ОЗНАЙОМЛЕННЯ ІЗ СЕРЕДОВИЩЕМ DELPHI

11-й клас

Валентина БОЙКО, Ольга ЗАГОРУЙКО, викладачі інформатики Технолого-економічного коледжу Білоцерківського НАУ, Київська обл.

Спілкуючись із колегами, ми не раз чули про те, що у процесі викладання предмета найбільшим дефіцитом інформативного наповнення є розділи, пов'язані з викладанням об'єктно-орієнтованого програмування. Лаконічність авторських сторінок підручника та зарозумілість технічних бестселерів не задовольняють потреб учителя. Відсутність практики роботи теж є істотним наріжним каменем цієї проблеми. Тож пропонуємо розробку однієї з перших практичних робіт, що містить вичерпний теоретичний блок і цікаву для дітей практичну частину. Ця розробка буде корисною вчителю не лише для проведення уроку в 11-му класі, а й у 8-му (за новим стандартом), оскільки це одна з тем нової програми з інформатики.

Практична робота № 5

Мета: сприяти освоєнню інтегрованого середовища розробки додатків Delphi; набути початкові навички у проектуванні Windows-додатків із використанням компонентів Delphi та у налаштуванні цих додатків.

Теоретичні відомості

1. Початкові відомості про інтегроване середовище розробки додатків Delphi

Після запуску Delphi на екрані з'являються п'ять вікон *(рис. 1)*:

1. Головне вікно — вгорі екрана.
2. Вікно **Конструктора форми** (порожня форма з ім'ям ***Form1***) під головним вікном праворуч.
3. Вікно **Редактора властивостей об'єктів** ліворуч унизу — ***Object Inspector.***
4. Вікно **Дерева об'єктів** під головним вікном ліворуч — ***Object TreeView.***
5. Вікно **Редактора коду** (позиціонується там, де і вікно **Конструктора форми,** перемикання за допомогою клавіші F12, Shift + F12 здійснює вибір однієї з кількох форм для перемикання) — ***Unit1.pas.***

Вікно **Редактора коду** майже повністю закрите вікном стартової форми. Це вікно можна перемістити, щоб бачити одночасно і вікно **Конструктора форми** (так зроблено на *рис. 1*).

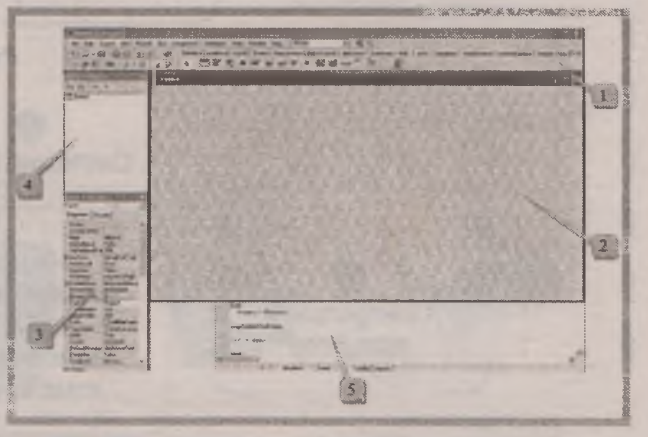


Рис. 1. Вигляд екрана після запуску Delphi

Головне вікно інтегрованого середовища Delphi (*далі* — IDE) має вигляд, що зображений на *рис. 2*. Вікно містить: рядок заголовка з іменем IDE (Delphi 7) та ім'ям проекту (за умовчанням Project 1); під ним рядок головного меню IDE; нижче головного меню ліворуч — панелі інструментів; праворуч від них — **Палітра компонентів,** що складається зі сторінок із закладками.

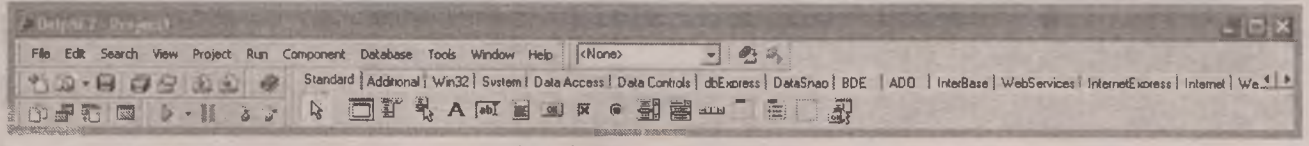
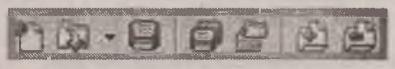
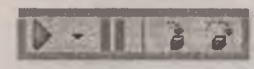


Рис. 2. Головне вікно IDE

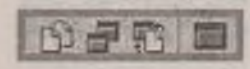
Панелі інструментів



Standard — Стандартна



Debug — Налагодження



View — Вид

Custom — Користувач

Палітра компонентів (Component Palette)

Пункти меню головного вікна IDE

Пункт		Призначення
File	Файл	Робота з файлами
Edit	Виправлення	Редагування
Search	Пошук	Пошук фрагментів
View	Вид	Вибір, що показувати
Project	Проект	Операції з проектом
Run	Запуск	Прогін і налагодження
Component	Компоненти	Робота з компонентами
Database	Бази даних	Робота з базами даних
Tools	Інструменти	Використання зовнішнього інструментарію
Windows	Вікно	Перелік вікон для вибору фокуса
Help	Допомога	Контекстна допомога

Вікно форми (Forml) — це проект Windows — вікна майбутньої програми. Програміст розміщує на формі потрібні компоненти з **Палітри компонентів.**

Вікно оглядача **Дерева об'єктів (Object TreeView)** призначене для наочного відображення зв'язків між окремими компонентами, розміщеними на активній формі або в активному модулі даних.

