МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ”

ІНЖЕНЕРНО-ХІМІЧНИЙ ФАКУЛЬТЕТ
КАФЕДРА «АВТОМАТИЗАЦІЯ ХІМІЧНИХ ВИРОБНИЦТВ»

МЕТОДИЧНІ ВКАЗІВКИ ДО ЛАБОРАТОРНИХ РОБІТ З КРЕДИТНОГО МОДУЛЯ «ПРОГРАМУВАННЯ МОВОЮ С» КУРСУ “ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ –1”

для студентів напрямку підготовки “6.050202 – Автоматизація та комп’ютерно-інтегровані технології ”

Київ
НТУУ «КПІ»
2013

Методичні вказівки до лабораторних робіт з кредитного модуля «Програмування мовою С» курсу “Прикладне програмне забезпечення – 1” для студентів напрямку підготовки “6.050202 – Автоматизація та комп’ютерно-інтегровані технології ”, [Текст] / Уклад. В. М. Ковалевський, // – К.: НТУУ «КПІ», 2013. – 137 с.

Гриф надано Методичною і Вченою радою ІХФ «КПІ»

(Протокол №_1__ від _27_ січня 2014 р.)

Навчальне видання

МЕТОДИЧНІ ВКАЗІВКИ ДО ЛАБОРАТОРНИХ РОБІТ З КРЕДИТНОГО МОДУЛЯ «ПРОГРАМУВАННЯ МОВОЮ С» КУРСУ “ПРИКЛАДНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ –1”

для студентів напрямку підготовки “6.050202 – Автоматизація та комп’ютерно-інтегровані технології ”

Укладач: *Ковалевський Валерій Михайлович,* канд. техн. наук, доц.

Відповідальний
за випуск: *А. І. Жученко,* док. техн. наук, професор.

Рецензент: *В. І. Сівецкий,* канд. техн. наук, професор.

Авторська редакція

Зміст

стор.

В С Т У П

Лабораторні роботи з кредитного модуля «Програмування мовою С» курсу “Прикладне програмне забезпечення – 1” передбачають створення прикладних програм в ***C***++ ***Builder*** і орієнтовані на вивчення техніки та методик по розробкам і побудуванню прикладних програм на мовах програмування ***C*** та ***C***++. На протязі навчального семестру з кредитного модуля «Програмування мовою С» курсу «Прикладне програмне забезпечення – 1» лабораторні роботи виконуються відповідно до графіку з лабораторних занять, який наведено у таблиці В-1.

Графік проведення лабораторних занять, кількість навчальних годин, теми і зміст занять.

Таблиця В-1.

№ п/п заняття	Кількість годин заняття	Зміст і тема заняття
1	2	Інструктаж бакалаврів по таких питаннях: - техніки безпеки при роботи по виконанню лабораторних робіт; - особливості структури інтегрованого програмувального середовища C++ Builder; - методика і правила використання C++ Builder для розробки листингів та виконавчого коду прикладних програм; - вимоги по підготовки бакалаврів до занять та по оформленню результатів до захисту виконаних завдань по лабораторним роботам.
2	2	Виконання вказівок і команд по протоколу лабораторної роботи № 1 на тему «Структура вікон і компонент та техніка їх використання в C++ Builder для побудування прикладної програми».
3	2	Виконання вказівок і команд по протоколу лабораторної роботи № 2 на тему «Вказівки препроцесору до умовної та багато-файлової компіляції файлів прикладної C++ програми при створенні консольного виконуючого коду для MS DOS».
4	1	Захист створеної прикладної програми у лабораторній роботі № 1 та результатів досліджень алгоритму по роботі програми і алгоритмів функцій з обробки подій до компонент ***Image*1** та ***Button*1**.

4	1	Захист створеної прикладної програми у лабораторній роботі № 2 та результатів досліджень алгоритму по роботі програми при виконанні умовної та багато-файлової компіляції файлів.
5	2	Виконання вказівок і команд по протоколу лабораторної роботи № 3 на тему «Команди редактора C++ Builder і структура створюваних файлів до проекту прикладної програми».
6	2	Виконання вказівок і команд по протоколу лабораторної роботи № 4 на тему «Правила налагодження компоненти ***MainMenu*** для створення меню команд прикладної C++ програми».
7	1	Захист створеної прикладної програми у лабораторній роботі № 3 та результатів досліджень алгоритму по роботі програми і команд редактора C++ Builder і структура створюваних файлів до проекту прикладної програми.
7	1	Захист створеної прикладної програми у лабораторній роботі № 4 та результатів досліджень алгоритму по роботі програми і налагодження компоненти ***MainMenu*** для створення меню команд прикладної C++ програми.
8	2	Виконання вказівок і команд по протоколу лабораторної роботи № 5 на тему « Техніка використання та контролю з обробки даних структурного типу в C++ програмі».
9	2	Виконання вказівок і команд по протоколу лабораторної роботи № 6 на тему «Техніка використання функцій при обробках команд мишки до подій в прикладних C++ програмах».
10	1	Захист створеної прикладної програми у лабораторній роботі № 5 та результатів досліджень алгоритму по роботі програми і алгоритмів функцій та команд ***Debug***.
10	1	Захист створеної прикладної програми у лабораторній роботі № 6 та результатів досліджень алгоритму по роботі програми і алгоритмів функцій для обробки команд до подій мишки в прикладних програмах.
11	2	Виконання вказівок і команд по протоколу лабораторної роботи № 7 на тему «Компоненти C++Builder та інструменти і функції для побудови графічних зображень на формі вікна прикладної програми».
12	2	Виконання вказівок і команд по протоколу лабораторної роботи № 8 на тему «Техніка створювання графічної анімації в C++ програмі на основі використання компоненти «***Animate***».
		Захист створеної прикладної програми у лабораторній роботі № 7 та результатів досліджень алгоритму по

13	1	роботі програми і алгоритмів функцій для побудови графічних зображень на формі вікна прикладної програми.
13	1	Захист створеної прикладної програми у лабораторній роботі № 8 та результатів досліджень алгоритму по програмуванню графічної анімації в прикладних С++ програмах.
14	2	Виконання вказівок і команд по протоколу лабораторної роботи № 9 на тему «Дослідження алгоритму з динамічної графіки на формі вікна прикладної С++ програми».
15	2	Виконання вказівок і команд по протоколу лабораторної роботи № 10 на тему «Алгоритм роботи з фрагментами зображення розташованого у компоненті «***Image***».
16	1	Захист створеної прикладної програми у лабораторній роботі № 9 та результатів досліджень алгоритму з виконання динамічної графіки на формі вікна прикладної С++ програми.
16	1	Захист створеної прикладної програми у лабораторній роботі № 10 та результатів досліджень алгоритму по роботі з фрагментами графічного зображення на формі вікна прикладної С++ програми.
17	2	Захист звіту та результатів з розробки С++ програми по програмуванню задачі до модульної контрольної роботи на тему «**Обробка меню команд прикладної навчальної С++ програми з технічного засобу автоматизації (***назва приладу з системи контролю та регулювання)* ».
18	2	Захист звіту та результатів після виправлення помилок з розробки С++ програми по програмуванню задачі до модульної контрольної роботи на тему «**Обробка меню команд прикладної навчальної С++ програми з технічного засобу автоматизації (***назва приладу з системи контролю та регулювання)* ».

Лабораторна робота № 1.

Структура вікон і компонент та техніка їх використання в C++ Bulder для побудування прикладної програми

Ціль виконання лабораторної роботи. Інтегроване програмувальне середовище ***C++ Builder*** має відповідну структуру вікон та меню команд і тому задача даної роботи складається у вивченні:

- структури і призначення елементів інтегрованого середовища ***C++ Builder*** для розробки ***C*** і ***C++*** програм за допомогою візуальних компонент бібліотеки ***VCL***;
- правил і техніки встановлення та налагодження властивостей компонент на формі прикладної програми;
- методики визначення подій до компонент на формі програми за допомогою спеціалізованого інспектора об'єктів (***Object Inspector***).

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Структура вікон *C++ Builder* та їх призначення для розробки програми

При запуску в роботу ***C++ Builder*** відкриваються на екрані дисплея чотири взаємно зв'язані вікна, зображення яких показано на рис. 1-1.

Перше - основне керуюче вікно, що має меню команд і бібліотеки візуальних компонент (***VCL***). *Друге* - форма (платформа) для розміщення візуальних і не візуальних компонентів, яки будуть використовуватися ***C++*** програмою. *Третє* - вікно має назву «редактор кодів вихідних текстів програми» і до якого примикає вікно структури класів та функцій, застосовуваних у програмі. *Четверте* – вікно інспектора об'єктів для налагодження властивостей (***Properties***) компонент і визначення подій (***Events***), що будуть відбуватися при роботі ***C++*** програми; *П'яте* – вікно дерево форм з їх компонентами і яке дозволяє бачити загальну структуру компонент, з яких складається кожна форма прикладної програми.

