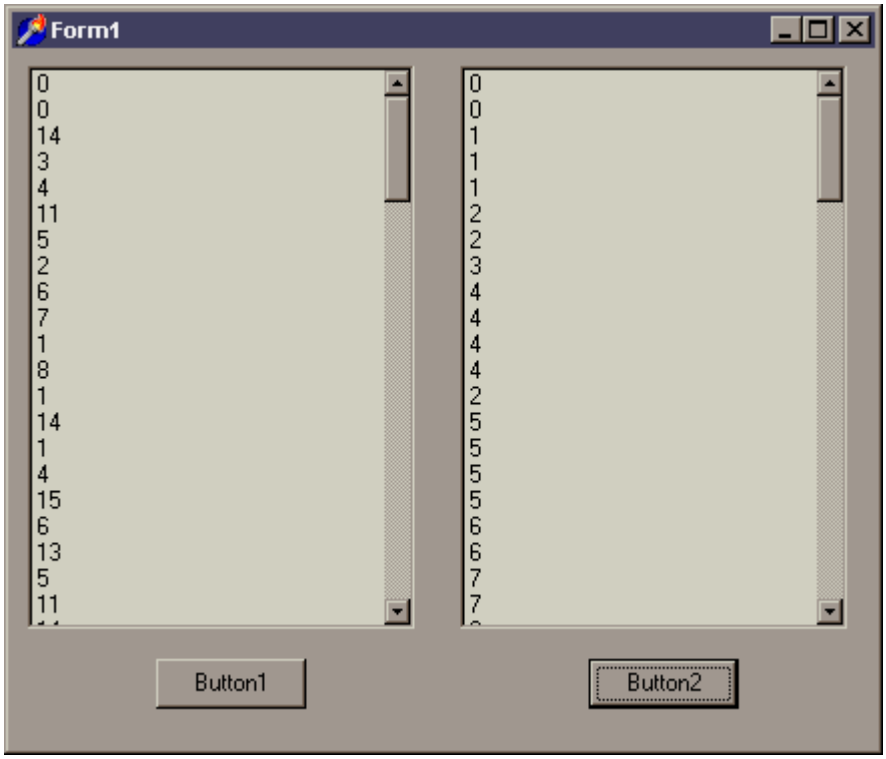
Розглянемо тепер роботу програми. Після її запуску на екран буде виведено вікно форми вигляду



Далі, після натискання на Button1, отримаємо згенерований масив зі 100 чисел, що виведено у ListBox1 і записано у масив a.



Після натискання на клавішу Button2 здійсюється сортування масиву a та виведення його у ListBox2.



Відмітимо, що так як ми використовуємо для генерування масиву функцію random, при кожному новому запуску програми або натисканню на клавішу Button1 ми отримаємо новий масив з іншими значеннями.

Цикл repeat – until.

На відміну від циклу for – to – do, у циклі repeat – until не вказується кількість ітерацій та початкова умова циклу. У цьому циклі є тільки кінцева **умова, по виконанню якої цикл завершує свою роботу**.
Загальна структура циклу repeat – until має вигляд

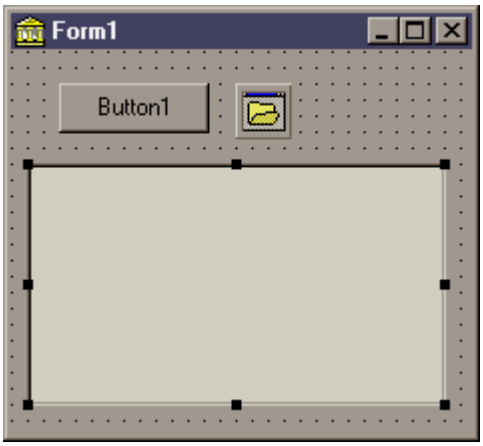
repeat
оператор;
оператор;
...
оператор;
until *умова_виходу*;

Використовуючи цей цикл слід формувати послідовність операторів та умову виходу з циклу таким чином, щоб умова виходу могла бути досягнутою. У противному випадку цикл буде нескінченним. Умовою виходу може бути довільна умова або змінна булевського типу.