Вікно **Інспектора об'єктів (Object Inspector)** призначене для перегляду та редагування властивостей компонента. Воно містить дві вкладки: ***Properties*** (Властивості) та *Events* (Події). Сторінка *Властивості* слугує для установки потрібних значень властивостей компонента, сторінка *Події* дає змогу визначити реакцію компонента на певну подію.

Вікно **Редактора коду**, яке можна побачити, відсунувши вбік вікно форми, призначене для набору тексту програми. На початку роботи над новим проектом це вікно **Редактора коду** містить сформований Delphi початковий шаблон програми, який надалі доповнює програміст.

2. Склад проекту Delphi

Проект Delphi складається з форм, модулів, установок параметрів проекту, ресурсів тощо. Уся ця інформація розміщається у файлах. Більшість із цих файлів автоматично створюються Delphi.

До складу проекту належать такі файли (у дужках зазначено розширення імен файлів):

- файл проекту (dpr);
- файли опису форм (dfm);
- файли модулів і модулів форм (pas);
- файл параметрів проекту (dof);
- файл параметрів середовища (cfg);
- файл опису ресурсів (res);
- файл конфігурації вікон (dsk).

Файл проекту є центральним файлом додатка і становить програму. Це текстовий файл, що використовується для зберігання інформації про форми й модулі. У ньому містяться оператори ініціалізації й запуску програм на виконання. Відображення коду файла проекту у вікні Редактора коду задається за допомогою команди меню **Project | View Source** (Проект | Перегляд джерела).

Файл опису форми створюється Delphi автоматично під час конструювання форми й містить характеристики форми та її компонентів. У разі потреби можна відобразити цей файл у вікні Редактора коду в текстовому вигляді за допомогою команди **View as Text** (Переглянути як текст) контекстного меню форми, або сполучення клавіш **Alt + F12.** Повернення до візуального подання форми виконується за допомогою команди **View as Form** (Переглянути як форму) контекстного меню тексту файла опису форми, або сполучення клавіш **Alt + F12.**

Файл модуля. Кожній створюваній формі відповідає файл модуля, що використовується для зберігання коду. Можуть бути модулі без форм, але форми без модулів не існують. Зазвичай увесь код розміщується в модулях.

Файл ресурсів містить значки, растрові зображення і курсори.

Можуть також створюватися файли резервних копій.

Після компіляції створюється *виконуваний файл (exe)* з ім'ям проекту. Він є автономним виконуваним файлом, для якого більше нічого непотрібно, якщо лише не використовуються бібліотеки, що містяться в *dll, ocx* тощо, а також, якщо не використовується підтримка пакетів часу виконання.

Для кожного модуля після компіляції створюється об'єктний файл модуля *(dcu).* Це відкомпільований файл модуля *(pas),* що компонується в остаточний виконуваний файл.

3. Налаштування програми

Процес усунення помилок у програмі називається ***налаштуванням*** (debugging), а самі помилки в програмі досить часто називають ***жучками*** (bugs). Серед помилок, які допускаються в програмі, виокремлюють синтаксичні, помилки часу виконання і логічні.

Синтаксичні помилки — це помилки, пов'язані з порушенням ***синтаксису*** (тобто правил граматики) мови програмування. Прикладом такої помилки, що трапляється досить часто, є пропуск символу «крапка з комою», яким розділяються оператори в програмах, написаних мовою Delphi. У разі виявлення синтаксичної помилки компілятор видає ***повідомлення про помилку*** (Errors), указуючи її передбачуване місцезнаходження і пояснюючи можливий її зміст. Досить часто компілятор виводить натяки (Hints) і попереджувальні повідомлення (Warnings) про деякі дещо незвичні конструкції в програмі.

Для запуску процесу компіляції програми слід обрати пункт меню Project ► Compile Project (гаряча клавіша Ctrl + F9).

Можливим є також вибір пункту меню Run ►Run (гаряча клавіша F9, або клік мишею над інструментальною кнопкою ▶). В останньому випадку насправді йдеться про запуск програми на виконання. Але перед виконанням програми, якщо в її тексті здійснювалася корекція, будуть автоматично виконані компіляція та редагування зв'язків. У разі відсутності синтаксичних помилок та помилок у редагуванні зв'язків програма буде автоматично запущена на виконання, інакше їх потрібно усунути.

Можливий також безпосередній запуск редактора зв'язків після виконання компіляції або ж відразу. Для цього потрібно виконати команду Project ► Build Project.