Закриття основного керуючого вікна викликає закриття і інших підлеглих та залежних вікон інтегрованого середовища C++ Builder. Текст програми записується і показується у вікні "Редактор код" (3), де спочатку

розташовано шаблон начала проекту програми і для форми вікна визначені початкові властивості: ***Width***, ***Height*** – ширина і висота; ***Top***, ***Left*** – положення форми на екрані; ***Caption*** – властивість з назвою до заголовка форми програми. Команди в C++ Builder можна виконувати через основне меню команд або швидкими кнопками інструментів, список і опис яких наведено у таблиці 1-1.

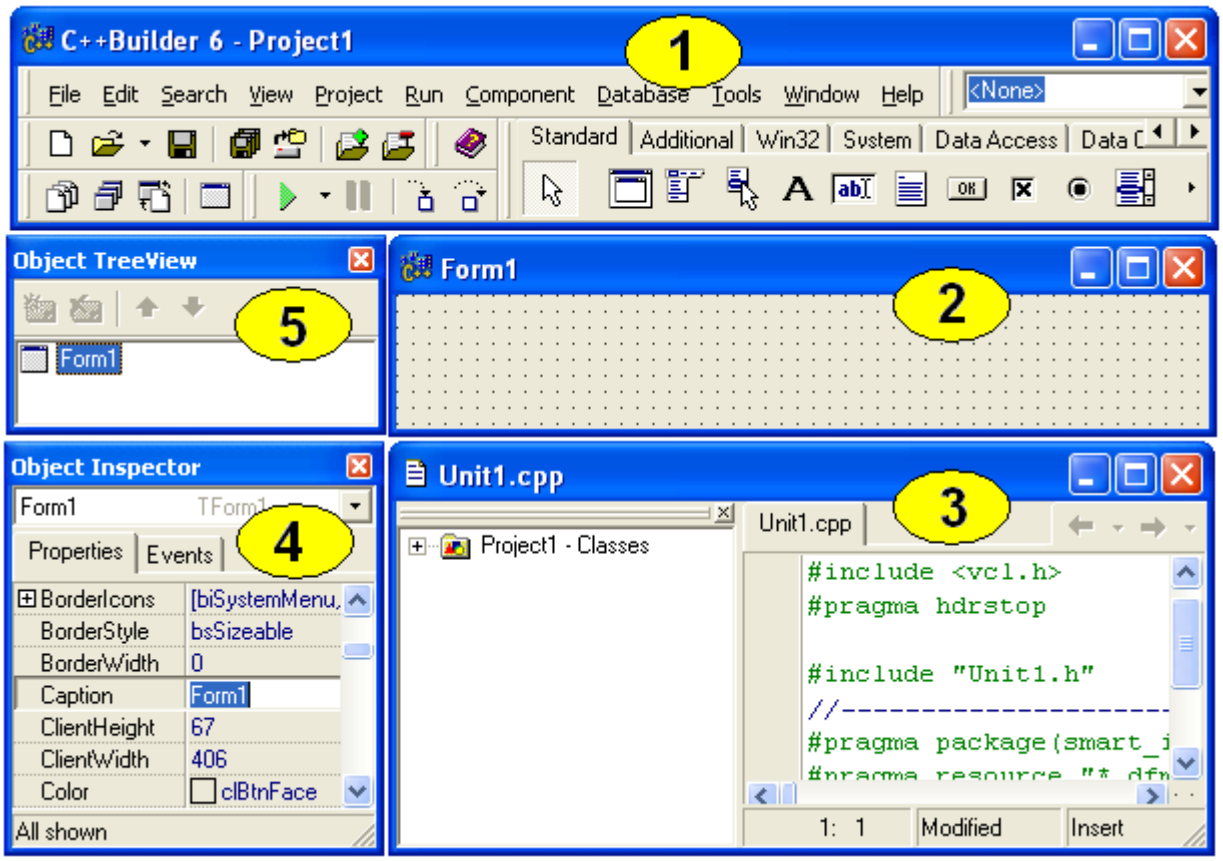


Рис. 1-1. Вікна інтегрованого програмувального середовища C++ Builder:
1 - основне керуюче вікно; 2 - вікно форми для розміщення компонентів з бібліотеки VCL; 3 - вікно редактора коду вихідних текстів С або С++ до прикладних програм; 4 - вікно “Інспектор об'єкту ”; 5 - вікно дерева компонент, встановлених на форму програми.

Список швидких кнопок ***C++ Builder*** та їх призначення. Таблиця № 1-1.

КНОПКА	**Команда меню і "гарячі" клавіші**	**ВИКОНАННЯ КОМАНДИ**
	File / Open	Відкрити проект або модуль з депозитарію.
	File / Open File / Reopen	Відкрити файл проекту, модуля, пакета.
	File / Save (Ctrl - S)	Зберегти файл модуля, з яким у даний момент йде робота.

	File / Save All	Зберегти все (усі файли модулів і проекту).
	File / Open Project (Ctrl-F1)	Відкрити файл проекту.
	Project / Add to Project (Shift-F1)	Додати файл у проект.
	Project / Remove from project	Видалити файл із проекту.
	Help / C++ Builder Help	Виклик сторінки змісту вбудованої довідки.
	View / Units (Ctrl-F12)	Переключення на перегляд тексту файлу, обраного зі списку.
	View / Forms (Shift-f12)	Переключення на перегляд форми файлу, обраного зі списку.
	View / Toggle Form/Unit (F12)	Переключення між формою і відповідним файлом модуля.
	File / New Form	Включити в проект нову форму.
	Run / Run (F9)	Виконати додаток. Кнопочка зі стрілкою праворуч від основного зображення дозволяє вибрати виконуваний файл, якщо ви працюєте з групою програм.
	Run / Program Pause	Пауза виконання програми і перегляд інформації CPU. Кнопка і відповідний розділ меню доступні тільки під час виконання програми (C++ програми).
	Run / Trace Into (F7)	Покрокове виконання програми з заходом у функцію.
	Run / Step Over	Покрокове виконання програми без заходу у функцію.
		Панель вибору зі списку конфігурації вікна.
	View / Desktops / Save Desktop	Збереження поточної конфігурації вікна.
	View / Desktops / Set Debug Desktop	Установка конфігурації вікна при налагодженні.

Компоненти вибираються в ***C++ Builder*** мишкою з палітри (рис. 1-2) ***Visual Component Library*** (***VCL***) і встановлюються на формі у відповідне місце. Склад та призначення компонент в ***VCL*** наводиться у табл. 1-2.

Рис. 1-2. Палітра компонент бібліотеки *VCL* .

Групи компонент бібліотеки ***VCL*** та їх призначення**.** Таблиця № 1-2.

WebServices	Стандартна (група компонентів, яки часто використовуються у C++ програмах).
Midos	Додаткова, що є доповненням стандартної групи.
InternetExoress	32-бітні компоненти у стилі Windows 95/98 і NT.
Internet	Системні (серед них є компоненти, як таймери, плеєри).
WebSnap	Доступ до даних через Borland Database Engine (BDE).
FastNet	Керування даними.
Decision Cube	Зв'язок з даними за допомогою dbExpress.
Qreport	Компоненти для зв'язку із сервером додатків при побудові багато-поточних програм, що працюють з даними.
Dialogs	Доступ до даних через Borland Database Engine.
Win 3.1	Зв'язок з базами даних через Activ Data Objects(ADO) - безліч компонентів Activ, що використовують для доступу до баз даних.
Samples	Прямий зв'язок з InterBase, минаючи Borland Database Engine(BDE) і Active Data Object(ADO).
ActiveX	Компоненти клієнтських додатків Web, що використовують доступ до служб Web за допомогою SOAP.
COM+	Побудова додатків баз даних з рівнобіжними потоками.
InterBaseAdmin	Побудова додатків InternetExpress - одночасно додатків сервера Web і клієнта баз даних з рівнобіжними потоками.
Servers	Компоненти для додатків, що працюють з Інтернет.
IndyClients	Компоненти для створення серверів Web, що містять складні сторінки, керованих даних.
IndyServers	Компоненти серверних додатків Internet Direct.(Indy).
IndyMisc	Різні допоміжні компоненти Internet Direct.(Indy).
Office2k	Оболонки VCL, для офісних додатків Microsoft.

Постановка задачі по програмуванню у лабораторній роботі

Необхідно створити ***C++*** програму для роботи у ***Windows*** і у який буде вікно з рисунком і також буде кнопка з назвою "СТАРТ". При роботі програми повинні відбуватися та оброблятись функціями наступні події:

- ✓ ***перша подія*** – поява у вікні текстового повідомлення " Перша програма створена для ***Windows*** у інтегрованому середовищі ***C++ Builder***", якщо виконується натискання мишкою кнопки "СТАРТ";

- ✓ ***друга подія*** – видалення рисунка з поля вікна програми, якщо виконується переміщення мишки по напису текстового повідомлення;
- ✓ ***третя подія*** – відновлення рисунка у вікні програми, якщо мишка буде встановлена на полі або переміщуватись по полю вікна.