Приклад читання тексту з файлу з використанням repeat - until.

Прочитати з текстового файлу a.txt, що розташований на диску c:\, рядки тексту і вивести їх на екран у ListBox.

Для програми потрібно буде розташувати на формі об'єкти Button1, OpenDialog1, ListBox1. Також знадобляться змінні: 2 рядкових, 1 текстова.
Форма матиме вигляд, наприклад



Далі, опишемо головну подію для об'єкту Button1. Для цього досить двічі клацнути мишкою у сам об'єкт. Отримаємо процедуру:

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

Опишемо змінні:

```
procedure TForm1.Button1Click(Sender: TObject);
var
        s, s1:string;
        f:textfile;
begin

end;
```

Далі, для того, щоб встановити ім'я файлу, треба отримати його з OpenDialog1 таким чином:

```
procedure TForm1.Button1Click(Sender: TObject);
var
        s, s1:string;
        f:textfile;
begin
        if OpenDialog1.Execute then s1:=OpenDialog1.FileName;
end;
```

Наступним кроком є підготовка файлу до читання.

```
procedure TForm1.Button1Click(Sender: TObject);
```

```
    var
        s, s1:string;
        f:textfile;
    begin
        if OpenDialog1.Execute then s1:=OpenDialog1.FileName;
        assignfile(f,s1);
        reset(f);
    end;
```

Тепер потрібно зчитувати з файлу по рядкам інформацію (наприклад у змінну s) і додавати ці рядки до ListBox1, причому робити це поки не досягнемо кінця файлу. Для цього нам доведеться організувати цикл (наприклад repeat - until). Текст процедури матиме вигляд:

```
    procedure TForm1.Button1Click(Sender: TObject);
    var
        s,s1:string;
        f:textfile;
begin
        if OpenDialog1.Execute then s1:=OpenDialog1.FileName;
        assignfile(f,s1);
        reset(f);
        repeat
            readln(f,s);
            ListBox1.Items.Append(s);
        until eof(f);

    end;
```

Але тепер, якщо при роботі програми двічі натиснути клавішу Button1 і вибрати файл, то зміст ListBox1 буде складатись з двох файлів, що були відкриті. Щоб запобігти цьому нам потрібно додати оператор очистки вікна ListBox1 - ListBox1.Items.Clear;. Тоді текст процедури матиме вигляд:

```
    procedure TForm1.Button1Click(Sender: TObject);
    var
        s,s1:string;
        f:textfile;
begin
        if OpenDialog1.Execute then s1:=OpenDialog1.FileName;
        assignfile(f,s1);
        reset(f);
        ListBox1.Items.Clear;
        repeat
            readln(f,s);
            ListBox1.Items.Append(s);
        until eof(f);
```

```
end;
```

І нарешті, після роботи з файлом, його потрібно закрити. Це робиться за допомогою оператора closefile.

Тоді повний текст модуля матиме вигляд:

```
unit Unit1;

interface

uses
      Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
      StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    OpenDialog1: TOpenDialog;
    ListBox1: TListBox;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
   s,s1:string;
   f:textfile;
begin
    if OpenDialog1.Execute then s1:=OpenDialog1.FileName;
    assignfile(f,s1);
    reset(f);
    ListBox1.Items.Clear;
    repeat
        readln(f,s);
        ListBox1.Items.Append(s);
```

```
    until eof(f);
    closefile(f);
end;
end.
```

Цикл while – do.

Як і для repeat – until у циклі while – do не вказується кількість ітерацій та початкова умова циклу. На відміну від repeat – until для циклу while – do вказується умова, **при якій працює, і, якщо умова не виконується, то цикл завершує свою роботу**. Тому логічно використовувати його або у тих випадках, коли вказати умови роботи циклу простіше, ніж у мову виходу з циклу (як у repeat – until), або коли спочатку потрібно перевірити умову, а потім виконувати оператори циклу.
Загальна структура циклу while – do має вигляд

```
while умова_роботи_циклу
    оператор;
```

або, якщо потрібно виконувати у циклі більш за один оператор

```
while умова_роботи_циклу
    begin
        оператор;
        оператор;
        ...
        оператор;
    end;
```

Використовуючи цей цикл слід формувати послідовність операторів та умову роботи циклу таким чином, щоб міг бути досягнутим момент, коли умова роботи циклу не виконається (тобто цикл закінчить свою роботу). У противному випадку цикл буде нескінченним. Умовою роботи циклу може бути довільна умова або змінна булевського типу.