Деякі помилки виявляються лише під час виконання програми, у зв'язку з чим вони отримали відповідну назву — ***помилки часу виконання*** (runtime errors). У термінології Delphi такі помилки називаються винятками (Exceptions). Із їхньою появою програма завершується аварійно з видачею пояснювального повідомлення. У середовищі поява таких помилок призводить до переривання виконання програми та виведення вікна з інформацією про вид виключення. Прикладами таких помилок часу виконання є ділення на нуль, одержання великих числових значень, які не можуть бути записані в комірку пам'яті (переповнення).

Часто найбільш неприємними є ***логічні помилки*** — помилки в алгоритмі, а також помилки, спричинені елементарною неуважністю (наприклад, використання в програмі операції * замість операції +, задавання неправильного числового значення, використання одного імені змінної замість іншого). Такі помилки компілятор найчастіше виявити не може (за винятком випадків, якщо вони призводять до порушення синтаксису). Крім того, логічні помилки можуть «не проявити себе» і під час виконання програми як помилки часу виконання.

Для виявлення логічних помилок здійснюється ***тестування*** програми, яке виражається в її запуску з кількома характерними наборами вхідних даних і перевірці відповідних їм результатів. При цьому в жодному разі не можна обмежуватися одноразовою перевіркою програми — мають бути відстеженні всі окремі випадки вхідних даних, із якими програма може зіткнутися. Лише після перевірки правильності функціонування програми на багатьох характерних

наборах даних можна отримати високу (але не абсолютну) гарантію її правильності.

Для виявлення більшості логічних помилок передбачений спеціальний засіб, що називається ***налаштувальник*** (Debugger).

Робота в режимі налаштування — це насамперед виконання програми по кроках із відстеженням послідовності виконання операторів програми та значень, які набувають змінні після виконання певного кроку.

Для виконання фрагмента програми по кроках можна використовувати такі команди меню:

- Run ▶ Trace Into (Трасування із заходженням у...) — покрокове виконання програми із заходженням у процедури та функції з подальшим покроковим виконанням їх рядків (аналогом є натискання клавіші **F7**, або клік мишею над інструментальною кнопкою);
- Run ▶ Step Over (По кроках без заходження у...) — покрокове виконання рядків програми, за якого виклик підпрограми вважається за одну команду, тобто вхід у неї не проводиться (аналогом є затискання клавіші **F8**, або клік мишею над інструментальною кнопкою);
- Run ▶ Trace to Next Source Line (Трасування до наступного рядка) — перехід до наступного рядка програми (аналогом є натискання сполучення клавіш **Shift + F7**);
- Run ▶ Run to Cursor (Виконати до курсора) — команда виконує програму до того виконуваного оператора, на якому розташований курсор у вікні **Редактора коду** (аналогом є натискання клавіші **F4**);
- Run ▶ Run Until Return (Виконати до виходу з функції) — виконання програми до виходу з поточної функції і зупинка на операторі, наступному за викликом цієї функції, із залишенням курсора в рядку, в якому було здійснене звертання до функції (аналогом є натискання сполучення клавіш **Shift + F8**);
- Run ▶ Show Execution Point (Показати точку виконання) — команда показує на екрані виконуваний рядок коду та переміщує на нього курсор.

Досить часто для налаштування програми використовуються так звані точки переривання. Щоб увести просту (безумовну) точку переривання, достатньо у вікні **Редактора коду** клікнути мишею на лівій межі вікна навпроти потрібного рядка коду. При цьому здійсниться забарвлення рядка, а навпроти нього на лівій межі вікна коду з'явиться червона кулька.

Якщо тепер запустити програму на виконання, то кожен раз, коли керування перейде до рядка, в якому вказана точка переривання, відбувається переривання виконання. Скасування точки переривання здійснюється за допомогою повторного кліку мишею на лівій межі вікна коду навпроти рядка з точкою переривання. Точку переривання можна встановити за допомогою команди Run ▶ Add Breakpoint ▶ Source Breakpoint із уведенням номера рядка, де має здійснюватися переривання.

Дія точки переривання дещо аналогічна до натискання клавіші F4, але перевага точок переривання полягає в тому, що можна одночасно вказати кілька таких точок у різних місцях коду і в різних модулях.

Програма в цьому разі виконується до першої точки переривання, яка трапиться під час виконання. Після аналізу проміжних результатів можна продовжити виконання програми, натиснувши інструментальну кнопку .

Точки переривання можна встановлювати лише на виконуваних операторах. Прибрати точку переривання можна, виконавши ті самі дії, що й виконувалися для її встановлення.

Для перегляду значення змінних у момент зупинки програми досить навести курсор миші на ім'я потрібної змінної в коді програми. Якщо ж є потреба у відстеженні стану відразу кількох змінних, то їх можна додати у вікно перегляду Watches List.

За умовчанням вікно перегляду стану змінних автоматично виводиться в режимі налаштування ліворуч від вікна коду. У разі відсутності потреби в цьому вікні його можна закрити. Щоб викликати раніше закрите вікно перегляду стану змінних, потрібно вибрати в головному меню опцію View ▶ Debug Windows ▶ Watches, або ж натиснути сполучення клавіш Ctrl + Alt + W.

Для додавання змінних у вікно Watches List достатньо здійснити подвійний клік мишею в цьому вікні і ввести назву змінної в рядок Expression вікна, яке відкриється після кліку.