Завдання для аналізу і обробки результатів роботи запрограмованої C++ програми

Після запуску в роботу ***C***++ програми лабораторної роботи № 1 для захисту результатів програмування необхідно виконати таке:

- нарисувати блок-схему алгоритму з роботи програми лабораторної роботи № 1;
- нарисувати блок-схему алгоритму по виконанню операторів функції, з обробки в програмі першої події;
- нарисувати блок-схему алгоритму по виконанню операторів функції, з обробки в програмі другої події;
- нарисувати блок-схему алгоритму по виконанню операторів функції, з обробки в програмі третьої події.

Порядок дій і команд для виконання програмування задачі:

Крок 1. Активізуємо файли проекту для нової ***C***++ програми і зберігаємо їх на диску в задану папку:

- ➢ Виконайте команду ***File/New Application***, щоб з'явилася нова чиста форма ***Form*1**;
- ➢ Перейдіть у вікно інспектора об'єктів і у властивості ***Caption*** задайте назву лабораторної роботи "Лабораторна робота № 1", щоб на заголовку вікна форми замість назви ***Form1*** з'явився напис "Лабораторна робота № 1";
- ➢ Збережіть новий проект із порожньою формою та змінами в заголовку вікна і це активізує новий проект файлів в ***C***++ ***Builder*** , яке запам'ятає шлях для швидкого збереження змін у проекті програми;

- Збереження файлів проекту виконайте на ***D:\LA_NN\LAB_1***, де ***NN*** - номер навчальної групи;
- У меню ***File*** виберіть команду ***Save Project As*** і з появою запиту на збереження змінить назву файлу ***Unit1*** на файл ***U_Lab1.cpp***, а назву проекту ***Project1.bpr*** замініть на ***P_Lab1.bpr***.

Крок 2. Установлюємо на форму вікна компоненту ***Panel1***:

- На палітрі компонентів ***VCL*** і закладці ***Standart*** одним щигликом миші та курсором виділяємо компоненту ***Panel***;
- На формі вікна з заголовком "Лабораторна робота № 1" установіть одним щигликом миші компоненту ***Panel1*** (рис. 1-4).

Крок 3. Задаємо маркерною рамкою по горизонталі і вертикалі необхідний розмір компоненти ***Panel1*** за допомогою текстового повідомлення (рис. 1-4):

- Маркерну рамку розтягніть методом перетаскування чорного квадратика під розмір майбутнього напису виведеного тексту.

Крок 4. Налаштуємо у вікні інспектора об'єктів властивості до тексту, що буде виводитися на компоненті ***Panel1***(шрифт, розмір і колір напису):

- На закладці ***Properties*** подвійним щигликом мишки виберіть властивість ***Font*/(*TFont*)** для відкриття вікна "Шрифт";
- Для напису вибираємо: необхідний шрифт; розмір і колір.

Крок 5. На формі вікна в полі для текстового повідомлення видаляємо напис ***Panel1***, щоб він не накладався на виведений текст повідомлення:

- На компоненту ***Panel1*** встановить мишкою маркерну рамку;
- Перейдіть у вікно інспектора об'єктів (***Object Inspector***) і на сторінці властивостей об'єкта (***Properties***) у виділеному полі ***Caption*** встановить курсор на напис ***Panel1*** і цей напис видалите з клавіатури клавішею ***Delete*** або ***Backspace.***

Крок 6. Установимо на форму вікна кнопку ***Button1***:

- На закладці ***Standart*** палітри компонентів ***VCL*** виберіть одним щигликом мишки компоненту "кнопка" з ярличком підказки ***Button***;

➢ На полі форми із сіткою зробіть щиглика мишкою, щоб встановилася кнопка з назвою ***Button1*** з охопленою маркерною рамкою.

Крок 7. На кнопці ***Button1*** змінюємо назву на "СТАРТ":

➢ Клацніть по кнопці ***Button1*** на формі вікна для появи маркерної рамки;

➢ Перейдіть у вікно інспектора об'єктів (***Object Inspector***) і на сторінці властивостей об'єкта (***Properties***) встановить курсор (рис. 1-3) у виділене поле ***Caption*** на напис ***Button1*** і змінить на назву "СТАРТ".

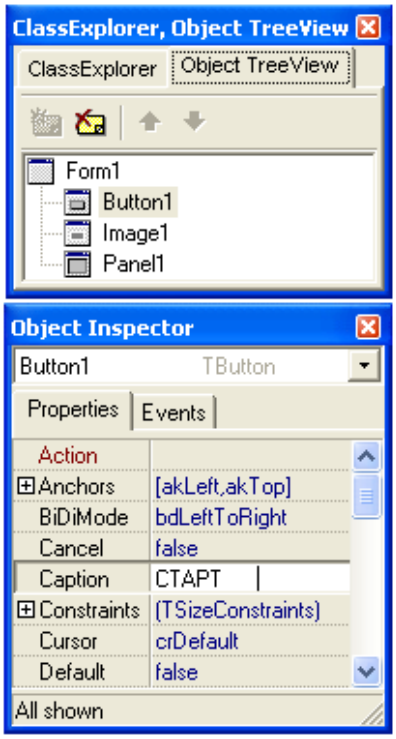


Рис. 1-3.

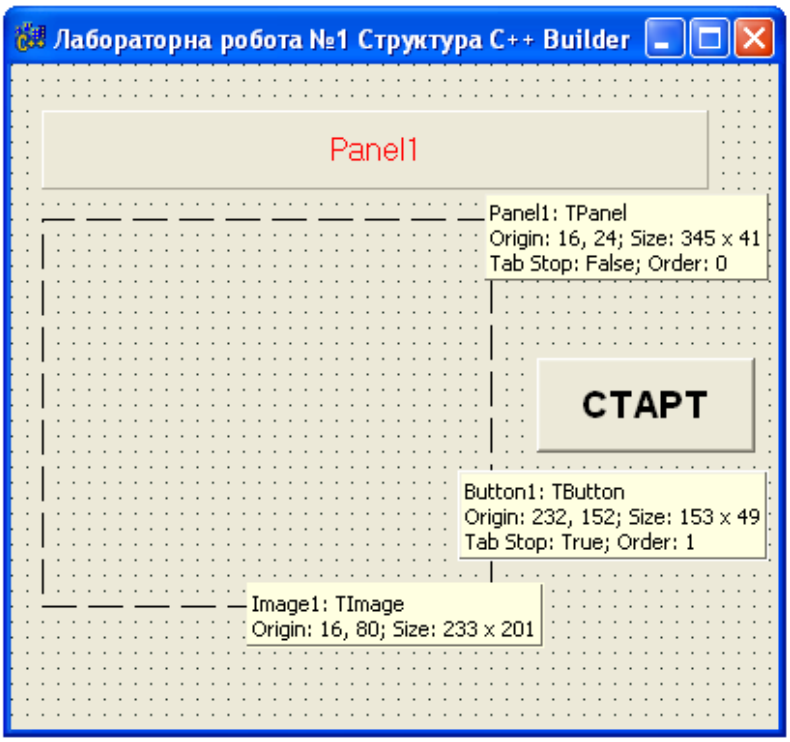


Рис. 1-4. Розміщення на формі компонент та їхні розміри.

Крок 8. Визначаємо для кнопки "СТАРТ" подію - вивід тексту «Перша програма створена для ***Windows*** у інтегрованому середовищі ***C++ Builder***». Для цього є два варіанти команд.

Варіант 1.

➢ Активізуйте мишкою маркерну рамку на кнопці "СТАРТ";

➢ Перейдіть у вікно ***Object Inspector*** на закладку подій (***Events***) і виконайте подвійного щиглика мишкою на білому полі біля напису ***OnClick***, щоб відбулося наступне:

- на білому полі з'являється напис події ***Button1Click***;

- активізується вікно редактора кодів, де буде показуватися доданий шаблон функції до обробки події **:: *Button*1*Click***.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Варіант 2.

- На формі по кнопці "СТАРТ" зробіть подвійного щиглика мишкою, щоб автоматично активізувалися:
  - інспектор об'єктів(***Object Inspector***) , де вже буде подія ***Button*1*Click***;
  - одночасно вікно редактора кодів відкриється з доданим шаблоном функції до обробки події **:: *Button*1*Click***.