Порівняння циклів.

Розглянемо кілька прикладів для порівняння доцільності використання того чи іншого циклу у програмі.

Обрахування суми фіксованої кількості елементів послідовності чисел.

Нехай у нас є послідовність з 20 дійсних чисел, записана у масив “a”, ціла змінна “i” та дійсна змінна “s”, у яку буде записана сума всіх чисел масиву “a”:

```
var
    a: array [1..20] of real;
```

```
        s : real;
        i : integer;
```

Потрібно обрахувати суму всіх елементів масиву “a”.

Розглянемо фрагменти програми з реалізацією алгоритму додавання елементів масиву для різних типів циклів:

for - to – do	repeat – until	while - do

s:=0;
for i:=1 to 20 do
  s:=s+a[i];
	s:=0;
	i:=0;
	repeat
	  i:=i+1;
	  s:=s+a[i];
	until i=>20;
		s:=0;
		i:=0;
		while i<=20 do
		  begin
		  i:=i+1;
		  s:=s+a[i];
		  end;

Отже у цьому прикладі більш доцільним є використання циклу for – to - do

<u>Обрахування суми нефіксованої кількості елементів послідовності чисел.</u>

Нехай у нас є послідовність з 100 дійсних чисел, записана у масив “a”, цілі змінні “i”, “j” та дійсна змінна “s”, у яку буде записана сума чисел масиву “a”:

```
var
        a: array [1..100] of real;
        j, i : integer;
        s: real;
```

Потрібно знайти суму та кількість послідовних елементів масиву “a” починаючи з першого такі, щоб сума елементів не перевищувала число 200.

Розглянемо тепер фрагменти програми з реалізацією алгоритму додавання елементів масиву для різних типів циклів:

for - to – do	repeat - until	while - do

s:=0;
for i:=1 to 100 do
  if s<=200 then
  begin
  s:=s+a[i];
  j:=i;
  end;
if s>200 then s:=s-a[j];
	s:=0;
	i:=0;
	repeat
	  i:=i+1;
	  s:=s+a[i];
	  j:=i;
	until ((s=>200) or (i=100));
	if s>200 then s:=s-a[j];
		s:=0;
		i:=0;
		while ((s<=200) and (i<=100)) do
		  begin
		  i:=i+1;
		  s:=s+a[i];
		  j:=i;
		  end;
		if s>200 then s:=s-a[j];

У цьому випадку використання усіх циклів є допустимим, хоча використання for - to – do є неєфективним через те, що можуть виконуватись багато зайвих операцій (якщо сума перевищить 200 на самому початку циклу).

Зчитування текстової інформації з файлу

У файлі c:\a.txt записана текстова інформація. Потрібно прочитати її та вивести на екран у ListBox1.

Для вирішення задачі нам потрібні будуть такі змінні:

```
var
  f: textfile;
  s:string;
```

Далі йде загальний фрагмент відкриття файлу для читання та очистка вікна ListBox1:

```
assignfile(f,'c:\a.txt');
reset(f);
ListBox.Items.Clear;
```

Розглянемо тепер фрагменти програми з реалізацією алгоритму додавання елементів масиву для різних типів циклів:

for - to – do	repeat - until	while - do

Через те, що загальна кількість ітерацій (у нашому випадку кількість рядків файлу c:\a.txt) невідома, цикл for – to – do використати неможливо (або досить складно).	repeat
	readln(f,s);
	ListBox1.Items.Append(s);
	until eof(f);
	closefile (f);
		while not eof(f) do
		begin
		readln(f,s);
		ListBox1.Items.Append(s);
		end;
		closefile (f);

У цьому прикладі використовувати можна тільки цикли repeat – until та while – do;

Зміст