За умовчанням у режимі налаштування ліворуч від вікна коду під вікном Watches List виводиться вікно локальних змінних Local Variables. У цьому вікні виводяться значення локальних змінних процедур і функцій (зокрема їх формальних параметрів).

У разі закриття вікна локальних змінних його повторне відкриття здійснюється за допомогою вибору в головному меню опції View ▶ Debug Windows ▶ Local Variables, або ж за допомогою натискання сполучення клавіш Ctrl + Alt + L.

Якщо вміст вікна локальних змінних визначається автоматично, то вміст вікна Watches List визначає програміст.

Розглянемо можливу послідовність дій із налаштування програми на прикладі задачі розробки програми відшукування коренів квадратного рівняння:

$ax^2 + bx + c = 0, a \neq 0.$

Далі окремі групи дій умовно поділимо на етапи з метою їх структуризації.

Етап 1. Конструювання форми

Завантажимо Delphi. Захоплюючи по черзі межі форми за натиснутої лівої кнопки миші, доберемо зручні розміри форми.

Далі слід розмістити на формі компоненти відповідно до вимог задачі. Задача передбачає наявність засобів із забезпечення введення трьох чисел, виведення результатів обчислень та безпосередньої реалізації процесу обчислень.

Проектування форми будемо здійснювати, орієнтуючись на *рис. 3*.

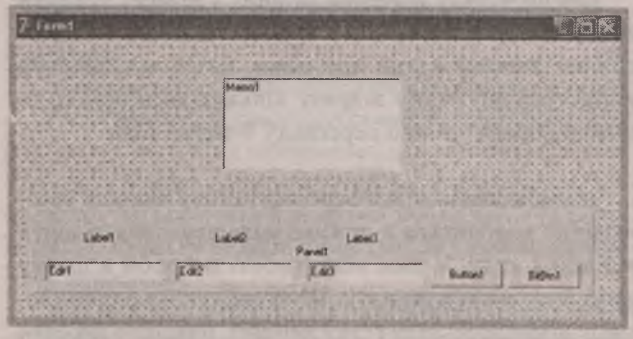


Рис. 3. Можливий початковий вигляд форми

Виберемо на вкладці Standard компоненти Memo, TPanel, три компоненти TEdit, три компоненти TLabel і компонент TButton, а на вкладці Additional — компонент TBitBtn. Розмістимо ці компоненти згідно з *рис. 3*.

Компонент TEdit (однорядкове редаговане текстове поле) застосовують для введення та (або) відображення досить довгих текстових рядків. Для зміни тексту, що міститься в ньому, достатньо змінити значення його властивості Text, що можна виконувати як на етапі проектування програми, так і за допомогою програмних засобів. Ми будемо використовувати три такі компоненти для введення трьох коефіцієнтів рівняння.

Компонент TLabel (Мітка) призначений для виведення різних повідомлень (підказок, результатів обчислень тощо). Ми будемо його використовувати для виведення підказок, що випереджають уведення (значення якого з коефіцієнтів потрібно набрати у відповідному редакторі).

Для зміни тексту, що міститься в цьому компоненті, досить змінити значення його властивості Caption, що також можна робити як на етапі проектування програми, так і за допомогою програмних засобів.

Компонент TButton (Кнопка) призначений для керування програмою. Ми вдамося до нього для організації обчислень і забезпечення виведення.

Компонент TBitBtn (Кнопка із зображенням) є різновидом кнопки TButton. Ми застосуємо один із варіантів цього компонента, який слугує для видачі сигналу про припинення роботи програми. Ця кнопка не є обов'язковою, оскільки припинення роботи програми забезпечується кліком над кнопкою закриття форми, або натисканням сполучення клавіш Alt + F4. Замість цього компонента можна використовувати і звичайну кнопку TButton із відповідним опрацьовувачем події OnClick.

Багаторядкове редаговане текстове поле TMemo застосуємо для виведення результату обчислень. Компонент TMeto вживають для уведення, редагування й (або) відображення досить довгих текстів, які можуть містити кілька рядків. Цей текст може корегуватися як на етапі візуального проектування, так і програмно, для чого потрібно змінювати властивість Lines (Рядки) цього компонента.

Відповідно до специфіки розв'язуваної задачі здійснимо такі зміни властивостей компонентів:

- Форма:

Position — poScreenCenter

Caption — Приклад

- Панель:

Align — alBottom

BevelOuter — bvNone

Caption — очистимо

- Мітки Label1, Label2 та Label3:

Caption — відповідно тексти Коефіцієнт a, Коефіцієнт b, Коефіцієнт c

- Редактори Edit1, Edit2 та Edit3:

Text — 0

- Багаторядкове поле:

Align — alClient

Lines — очистимо

- Кнопка Button1:

Caption — Обчислити

- Кнопка Close:

Kind — bkClose

У результаті форма набуде вигляду, зображеного на *рис. 4*.