Крок 9. У вікні редактора кодів для файлу ***U_Lab*1*.cpp*** записуємо команди операторів між фігурними дужками функції події **:: *Button*1*Click***:

- У редакторі кодів допишіть оператор для події – показу тексту повідомлення " Перша програма створена для Windows у інтегрованому середовищі ***C***++ ***Builder*** ". Цей оператор записується у визначення функції ***TForm*1::*Button*1*Click***, що повинна мати такий вигляд:

```
//---------------------------------------------------------------------------
    void __fastcall TForm1::Button1Click(TObject *Sender)
    {
     Panel1->Caption = "Першу програму створено для Windows у інтегрованому
    середовищі C++ Builder ";
    }
//---------------------------------------------------------------------------
```

Крок 10. Командами ***Run/Run*** чи на клавіатурі клавішею ***F*9** виконуємо компіляцію вихідних файлів і запускаємо на виконання програму, щоб перевірити правильність виконання операторів з обробки першої події:

- Якщо не було помилок при запису оператора ***Panel1->Caption =*** , то на екрані знімається вікно інспектора об'єктів і з'являється вікно створюваної ***C***++ програми з кнопкою "СТАРТ";

- Мишкою клацніть кнопку "СТАРТ", щоб у вікні на компоненті ***Panel*1** з'явився напис " Перша програма створена для ***Windows*** у інтегрованому середовищі ***C*++ *Builder*** ".

Крок 11. Налаштовуємо показ жирним шрифтом і червоним кольором напису " Перша програма створена для ***Windows*** у інтегрованому середовищі ***C*++ *Builder*** ":

- У вікні інспектора об'єктів вгорі у списку компонент з форми виберіть назву ***Panel*1**, щоб для компоненти ***Panel1***на формі вікна активізувалася маркерна рамка;
- Розкрийте список властивостей ***Font*** щигликом по значку + та потім подвійним щигликом мишки розкрийте ***TFont***, щоб відкрилося вікно "Шрифт", де задайте для символів шрифт, розмір і колір.

Крок 12. Перевіряємо правильність внесених змін у текст програми:

- Виконаєте компіляцію вихідних файлів і запустите на виконання програму командами ***Run/Run*** чи на клавіатурі натисніть клавішу ***F*9.**

Крок 13. Змінюємо вид і розмір шрифту для кнопки "СТАРТ":

- Активізуйте маркерну рамку на кнопці "СТАРТ" одиночним щигликом мишки;
- У вікні ***Object Inspector*** автоматично активізується властивість ***Font*** з режимом ***Tfont*** і праворуч на білому полі буде видна кнопка з трьома горизонтально розташованими крапками, яку клацніть мишкою, щоб відкрилося стандартне вікно "Шрифт" для зміни виду шрифту і розміру букв.

Крок 14. Командами ***Run/Run*** чи на клавіатурі клавішею ***F*9** виконуємо компіляцію вихідних файлів і запускаємо на виконання для перевірки роботи ***C*++** програму.

Крок 15. На закладці ***Additional*** вибираємо компоненту ***Image*** і встановлюємо її в нижній частині форми вікна (рис. 1-4).

Крок 16. Для форми ***TForm1*** задаємо функцію для відкриття вікна програми:

- Виконаєте щиглика мишкою на сітці форми і перейдіть у вікно інспектора об'єктів на закладку подій ***Events***;
- Виберіть ***OnCreate*** і зробіть подвійного щиглика мишкою, щоб у вікні редактора кодів був вставлений наступний шаблон до функції:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{

}
//---------------------------------------------------------------------------
```

Крок 17. У вікні редактора кодів для файлу ***U_Lab*1.*cpp*** додаємо оператори в шаблон функції для відкриття вікна при запуску ***C***++ програми в роботу:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
TImage *Pict = new TImage(Form1);
Pict->AutoSize = true;
//---------- завантаження вихідного рисунка в поле Image при відкритті вікна
//------рисунок Lab1_On.bmp помістити в папку LAB_1
Pict->Picture->LoadFromFile("Lab1_On.bmp");
Image1->Canvas->CopyRect(Image1->ClientRect, Pict->Canvas,
                  Rect(0,0,Pict->Width,Pict->Height));
On_Off = 1;     //-----рисунок показано
}
//---------------------------------------------------------------------------
```

Крок 18. Командами ***Run/Run*** або на клавіатурі клавішею ***F*9** виконуємо компіляцію вихідних файлів і запускаємо на виконання ***C***++ програму для перевірки роботи доданих операторів у функцію.

Крок 19. Для компоненти ***Panel*1** задаємо обробку події – видалення малюнка з полю форми вікна, якщо покажчик мишки буде розташований на текст напису "Першу програму створено для ***Windows*** у інтегрованому середовищі ***C***++ ***Builder*** ":

- Виділить на формі маркерною рамкою компоненту ***Panel*1**;
- Перейдіть у вікно інспектора об'єктів на закладку подій ***Events***;
- Виберіть ***OnMouseMove*** і зробіть подвійного щиглика мишкою, щоб у вікні редактора кодів уставився шаблон для функції з обробки події:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Panel1MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
}
//---------------------------------------------------------------------------
```

Крок 20. У вікні редактора кодів для файлу ***U_Lab*1.*cpp*** записуємо оператори в шаблон функції для обробки події - видалення малюнка з поля вікна програми при переміщенні покажчика мишки над текстовим повідомленням:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::Panel1MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
TImage *Pict = new TImage(Form1);
Pict->AutoSize = true;
//---------- видалення рисунка в полі Image
//------ рисунок Lab1_Off.bmp помістити в папку LAB_1
Pict->Picture->LoadFromFile("Lab1_Off.bmp");
Image1->Canvas->CopyRect(Image1->ClientRect, Pict->Canvas,
                  Rect(0,0,Pict->Width,Pict->Height));
On_Off = 0;   //---рисунок виключено
}
//---------------------------------------------------------------------------
```

Крок 21. Командами ***Run/Run*** чи на клавіатурі клавішею ***F*9** виконуємо компіляцію вихідних файлів і запускаємо на виконання програму.

Крок 22. *Я*кщо з'явилося повідомлення, що не оголошена змінна ***On_Off***. Додаємо для змінної таке оголошення ***int On_Off* = 0** на зовнішньому рівні програми (дивись крок № 25).

Крок 23. Для форми ***Form1*** задаємо подію – відновлення рисунка на формі, якщо покажчик мишки буде розташовано на полі вікна програми:

- На сітці форми ***Form1*** зробіть щиглика мишкою;
- Переходимо у вікно інспектора об'єктів на закладку подій ***Events*** ;
- Виберіть подію ***OnMouseMove*** подвійним щигликом мишки, щоб у вікні редактора кодів з'явився шаблон такої функції:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift, int X, int
Y)
{
```

```
}
//---------------------------------------------------------------------------
```

Крок 24. У вікні редактора кодів для файлу ***U_Lab*1.*cpp*** заповнюємо оператори в шаблон функції для обробки події – переміщення покажчика мишки по полю вікна програми і відновлення рисунка на полі вікна:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
TImage *Pict = new TImage(Form1);
Pict->AutoSize = true;
    if(On_Off = = 0 )  //---------- відновлення рисунка в полі Image
    {
Pict->Picture->LoadFromFile("Lab1_On.bmp");
Image1->Canvas->CopyRect(Image1->ClientRect, Pict->Canvas,
                  Rect(0,0,Pict->Width,Pict->Height));
 On_Off = 1;  //-----рисунок показано
    }
}
//---------------------------------------------------------------------------
```

Крок 25. Перевіряємо вказівки препроцесору і оголошення у файлі ***U_Lab*1.*cpp***, яки повинні бути такими:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "U_Lab11.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
int On_Off = 0; //--- для контролю показу-зняття рисунка та щоби не мигав рисунок
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
                    : TForm(Owner)
{