Зміна значень властивостей буде автоматично приводити до зміни тексту модуля. У результаті текст модуля набуде такого вигляду:

```
unit Unit1;

interface

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

```
type
TForm1 = class(TForm)
Panel1: TPanel;
Edit1: TEdit;
Edit2: TEdit;
Edit3: TEdit;
Button1: TButton;
BitBtn1: TBitBtn;
Label1: TLabel;
Label2: TLabel;
Label3: TLabel;
Memo1: TMemo;
private
{ Private declarations }
public
{ Public declarations }
end;

var
Form1: TForm1;

implementation

{$R *.dfm}

end.
```

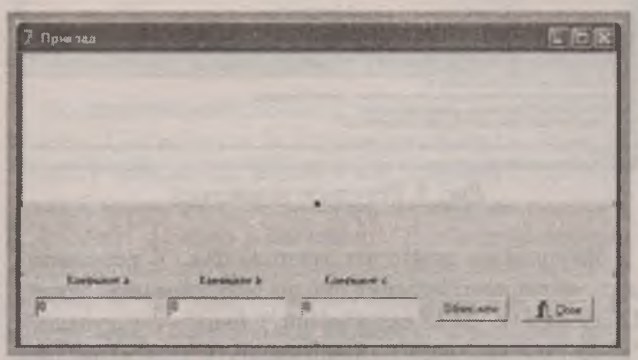


Рис. 4. Можливий остаточний вигляд форми

Як це видно з наведеного вище тексту модуля, Delphi автоматично вставила у клас розміщені на формі компоненти, надавши їм стандартні імена (якщо компонент під час проектування форми перейменовувати, змінюючи значення властивості Name, то відповідно буде змінюватися текст модуля).

Етап 2. Написання початкового коду

Насамперед перейдемо у вікно коду та впишемо в секції **private** опису класу форми такий рядок:

a, b, c: Real;

Тим самим ми опишемо змінні для позначення коефіцієнтів рівняння, указавши, що вони приймають дійсні значення.

Виконаємо подвійний клік мишею над зображенням форми у вікні Дерево об'єктів або над будь-яким вільним місцем форми у її вікні. У відповідь на клік мишею Delphi автоматично вставить у код модуля перед термінатором end із крапкою такий код:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
end;
```

Це код так званого опрацьовувача події OnCreate (За створенням) для форми.

Слово **procedure** вказує на те, що опрацьовувач є так званою підпрограмою-процедурою. Складене ім'я TForm1.FormCreate — це ім'я процедури, що складається з двох частин: імені класу TForml і власне імені процедури FormCreate. Після імені процедури у круглих дужках зазначено опис параметра, із яким вона буде викликатися (Sender: TObject). У нашій програмі параметр Sender використовуватися не буде. У більш складних програмах за допомогою цього параметра програміст може визначити, який компонент згенерував цю подію (у цьому випадку подія OnCreate). У загальному випадку у процедури може бути кілька параметрів; вони також можуть бути відсутніми. Перший рядок розглянутого фрагмента коду називається ***заголовком процедури*** (він завершується символом «крапка з комою»). Слідом за заголовком процедури розташовується її тіло (у цьому випадку воно порожнє й містить лише так звані операторні дужки begin та end). Щоб опрацьовувач події виконував які-небудь дії, його тіло має бути наповнене операторами. Відзначимо, що відразу ж за заголовком може розміщуватися секція описів, у якій програміст описує допоміжні елементи, потрібні для реалізації тіла процедури.

Крім вставки коду опрацьовувача події, Delphi автоматично модифікує опис класу, вставляючи в нього перед словом **private** заголовок опрацьовувача події:

```
procedure FormCreate (Sender: TObject);
```

Модифікуємо код опрацьовувача події OnCreate форми, надавши йому такого вигляду:

```
procedure TForm1.FormCreate(Sender: TObject);
begin
DecimalSeparator:='.';
end;
```

Якщо ви працюєте з русифікованою версією Windows, то в ній як роздільник цілої і дробової частин дійсного числа використовується не десяткова точка, яка застосовується в мовах програмування (зокрема у Delphi), а кома. Результатом цього може бути неправильна робота деяких підпрограм, що перетворюють рядки у формат дійсних чисел (наприклад, StrToFloat). Системна змінна DecimalSeparator містить символ, що використовується як роздільник цілої і дробової частин дійсних чисел. Єдиний оператор процедури TForml.FormCreate на початку роботи програми змінює цей символ на символ «крапка», що використовується в Delphi та в нерусифікованій версії Windows.

Клікнемо тепер мишею над кнопкою **Обчислити.** У відповідь на клік мишею Delphi автоматично вставить у код модуля перед термінатором end із крапкою такий код:

```
procedure TForm1.ButtonlClick(Sender: TObject);
begin
end;
```

Це код так званого опрацьовувача події OnClick (За кліком) для кнопки Обчислити. Узагалі в разі кліку мишею в програмі, що працює, виникає подія OnClick, що зв'язується з компонентом, над яким був здійснений клік.

Якщо опрацьовувач події відсутній, то подія ніяк не опрацьовується; за умови його наявності клік мишею активізує опрацьовувач і забезпечує виконання дій, записаних у цьому опрацьовувачі.