}
//---------------------------------------------------------------------------
```

Крок 26. Перевіряємо роботу програми з лабораторної роботи шляхом виконання компіляції вихідних файлів і для цього виконуємо команди ***Run/Run*** або на клавіатурі натискається клавіша ***F*9**.

Контрольні запитання до лабораторної роботи № 1

1. Пояснить загальні правила активізації (ініціалізації) основних файлів проекту, створюваної програми у ***C***++ ***Builder***.
2. Яки команди на блок схемі алгоритму виконуються для обробки події – поява на полі вікна текстового повідомлення.
3. Пояснить алгоритм та початкові тексти ***C***++ програми по обробці події – вилучення рисунка з поля вікна програми.
4. Пояснить алгоритм та початкові тексти ***C***++ програми по обробці події – відновлення рисунка на полі вікна програми.
5. Покажіть команди ***C***++ ***Builder***, якими виконується налаштування компонент для обробки подій мишки.

Лабораторна робота № 2

Вказівки препроцесору до умовної та багато-файлової компіляції файлів прикладної C++ програми при створенні консольного виконуючого коду для MS DOS

Ціль виконання лабораторної роботи. Передбачається виконання таких навчальних задач:

- вивчення правил і методики розробки у ***C***++ ***Builder*** прикладної ***C***++ програми для роботи в операційному середовищі ***MS DOS***;
- отримання навичок практичної роботи з командами для формування виконуючого файлу (.***exe***) для роботи в операційній системі ***MS DOS***;
- практичне вивчення правил і команд з багато-файлової компіляції окремих програмних модулів ***C***++ програми і бібліотечних файлів;

- практична робота з набором файлів для виконання умовної компіляції для прикладної ***C***++ програми методом "вказівок препроцесору";
- освоєння техніки застосування редактора коду ***C***++ ***Builder*** для поділу початкового тексту ***C***++ програми на окремі програмні модулі;
- прикладне програмування алгоритмів для розрахунків числовими методами значень інтегралів до заданих функцій.

Методика одержання коду C++ програм для виконання в MS DOS

В ***Windows C***++ програми працюють с ***MS DOS*** у спеціальному вікні і програма може виконуватися у віконному або в режимі повного екрану. Для роботи в операційному середовищі ***MS DOS*** інтегроване програмувальне середовище ***C***++ ***Builder*** забезпечує одержання виконуючого файлу прикладної програми за допомогою програмного майстра (***Consol Wizard***), що активізується з вікна ***New Items*** через меню ***File*** командою ***New***. Програмний майстер ***Consol Wizard*** формує необхідні файли проекту і у вікно редактора коду розміщує каркас програмного шаблона для заповнення тексту ***C*** програми або ***C***++ програми для ***MS DOS***.

Умовна компіляція програм для ***MS DOS*** і програм для ***Windows*** дає можливість програмісту керувати процесом виконання директив(вказівок) препроцесора при компіляції одержуваного програмного коду. Програмні модулі можуть бути написані на всі можливі варіанти роботи прикладної програми, а застосування умовної компіляції дозволяє компілювати файли вибірково для одержання заданого варіанту роботи виконавчого коду. Умовна директива препроцесора #***if*** багато в чому схожа на запис умовного оператора ***if***( ). Синтаксис умовної директиви препроцесора має такий вигляд

#***if*** умова

фрагмент коду програми

#***endif***

У цьому записі умова є цілочисловим заданим вираженням. Якщо вираження повертає не нуль(істинно), то фрагмент коду, записаний між #***if*** та #***endif***

компілюється. Якщо ж умова помилкова повертається нуль, то текст коду пропускається препроцесором і не компілюється. Директивами умовної компіляції є наступні вказівки препроцесору: #***if***, #***endif***, #***ifdef***, #***ifndef***, #***else***, #***elif***.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Постановка задачі по програмуванню у лабораторній роботі

У лабораторній роботі № 2 необхідно виконати наступні дві задачі:

Задача А. Для обчислення заданих інтегралів створюємо за допомогою ***Consol Wizard*** файл ***C***++ програми, що виконуються, у вигляді консольного додатка ***MS DOS***. Задача А складається з двох частин.

У першій частині (задача А1*)* – компілюємо повний вихідний файл, у якому містяться об'яви та визначення усіх нестандартних функцій. Для цього в папці ***LAB2*** створюємо папку ***LAB2_A***1 і в неї зберігаємо усі файли проекту ***P_Lab2_A1***.

В другій частині (задача А2*)* – текст програми розділяємо за допомогою редактора коду на окремі файли програмних модулів, у які поміщаємо визначення нестандартних функцій. Файл, що виконується, формуємо шляхом багато-файлової компіляції методом "вказівок препроцесору". Усі вихідні файли прикладної програми і файли проекту ***P_Lab2_A2*** зберігаємо в папці ***LAB2_A2***.

Задача Б. Для обчислення інтегралу по заданому методу створюємо за допомогою ***Consol Wizard*** виконуючого файлу С++ програми у вигляді консольного додатка ***MS DOS***. При формуванні виконуючого коду використовуємо умовну компіляцію окремих програмних модулів у залежності від заданого методу розрахунку значення інтеграла функції. Усі вихідні файли програми і файли проекту ***P_Lab2_B*** зберігаємо в папці ***LAB2_B*** з папці ***LAB2*** .

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Завдання для аналізу і обробки результатів роботи запрограмованої C++ програми

Після запуску в роботу ***C***++ програм до задач А1, А2 та Б з лабораторної роботи № 2 для захисту результатів програмування необхідно виконати таке:

- нарисувати блок-схему алгоритму до умовної компіляції файлів прикладної програми з проекту ***P_Lab2_B*** лабораторної роботи № 2;
- нарисувати блок-схему алгоритму до багато-файлової компіляції файлів прикладної програми з проекту ***P_Lab2_A2*** лабораторної роботи № 2.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Методика виконання задачі А1.

***Порядок дій і команд при виконанні програмування задачі А1 для однофайлової компіляції файлу програми*:**

Крок 1. Активізуємо файли проекту для нової прикладної програми з обчислення інтегралу функції і зберігаємо їх на диску в задану папку:

- На диску ***D***:\ створіть папку ***Lab2*** і усередині таки папки: ***Lab2_A1*** та ***Lab2_A2***;
- Запустить у роботу ***C***++ ***Builder*** і в меню ***File*** виконаєте команду ***New***. В результаті відкриється вікно ***New Items***, де виберіть ***Consol Wizard***;
- Установить у вікні ***Consol Wizard*** для програми опцію ***C***++ і збережіть через команду ОК. В заголовку вікна ***C***++ ***Builder*** буде видно назву ***Project1***, а в заголовку вікна редактора кодів буде показуватися назва ***Unit1.cpp*** та на полі редактора кодів буде вставлено для ***C***++ програми такий шаблон функції ***main( )*** :

```
//---------------------------------------------------------------------------
  #pragma hdrstop
//---------------------------------------------------------------------------

 #pragma argsused
 int main(int argc, char* argv[ ])
 {
```

```
        return 0;
    }
//---------------------------------------------------------------------------
```

- У меню ***File*** виберіть команду ***Save Project As*** і з появою запиту від інтегрованого програмувального середовища на збереження файлів проекту змінить назву файлу ***Unit*1** на файл ***U_Lab*2_A1.*cpp***, а назву проекту ***Project*1.*bpr*** змінить на ***P_Lab*2_A1.b*pr***.

Крок 2. Набираємо текст прикладної програми для задачі А1:

- Наберіть у вікні редактора коду наступний текст програми

```
//---------------------------------------------------------------------------
#pragma hdrstop
//---------------------------------------------------------------------------
#include <vcl.h>
#include <system.hpp>
#include <process.h>
#include <stdio.h>
#include <math.h>
#include <conio.h>
#define    F(x)   (5*(x)*(x)+18*(x)-11)
float d[3];
float h;
 /* Функція введення даних*/
  float * input_data(void)
{ char S_1[35],S_2[30],S_3[30],S_4[40];
CharToOem("Уведіть межі інтегрування\n",S_1);
printf(S_1);
CharToOem("Величина a =",S_2);
printf(S_2);
    scanf("%f",&d[0]);
CharToOem("\n Величина b=",S_3);
printf(S_3);
    scanf("%f",&d[1]);
CharToOem("\n Для ординат F(x) припустима різниця E =",S_4);
printf(S_4);
    scanf("%f",&d[2]);
}
    /* Функція пошуку оптимального кроку
      зміни аргументу для  F(x)  */
      float avt_step(float a,float b,float E)
{ float ah;  float time;
 h=(b-a)/2;
 ah=a+h;
gotoxy(40,10);
```

```
    while(fabs(F(a) - F(ah)) > E)
      { h=h/2;
        gotoxy(57,10);
        printf("%f",h);
     time=0;
     while(time < 1000 )   /* затримка часу для спостереження*/
      { time+=0.0001;           /* зміна розрахункового кроку h      */
      }
        ah = a+h;
      }
  char S_1[25];
        CharToOem("\n Оптимальний крок h дорівнює",S_1);
printf(S_1);   printf(" %f",h);
                return h;
            }
            /* Функція підсумовування ординат F(x) */
             float sum_ord(float x0,float xk,float h)
               { float x,s=0;
                 for( x = x0; x <= xk; x += h)
                       s += F(x);
                return s;
               }
            //---------------------------------------------------------------------
            #pragma argsused
            int main(int argc, char* argv[])
            {   char S1[30],S2[30];
            float IL,IR,Itr;
             clrscr();
                input_data();
            avt_step( d[0], d[1], d[2] );
               gotoxy(33,11);
            CharToOem("Обчислення інтегралів\n",S1);
            printf(S1);
             IL=sum_ord( d[0],              /* початок інтервалу інтегрування*/
                       d[1]-h,             /* кінець інтервалу інтегрування*/
                       h )*h;              /* h - крок зміни аргументу F(x) */
             IR=sum_ord( d[0]+h,                  /* початок інтервалу інтегрування*/
                       d[1],                      /* кінець інтервалу інтегрування*/
                       h )*h;                     /* h - крок зміни аргументу F(x) */
            Itr=sum_ord( d[0],           /* початок інтервалу інтегрування */
                       d[1],             /* кінець інтервалу інтегрування*/
                        h )*h            /* h - крок зміни аргументу F(x) */
                        -0.5*F(d[0])*h   /* корекція по ординаті F(a) */
                        -0.5*F(d[1])*h;  /* корекція по ординаті F(b)*/
            CharToOem("Обчислено такі інтеграли \n",S2);
            printf(S2);
            printf("\n  Il = %f",IL);
            printf("\n  IR= %f",IR);
```