Крім того, в опис класу перед словом **private** буде автоматично вставлений заголовок опрацьовувача події:

```
procedure ButtonlClick(Sender: TObject);
```

Модифікуємо код процедури TForml.ButtonlClick, надавши йому такого вигляду:

```
procedure TForml.ButtonlClick(Sender: TObject);
begin
a:= StrToFloat(Editl.Text);
b:= StrToFloat(Edit2.Text);
a:= StrToFloat(Edit3.Text);
d:=b*b-4*a*a;
if d < 0
Memol.Lines.Add('The equation has no roots');
Memol.Lines.Add('xl = '
+ FloatToStr((-b + Sqrt(d)) / 2 * a)));
Memol.Lines.Add('x2 = '
+ FloatToStr((-b — Sqrt(d)) / 2 * a)));
end;
```

Етап 3. Виконання компіляції

Як це видно з *рис. 5,* компілятор виявив 3 синтаксичні помилки й 1 натяк та вивів відповідні пояснення. Червоним кольором виділено рядок, у якому вперше було знайдено помилку, а курсор розміститься за якимнебудь символом цього рядка.

Програміст повинен шукати синтаксичну помилку перед курсором за текстом програми, причому необов'язково в цьому ж рядку.

Помилки та попередження мають такий вигляд:

[Error] Unit1.pas(47): Undeclared identifier: 'd' — невизначений символ 'd';

[Warning] Unit1.pas(49): Comparing signed and unsigned types — widened both operands — порівняння знакового та беззнакового операндів — розширюються обидва операнди;

[Error] Unit1.pas(58): Statement expected but end of- file found — очікувався оператор, але знайдений кінець файла;

[Fatal Error] Project1.dpr(5): Could not compile used unit 'Unitl.pas' — компілятор не зміг зібрати модуль Unit1.pas'.

Перше повідомлення свідчить про те, що змінна d (для запису значення дискримінанта) використовується, але не визначена в програмі. Тому опишемо її в розділі implementation модуля Unitl:

```
var
d: Real;
```

Друга помилка — це помилка, зумовлена неправильним синтаксисом оператора **if:** у цьому операторі пропущено службове слово **then.** Впишемо це слово після умови d < 0.

Останнє повідомлення — це узагальнювальне повідомлення про наявність синтаксичних помилок.

Отже, ми виявили лише дві помилки.

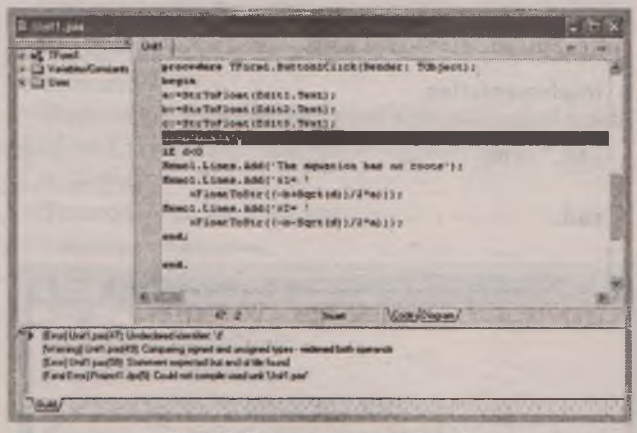


Рис. 5. Результат компіляції

Виконаємо повторну компіляцію. У результаті отримаємо повідомлення про ще одну помилку: [Error] Unitl.pas(51): 'END' expected but ')' found — очікувалося 'END', але знайдено')'. Якщо клікнути мишею над цим повідомленням, то курсор займе місце перед останньою закриваючою дужкою в рядку:

```
+ FloatToStr((-b — Sqrt(d)) / 2 * a)));
```

Проаналізувавши оператор (а він займає два рядки), виявляємо зайву закриваючу дужку. Знищимо дужку перед символом. Помічаємо також, що в наступному операторі спостерігається така сама помилка. Тому виправимо і її. Відзначимо, що компілятор під час першого сеансу своєї роботи не виявив ці помилки. Крім того, під час другого сеансу була виявлена лише одна помилка, хоч їх було дві.

Етап 4

Натиснемо інструментальну кнопку ▶, запустивши програму на виконання, і переконаємося, що компіляція відбудеться успішно і програма почне виконуватися.

Якщо ввести значення a = 1, b = 2, c = 1, то ми переконаємося, що отримані корені є правильними. Однак для a = 1, b = 5, c = 2 будуть отримані неправильні корені. Тому слід вже шукати логічні помилки.

Етап 5

Відкриємо вікно Watches List (Ctrl + Alt + W), додамо в нього змінну d, помістимо курсор у рядок, наступний за обчисленням значення дискримінанта d, і натиснемо клавішу F4. Після введення вказаних вище значень коефіцієнтів переконуємося, що дискримінант обчислений неправильно (у вікні Watches List виведено значення d, що дорівнює 9, замість 17). Помічаємо, що в тексті програми оператор

d: = b * b – 4 * a * a;

неправильний. Виправляємо помилку:

d: = b * b – 4 * a * c;

Якщо виконати ті самі дії, що й раніше, то ми переконаємося, що тепер у вікні для змінної d буде виведено 0. Отже, десь вище є логічна помилка, яку потрібно виправити.