```
printf("\n  (IL + IR)/2 =%f",(IL+IR)/2);
printf("\n  Itr=%f",Itr);
  getch();
      return 0;
}
//-------------------------------------------------------------------------------------------
```

Крок 3. Виконайте компіляцію тексту прикладної ***C***++ програми з обчислення інтегралу і перевірте правильність роботи виконуючого консольного коду для ***MS DOS*** при таких значеннях змінних: ***a*** = 10, ***b*** = 45, ***E*** = 0,2 .

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Методика виконання задачі А2.

***Порядок дій і команд при виконанні програмування задачі А2 для багато-файлової компіляції окремих файлів програмних модулів*:**

Крок 1. Відкриваємо додаткову сторінку з закладкою у вікні редактора коду:

- У меню ***File*** виконайте команду ***New*** і виберіть ***Item*** та файл ***C***++. В результаті у вікні редактора коду відкриється додаткова чиста сторінка з закладкою ***File*1.*cpp***;
- На чисту сторінку вставляйте з буфера обміну опис функцій і текст зберігайте командою ***Save AS*** в окремий файл.

Крок 2. Створюємо для функції ***float * input_data*(*void*)** опис виконання її операторів і зберігаємо у файл:

- З файлу ***U_Lab*2_A1**.cpp копіюйте в буфер обміну опис функції і вставте з буфера обміну на сторінку ***File*1.*cpp*** у редакторі коду;
- Збережіть файл з назвою ***Input_ab.cpp*** в папку ***Lab*2_A2**.

```
//-------------------------------------------------------------------
 /* Функція введення даних*/
  float * input_data(void)
        { char S_1[35],S_2[30],S_3[30],S_4[40];
  CharToOem("Уведіть межі інтегрування\n",S_1);
printf(S_1);
  CharToOem("Величина a =",S_2);
printf(S_2);
```

```
        scanf("%f",&d[0]);
    CharToOem("\n Величина b =",S_3);
printf(S_3);
        scanf("%f",&d[1]);
    CharToOem("\nДля ординат F(x) припустима різниця E =",S_4);
printf(S_4);
                    scanf("%f",&d[2]);
                  return d;
                 }
//---------------------------------------------------------------------
```

Крок 3. Виділяємо через буфер обміну опис функції для макросу ***F*(*x*)** з файлу ***U_Lab*2_A1.*cpp*** та зберігаємо у файл з назвою ***F_x.cpp*** в папку ***Lab*2_A2**:

- Оголошення макросу запишіть у такому вигляді:

```
#define    F(x)   (5*(x)*(x)+18*(x)-11)
```

Крок 4. Створюємо для функції ***float sum_ord*(*float x*0, *float xk*, *float h*)** опис її роботи та зберігаємо у файл:

- З файлу ***U_Lab2*_A1.*cpp*** копіюйте в буфер обміну для визначення функції і далі вставте з буфера обміну на додаткову сторінку в редакторі коду та збережіть у файл з назвою ***Sum_ord.cpp*** і в папку ***Lab*2_A2:**

```
//---------------------------------------------------------------------
/* Функція підсумовування ординат F(x) */
 float sum_ord(float x0,float xk,float h)
   { float x,s=0;
     for( x=x0; x<=xk; x+=h)
           s += F(x);
    return s;
   }
//---------------------------------------------------------------------
```

Крок 5. Створюємо для функції ***float avt_step*(*float a*, *float b*, *float E*)** опис виконання операторів та зберігаємо у файл:

- З файлу ***U_Lab*2_A1.*cpp*** копіюйте в буфер обміну опис функції і вставте з буфера обміну на додаткову сторінку в редакторі коду та збережіть у файл з назвою ***Avt_step.cpp*** і в папку ***Lab*2_A2.**

```
//---------------------------------------------------------------

/* Функція пошуку оптимального кроку зміни аргументу для  F(x)*/
                float avt_step(float a,float b,float E)
                     { float ah;  float time;
          h=(b-a)/2;
         ah=a+h;
        gotoxy(40,10);
     while(fabs(F(a) - F(ah)) > E)
        { h=h/2;
          gotoxy(57,10);
          printf("%f",h);
       time=0;
       while(time < 1000 )   /* затримка часу для спостереження */
        { time+=0.0001;            /* змін розрахункового кроку h*/
        }
          ah=a+h;
         }
  char S_1[25];
         CharToOem("\n Оптимальний крок h дорівнює",S_1);
printf(S_1);   printf(" %f",h);
                    return h;
                      }
//---------------------------------------------------------------------
```

<u>Крок 6</u>. Створюємо основний програмний модуль для багато-файлової компіляції методом вказівок препроцесору і зберігаємо у файл:

- З файлу ***U_Lab2_A1.cpp*** копіюйте в буфер обміну оператори функції ***int main( )*** і вставте з буфера обміну на додаткову сторінку у вікні редактора коду та збережіть у файл з назвою ***U_Lab2_A2.cpp*** і в папку ***Lab2_A2:***

```
  //---------------------------------------------------------------------------
#pragma hdrstop
//---------------------------------------------------------------------------
#include <vcl.h>
#include <system.hpp>
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include "D:\Lab2\Lab2_A2\F_x.cpp"
float d[3];
float h;
#include "D:\Lab2\Lab2_A2\ Input_abE.cpp"
#include "D:\Lab2\Lab2_A2\ Avt_step.cpp"
#include "D:\Lab2\Lab2_A2\ Sum_ord.cpp"
//---------------------------------------------------------------------------
```

```
#pragma argsused
int main(int argc, char* argv[])
{

    float IL,IR,Itr;
     clrscr();
        input_data();
    avt_step( d[0], d[1], d[2] );
       gotoxy(33,11);
    char S1[30],S2[30];
    CharToOem("Визначення інтегралів \n",S1);
    printf(S1);
     IL=sum_ord( d[0],              /* початок інтервалу інтегрування*/
              d[1]-h,               /* кінець інтервалу інтегрування*/
              h )*h;                /* h - крок зміни аргументу F(x)*/
     IR=sum_ord( d[0]+h,                    /* початок інтервалу інтегрування*/
              d[1],                         /* кінець інтервалу інтегрування*/
              h )*h;                        /* h - крок зміни аргументу F(x)*/
    Itr=sum_ord( d[0],            /* початок інтервалу інтегрування*/
              d[1],               /* кінець інтервалу інтегрування*/
               h )*h              /* h - крок зміни аргументу F(x)*/
               -0.5*F(d[0])*h     /* корекція по ординаті F(a)*/
               -0.5*F(d[1])*h;    /* корекція по ординаті F(b)*/
    CharToOem("Обчислено інтеграли \n",S2);
    printf(S2);
    printf("\n  Il = %f",IL);
    printf("\n  IR= %f",IR);
    printf("\n  (IL + IR)/2 =%f",(IL+IR)/2);
    printf("\n  Itr=%f",Itr);
  getch();
     return 0;
}
//---------------------------------------------------------------------------
```

Крок 7. Виконуємо багато-файлову компіляцію програмних модулів ***Input_ab.cpp***, ***F_x.cpp***, ***Avt_step.cpp***, ***Sum_ord.cpp*** та ***U_Lab2_A2.cpp*** і перевіряємо роботу програми в середовищі ***MS DOS***:

- Перейдіть у вікні редактора коду на додаткову сторінку з закладкою ***U_Lab2_A2.cpp*** і зробіть у вікні щиглик правою клавішею мишки, а потім у меню команд виберіть команду ***Close Page*** , що викликає закриття додаткового вікна ***U_Lab2_A2.cpp*** ;
- Відкрийте новий проект для багато-файлової компіляції за допомогою

вказівок препроцесору. Для цього через меню ***File/New*** чи швидкою кнопкою ***New*** відкрийте вікно ***New Items*** і виберіть ***Console Wizard***. Задайте в полі ***Source Type*** опцію ***C***++ , щоб у результаті на екрані було наступне:

- в заголовку вікна редактора кодів встановиться назва ***Unit1.cpp***;
- в заголовку вікна ***C***++ ***Builder*** з'явиться назва проекту ***Project2***.

- Завантажте з папки ***Lab2_A2*** файл ***U_Lab2_A2.cpp*** і у вікні редактора кодів будуть видні дві закладки: одна ***Unit1.cpp*** інша ***U_Lab2_A2.cpp*** ;
- Копіюйте через буфер обміну текст із файлу закладки ***U_Lab2_A2.cpp*** у вікно ***Unit1.cpp*** , а потім через праву клавішу мишки і команду ***Close Page*** закрийте вікно ***U_Lab2_A2.cpp*** ;
- Збережіть новий проект командою ***File/Save Project As…*** у папку ***Lab2_A2*** і при запиті на збереження задайте замість ***Unit1.cpp*** назву ***U_Lab2_A2.cpp***, а замість ***Project2.bpr*** запишіть файлу назву ***P_Lab2_A2.bpr*** ;
- Виконайте команду ***Run*** для компіляції окремих програмних модулів і перевірки виконання ***C***++ програми в ***MS DOS*** при таких значеннях змінних:

$$a = 10; \quad b = 45; \quad E = 0{,}2\ .$$

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Методика виконання задачі Б.

***Порядок дій і команд при виконанні програмування задачі Б для умовної багато-файлової компіляції програмних модулів*:**

Крок 1. Копіюємо файли програмних модулів для умовної компіляції:

- Створіть папку ***Lab2_B*** на диску ***D***:\ у папці ***Lab2***;
- Скопіюйте файли **Input_abE.cpp**, **Avt_step.cpp**, **Sum_ord.cpp** із папки ***Lab2_A2*** в папку ***Lab2_B***.

Крок 2. Створюємо текст файлу ***F_x_B.cpp*** у папці ***Lab2_B***:

- Запишіть у файл ***F_x_B.cpp*** наступні оператори і коментарі:

```
//---------------------------------------------------------------------

/* F_x_B.cpp */
//--Коментарій – зняти для обраного методу інтегрування
//--при виконанні умовної компіляції
//----#define METOD ('PR')      /* Метод прямокутників*/
//----#define METOD ('TR')      /* Метод трапецій*/
#define F(x)  sqrt(0.3*x+1.2)/(1.6*x+sqrt(x*x+0.5))
//---------------------------------------------------------------------
```

Крок 3. Активізуємо програмний майстер ***Consol Wizard*** та проект ***C***++ програми:

- Виконайте команду ***New*** у меню ***File*** в керуючому вікні ***C***++ ***Builder***, щоб відкрилося вікно ***New Items***, де виберіть ***Consol Wizard*** ;
- Установить у вікні ***Consol Wizard*** опцію ***C***++ та збережіть через команду ОК. В результаті у вікні ***C***++ ***Builder*** буде видна назва ***Project*2**, а в заголовку вікна редактора коду буде назва ***Unit*1.*cpp*** і в полі редактора буде записаний наступний шаблон програми:

```
//---------------------------------------------------------------------
                #pragma hdrstop
//---------------------------------------------------------------------
               #pragma argsused
               int main(int argc, char* argv[])
              {
                    return 0;
              }
 //---------------------------------------------------------------------
```

- Виконайте в меню ***File*** команду ***Save Project As*** та змінить назву файлу ***Unit*1** на файл ***U_Lab*2_B.*cpp***, а назву проекту ***Project*2.*bpr*** на назву ***P_Lab*2_B.*bpr*** .

Крок 4. Для обчислення інтегралу по методу прямокутників створюємо у вікні редактора коду програмний модуль ***U_Integ_pr.cpp*** :

```
/*    U_Integ_prB.cpp    */

//---------------------------------------------------------------------
#include <vcl.h>
#include <system.hpp>
#include <stdio.h>
#include <math.h>
#include <conio.h>
float d[3];
```

```
float h;
#include "D:\Lab2\Lab2_B\Input_abE.cpp"
#include "D:\Lab2\Lab2_B\Avt_step.cpp"
#include "D:\Lab2\Lab2_B\Sum_ord.cpp"
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{   float IL,IR;
  clrscr();
     input_data();
avt_step( d[0], d[1], d[2] );
    gotoxy(33,11);
              char S1[30],S2[30];
              CharToOem("Визначення інтегралів \n",S1);
              printf(S1);
 IL=sum_ord( d[0],     /* початок інтервалу інтегрування*/
           d[1]-h,   /* кінець інтервалу інтегрування*/
           h )*h;    /* h - крок зміни аргументу F(x)*/
 IR=sum_ord( d[0]+h, /* початок інтервалу інтегрування*/
           d[1],   /* кінець інтервалу інтегрування*/
           h )*h;  /* h - крок зміни аргументу F(x)*/
              CharToOem("\n    ІНТЕГРАЛ  ДОРІВНЮЄ",S2);
              printf(S2);
printf("\n IL= %f",IL);
printf("\n IR= %f",IR);
printf("\n (IL + IR)/2 = %f",(IL+IR)/2);
   getch();
       return 0;
}
//---------------------------------------------------------------------------
```

Крок 5. Для обчислення інтегралу по методу трапецій створюємо у вікні редактора коду програмний модуль ***U_Integ_tr.cpp*** :

```
/* U_Integ_trB.cpp */
//---------------------------------------------------------------------------
#include <vcl.h>
#include <system.hpp>
#include <stdio.h>
#include <math.h>
#include <conio.h>
float d[3],h;
#include "D:\Lab2\Lab2_B\Input_abE.cpp"
#include "D:\Lab2\Lab2_B\Avt_step.cpp"
#include "D:\Lab2\Lab2_B\Sum_ord.cpp
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
```

```
{    float Itr;
  clrscr();
     input_data();
avt_step( d[0], d[1], d[2] );
    gotoxy(33,11);
              char S1[30],S2[30];
              CharToOem("Визначення інтегралів \n",S1);
              printf(S1);
Itr=sum_ord( d[0],     /* початок інтервалу інтегрування*/
           d[1],     /* кінець інтервалу інтегрування */
            h )*h    /* h - крок зміни аргументу F(x)*/
            -0.5*F(d[0])*h  /* корекція по ординаті F(a)*/
            -0.5*F(d[1])*h; /* корекція по ординаті F(b)*/
              CharToOem("\n ІНТЕГРАЛ  ДОРІВНЮЄ",S2);
              printf(S2);
printf("\n  Itr = %f",Itr);
  getch();
      return 0;
}
//---------------------------------------------------------------------------------------------
```

Крок 6. Створюємо у вікні редактора коду основний програмний модуль ***U_Lab2_*B*.cpp***:

```
/ *U_Lab2_B.cpp* /
//------------------Умовна компіляція файлів--------------------
//----------------------закрита зона-----------------------------
#pragma hdrstop
//----------------------закрита зона-----------------------------
#include "F_x_B.cpp"
#ifdef METOD
#if METOD == 'PR'
#include "U_Integ_prB.cpp"
#else
#include "U_Integ_trB.cpp"
#endif
#else
#include <conio.h>
//---------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
clrscr();
  window(10,10,80,25);
    textcolor(WHITE+BLINK);
              char S1[30],S2[30];
              CharToOem("\n Інтеграл не обчислюється",S1);
              cprintf(S1);
```

```
   textbackground(YELLOW);
  textcolor(BLACK);
  gotoxy(12,4);
              char S1[30],S2[30];
              CharToOem("\n Ви забули задати метод інтегрування",S2);
              cprintf(S2);
  gotoxy(12,6);
  textcolor(WHITE);
  cputs("  У файлі -> F_x_B.cpp задайте метод інтегрування функції");
  textmode(LASTMODE);
      getch();
       return 0;
}
#endif
//---------------------------------------------------------------------
```

Крок 7. У файлі ***F_x_*B.*cpp*** задаємо обчислення інтегралу методом прямокутників і перевіряємо правильність роботи ***C***++ програми:

- Виконайте умовну компіляцію командою ***Run*** або ***F*9** для файлу ***U_Lab2_*B.*cpp*** і перевірте правильність роботи ***C***++ програми у вікні ***MS DOS*** для таких значень: ***a*** = 10; ***b*** = 45; ***E*** = 0,2.

Крок 8. У файлі ***F_x_*B.*cpp*** задаємо обчислення інтегралу методом трапецій і перевіряємо правильність роботи ***C***++ програми:

- Виконайте умовну компіляцію командою ***Run*** або ***F*9** для файлу ***U_Lab2_*B.*cpp*** і перевірте правильність роботи програми у вікні ***MS DOS*** для таких значень: ***a*** = 10; ***b*** = 45; ***E*** = 0,2.

Крок 9. Відновлюємо у файлі ***F_x_*B.*cpp*** наступні коментарі (не задаємо ***METOD***)