Етап 6

Не перериваючи виконання програми, додамо у вікно Watches List змінні **a, b, c** (точніше, Forml.a, Forml.b, Forml.c). Бачимо, що змінна **c** має значення 0, хоч уведення її здійснювалося. Водночас змінна **a** має значення 2, хоч у неї уводилося значення 1. Для відшукання помилки припиняємо виконання програми та натискаємо клавішу F8 для нового запуску програми в режимі налаштування. Після кількох натискань клавіші F8 (при цьому будуть по черзі забарвлюватися виконувані рядки) і введення вказаних трьох значень ми помічаємо, що третій оператор уведення замість того, щоб записати значення 1 у змінну **c**, записав його у змінну **a** (це можна було побачити дещо раніше, але людина зазвичай бачить ті значення, що явно виділяються, — тут значення 0 у змінній **c**).

Указану помилку можна було побачити ще після першої компіляції. Річ у тім, що тоді ми не звернули увагу на натяк, у якому вказувалося, що змінна **c** оголошена, але ніколи не використовується.

Отже, ми переконалися, що навіть натяк (Hint) може свідчити про помилку і потрібно звертати увагу не лише на повідомлення про помилки, а й на попередження й натяки.

Тож слід виправити оператор

a:= StrToFloat(Edit3.Text);

записавши його так:

c:= StrToFloat(Edit3.Text);

Якщо тепер запустити програму на виконання, то буде одержаний правильний результат, який однак ще не свідчить про відсутність логічних помилок, оскільки ми ще не перевіряли випадок відсутності коренів.

Етап 7

Запустимо програму на виконання і введемо тепер такі значення: a = 1, b = 2, c = 3. У цьому разі ми бачимо, що, з одного боку, програма виводить повідомлення про відсутність коренів, а з другого — вона видає аварійне повідомлення. Натиснувши кілька разів клавішу F8, бачимо, що оператори виконуються в потрібній послідовності, але після виведення повідомлення про відсутність коренів здійснюється перехід до рядка, у якому обчислюється і виводиться перший корінь, хоча цей рядок не має виконуватися. Річ у тім, що в програмі використаний неправильний для цієї задачі формат оператора if.

Знищимо перед рядком, де виводиться перший корінь рівняння, символ «крапка з комою» і впишемо такий рядок:

else

Якщо тепер запустити програму на виконання, то ми побачимо, що помилка залишилася, але лише щодо другого кореня. За допомогою клавіші F8 переконуємося в тому, що в програмі виконується оператор, який обчислює другий корінь рівняння, тоді як його виконання не мало бути. Річ у тім, що в цьому випадку два оператори

Memo1.Lines.Add(‘ xl = ‘
+ FloatToStr((-b + Sqrt(d)) / 2 * a));
Memol.Lines.Add(‘ x2 = ‘
+ FloatToStr((-b — Sqrt(d)) / 2 * a));

мають розглядатися як один (так званий складений оператор), для чого їх потрібно ввести в операторні дужки **begin — end.**

Зробимо це й отримаємо такий текст процедури TForml.ButtonlClick:

```
procedure TForml.ButtonlClick(Sender: TObject);
begin
a:= StrToFloat(Editl.Text);
b:= StrToFloat(Edit2.Text);
c:= StrToFloat(Edit3.Text);
d:=b*b-4*a*c;
if d < 0 then
Memo1.Lines.Add('The equation has no roots')
else begin
Memo1.Lines.Add(‘ x1 =‘
+ FloatToStr((-b + Sqrt(d)) / 2 * a));
Memo1.Lines.Add (‘ x2 = ‘
+ FloatToStr((-b — Sqrt(d)) / 2 * a));
end;
end;
```

Запуск програми зі значеннями a = 1, b = 2, c = 3 тепер приведе до одержання правильного результату.

Етап 8

Запустимо програму на виконання і введемо нові значення коефіцієнтів: a = 2, b = 5, c = 3. У цьому разі ми бачимо, що обчислені корені (x1 = –4, x2 = –6) знову є неправильними (мають бути отримані значення x1 = –1, x2 = –1.5). Потрібно знову здійснювати покрокове виконання програми, натискаючи клавішу F8. Оскільки ми не передбачили змінні для запису коренів рівняння, доведеться дивитися на результат, що виводиться на екран користувача. У результаті ми побачимо, що всі обчислення до виведення значення першого кореня виконуються правильно, а сам корінь є неправильним. Висновок: є помилка в обчисленні кореня. Уважно подивившись на оператор, ми бачимо, що в ньому порушений порядок виконання операцій, оскільки операції множення й ділення мають один і той самий пріоритет, то під час обчислення кореня спочатку здійснюється ділення на 2, після чого одержаний результат помножується на **a**. Щоб одержати правильний результат, потрібно забрати знаменник **2 * a** в дужки, причому це, звичайно, потрібно зробити і під час обчислення другого кореня.

Після виправлення тексту одержуємо правильний результат. Отже, одноразове одержання правильного результату не дає гарантії того, що програма буде правильною. Остання помилка не проявлялась раніше, тому що для коефіцієнта **a** вводилося значення 1, яке не впливало на результат і в разі попадання цього значення в чисельник, і у разі попадання його у знаменник. Це ще раз доводить необхідність у ретельному доборі тестових даних та багаторазового тестування.

Завдання для практичної роботи

Під час практичної роботи потрібно:

— повторити описані в наведеному вище прикладі дії, пов'язані зі створенням та налаштуванням нового додатка;

— виконати наведені нижче завдання, пов'язані з розробкою форми *(рис. 6)* та набиранням тексту модуля для нового додатка, збереженням його, збереженням із новим іменем, завантаженням проекту з диска, редагуванням та налаштуванням програми.

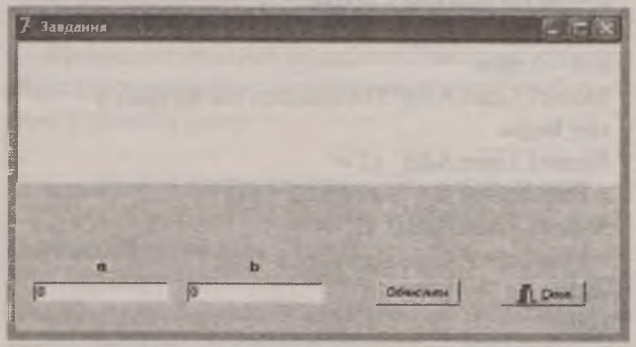


Рис. 6. Вигляд форми для відтворення

Завдання 1

Створіть додаток, у якому повторіть наведений нижче текст опрацьовувача події OnClick кнопки Button1. З'ясуйте, що виконує програма.

```
procedure TForm1.Button1Click(Sender: TObject);
var
a, b, c: Integer;
begin
a:= StrToInt (Edit1. Text);
b:= StrToInt(Edit2.Text);
if a > b then
Memo1.Lines.Add('1 — Yes')
else
Memo1.Lines.Add('1 — No');
c:= a * b;
if c > 100 then
Memo1.Lines.Add('2 — Yes')
else
Memo1.Lines.Add('2 — No');
if a mod 2 = b mod 2 then
Memo1.Lines.Add('3 — Yes')
else
Memo1.Lines.Add('3 — No');
end;
```

Завдання 2

Нижче наведено код опрацьовувачем події OnClick кнопки Button1 із синтаксичними помилками. На основі цього коду створіть Windows-додаток, у якому усуньте всі знайдені синтаксичні помилки.

```
procedure TForm1.Button1Click(Sender: TObject);
var
a, b, c: Integer;
begin
a:= StrToInt(Edit1.Text);
b:= StrToInt(Edit2.Text);
if (a >< b) and (b >< 0) then begin
if a * b > 0 then
Memo1.Lines.Add('1 — Yes');
else
Memo1.Lines.Add('1 — No');
else
Memo1.Lines.Add('?');
end;
```

Завдання 3

Нижче наведено варіант коду, що нормує вектор і здійснює виведення результату. У коді містяться помилки. На основі наведеного коду створіть застосунок, що працюватиме коректно.

```
procedure TForm1.Button1Click(Sender: TObject);
var
a, b, c, d: Real;
begin
a:= StrToFloat(Edit1.Text);
a:= StrToFloat(Edit2.Text);
```

```
c = Sqrt(a * a + b * b);
a = a / d;
b = b / d;
Memo1.Lines.Add(FloatToStr(a) + ' ' + FloatToStr(b));
end;
```

Вимоги до звіту

У звіті з практичної роботи (надрукованому на принтері) мають бути відображені наступні складові.

1. Тема роботи.
2. Мета роботи.
3. Короткі теоретичні відомості з теми.
4. Текст і результати роботи програм.
5. Висновки щодо роботи.

Завдання для практичної роботи

Контрольні запитання

1. Які дії потрібно виконати для створення нового додатка?
2. Яке призначення основних вікон інтегрованого середовища Delphi?
3. Яке призначення вікна **Дизайнера форми?**
4. Яке призначення вікна коду?
5. Яке призначення вікна **Інспектора об'єктів?**
6. Яке призначення вікна **Дерева об'єктів?**
7. Як поділяються помилки у програмі? Опишіть їх особливості.
8. Опишіть структуру модуля форми.
9. Що відстежується під час покрокового виконання програми?
10. У чому полягає відмінність режимів покрокового виконання Trace Into та Step Over?
11. Що таке «виконання до курсора»?
12. Що таке точка переривання і як вона встановлюється і видаляється?
13. У чому відмінність точки переривання і виконання до курсора?
14. Чи можна одночасно встановити кілька точок переривання?
15. Для чого використовується вікно Watches List?

Використані джерела

1. *Архангельский А. Я.* Программирование в Delphi 6 / А. Я. Архангельский. — М. : БИНОМ, 2002. — 1120 с.
2. *Безменов М. І.* Основи програмування в середовищі Delphi: навч. посіб. / М. І. Безменов. — Ч. : НТУ «ХПІ», 2010. — 608 с.
3. *Глинський Я. М.* Паскаль. Turbo Pascal і Delphi: навч. посіб. / Я. М. Глинський — Львів : СПД Глинський, 2008. — 192 с.
4. *Культин Н. Б.* Основы программирования в Delphi 2007 / Н. Б. Культин. — СПб. : БХБ-Петербург, 2008. — 480 с.
5. *Яременко Г. І.* Програмування в середовищі Delphi: навч. посіб. / Г. І. Яременко. — Черкаси : ЧДТУ, 2011. — 188 с.