```
//---- #define METOD ('PR')      /* Метод прямокутників*/
//---- #define METOD ('TR')      /* Метод трапеції*/
```

і перевіряємо правильність роботи ***C***++ програми:

- Виконайте умовну компіляцію командою ***Run*** або ***F*9** для файлу ***U_Lab2_*B.*cpp*** і перевірте правильність роботи прикладної ***C***++ програми у вікні ***MS DOS***.

Контрольні запитання до лабораторної роботи № 2

1. Пояснить призначення багато-файлової компіляції файлів при одержанні виконавчого коду програми.
2. Пояснить, яка компіляція файлів називається "умовною".
3. Як налаштовується ***C++ Builder*** для одержання коду консольної прикладної ***C++*** програми для виконання у ***MS DOS***.
4. Які види вказівок препроцесору застосовуються при розробці прикладної ***C++*** програми.
5. Яка функція виконує узгодження кодів для шрифтів кирилиці.

Лабораторна робота № 3

Команди редактора C++ Builder і структура створюваних файлів до проекту прикладної програми

Ціль виконання лабораторної роботи. В процесі розробки прикладної програми за допомогою ***C++ Builde*** створюється набір файлів з назвою – проект файлів і тому вивчається таке:

- структура файлів проекту та їх призначення в забезпеченні виконання прикладної програми;
- правила і техніка роботи з компонентами бібліотеки ***VCL*** при розробках проектів файлів до прикладних програм;
- властивості та команди редактора коду ***C++ Builder*** для підключення описів функцій у програмні модулі форм;
- способи виконання багато-файлової компіляції з отриманням двох видів виконуючих файлів з різними можливостями їх виконання на різних комп'ютерах.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Особливості формування структури файлів у проекті C++ Builder

За допомогою ***C++ Builder*** можна розробляти прикладні ***C++*** програми з проектами файлів двох видів: перший вид це автономні виконуючи файли

(***.exe***); другий вид це проекти файлів до прикладної ***C***++ програми у вигляд пакетів (***packages***) часу виконання.

Для окремого проекту файлів прикладної програми доцільно створювати автономний виконуючий файл, в якому розміщується ***C***++ програма та всі необхідні програмні ресурси. Розміри таких файлів у ***C***++ ***Builder*** створюються досить невеличкі в порівнянні з іншими системами, що створюють автономні модулі. При роботі такого автономного виконуючого файлу на комп'ютері користувача не потрібно установка ***C***++ ***Builder*** , або яких-небудь інших спеціальних бібліотек. Прагнення зменшити витрати на збереження і поширення виконуючих файлів привело до концепції застосування пакетів.

Пакети (***packages***) це спеціальні динамічні бібліотеки (***DLL***), що містять файли візуальних компонент, інші об'єкти, функції, процедури і ін., яки приєднуються к проекту файлів прикладної програми Ці ***DLL*** бібліотеки дозволяють створювати невеликі виконуючи модулі, що звертаються за підтримкою до пакетів. Можна також скомпілювати в пакети свої власні компоненти і бібліотеки. Файли пакетів (***Borland package library***) мають розширення (***.bpl***), щоб відрізнятися від звичайних ***DLL***. Програми у вигляді пакетів підрозділяються на *пакети часу проектування* та *пакети часу виконання*.

Пакети часу проектування сама ***C***++ ***Builder*** викликає в процесі проектування прикладної програми. Файли цього пакету використовуються тільки інтегрованим середовищем ***C***++ ***Builder***. Цей пакет існує завжди тимчасово і постійно змінюється по ходу процесу проектування прикладної програми.

Пакети часу виконання можуть містити такі елементи: бібліотеки візуальних компонентів ***C***++ ***Builder***; замовлені пакети інших розроблювачів; розроблені вами нові компоненти; придбані інші комерційні візуальні компоненти. При реалізації пакетів часу виконання повинні передаватись користувачу не тільки виконуючий модуль, але і усі файли з пакету часу

виконання, котрі повинні використовуватись прикладною програмою. Розміри виконуючих модулів істотно скорочуються (приблизно в 10 разів менше) за рахунок того, що велика частина програмних кодів міститься в цих пакетах. Інтегроване програмувальне середовище ***C++ Builder*** при виконанні команд ***Project / Build*** створюють автономний виконуючий модуль кодів ***project.exe***, що виконується, без підтримки пакетів. Для створення пакета файлів часу виконання необхідно попереднє виконати таки налаштування:

– в меню команд вибираємо ***Project / Options***;

– переходимо на закладку ***Packages***(пакети);

– встановлюємо прапорець для режиму ***Built with runtime packages*** та зберігаємо кнопкою ОК.

Після налаштування компіляції в режимі ***Packages***(пакети) для вже створеного проекту ***project.exe*** потрібно виконати команди ***Project / Build***.

Які пакети часу виконання і бібліотеки ***DLL*** потрібні для роботи прикладної програми в середовищі ***Windows*** можна з'ясувати за допомогою спеціальної програми ***tdump.exe***, що входить до складу інтегрованого середовища ***C++ Builder*** і зберігається в каталозі ***...\bin***. Ця програма працює в режимі ***DOS*** і тому необхідно перейти в ***DOS*** і увійти в каталог, де знаходяться файли проекту прикладної програми і далі в командному рядку виконати наступну команду (***tdump Project1.exe > dump.txt***). Програма ***tdump.exe*** проаналізує файл, що виконується, і результати аналізу внесе в зазначений текстовий файл ***dump.txt***. У цьому файлі можна буде бачити список пакетів і бібліотек ***DLL*** , що будуть використовуватися при роботі прикладної програми. Пакети і бібліотеки з цього списку повинні обов'язково бути записаними на комп'ютері користувача, щоб прикладна програма могла працювати в ***Windows***. При проектуванні прикладної програми для ***Windows*** створюється проект (комплект файлів) і в таблиці № 3-1 показано список файлів, що створює інтегроване середовище ***C++ Builder***.

Таблиця № 3-1.

Назва файлу	Призначення файлу
Головний файл проекту (.cpp)	C++ Builder створює файл .cpp для головної функції WinMain, ініціалізує програму і запускає в роботу.
Файл опцій проекту (.bpr)	Цей текстовий файл містить установки опцій проекту і вказівки на те, які файли повинні компілюватися і компонуватися у проект. Зберігається файл у форматі XML.
Файл ресурсів проекту (.res)	Файл, що містить ресурси проекту: піктограмки, курсори, значки іконок і т.п. За замовчуванням містить тільки піктограму проекту і може доповнюватися за допомогою "редактора зображень".
Файл реалізації модуля (.cpp)	Кожній створюваній формі відповідає текстовий файл реалізації модуля, використовуваний для збереження вихідного коду. Можна також створювати програмні модулі, не зв'язані з формами.
Заголовний файл модуля (.h)	Кожній створюваній формі відповідає не тільки файл реалізації модуля, але ще і заголовні файли з описом класу форми. Можна створювати додаткові необхідні заголовні файли.
Файл форми(.dfm)	Двоічний чи текстовий файл, у якому зберігаються дані про створені форми C++ Builder. Цей файл можна переглядати в текстовому виді чи у виді форми (.cpp).
Заголовний файл компоненти (.hpp)	Файл створюється при створенні нового компонента. Також часто ці файли підключаються до проекту з бібліотеки компонентів, розташованих у каталозі Include/VCL.
Файл групи проектів(.bpg)	Текстовий файл , створюваний у C++ Builder при створенні групи проектів.
Файли пакетів(.bpl) і (.bpk)	Ці файли використовує C++ Builder при роботі з пакетами: .bpl - файл самого проекту; .bpk - файл, що визначає компіляцію і компонування проекту.
Файл стола проекту (.dsk)	У цьому текстовому файлі C++ Builder зберігає інформацію про останній сеанс роботи з проектом, про відкриті вікна, їхніх розмірах і положенні. Завдяки цьому файлу в новому сеансі роботи з проектом автоматично встановлюється на екрані теж розташування вікон, що і було при попередньому сеансі роботи. Файл створюється якщо була включена опція Avtosave/ Project desktop.
Файли резервних копій (.~bp, .~df, .~cp, .~h)	Це відповідні файли резервних копій для файлів проекту, форми, реалізації модуля і заголовного файлу. Якщо безнадійно щось зіпсувалося в проекті, то можна відповідно змінити розширення цих файлів і в такий спосіб повернуться до попереднього не зіпсованому варіанту проекту.

Група файлів створюваних у процесі компіляції. Таблиця № 3-2.

Файл, що виконується,(.exe)	Файл програми, що виконується.
Об'єктний файл модуля (.obj)	Файл модуля (.cpp) після обробки вказівок препроцесору і компіляції, який редактором зв'язків компонується в остаточний файл, що виконується.
Бібліотека, що динамічно приєднується, (.dll)	Файл створюється у випадку, якщо ви проектуєте свою власну DLL.
Файл таблиці символів (.tds)	Файл, використовуваний відладчіком у процесі налагодження програми.
Файли вибіркового компонування (.il)	Файли з розширеннями, що починається з il (.ile, .ild, .ilf, .ils), дозволяють повторно компонувати тільки ті файли, що були змінені після останнього сеансу.

Редактор кодів ***C++ Builder*** є повноцінним програмним редактором. Його можна налаштовувати на різний стиль роботи, що вам більш підходить. У редакторі застосовується виділення кольором і шрифтом синтаксичних елементів тексту програми. Жирним шрифтом виділяються ключові слова мови ***C++***. Зеленим кольором виділяються директиви препроцесора, яки починаються із символу #. Синім кольором і курсивом виділяються коментарі, що починаються із символу двійна коса (//).

Файли ***Windows***, яки можуть входити до складу проекту ***C++ Builder***.
Таблиця № 3–3.

Файли довідки (.hlp)	Стандартні файли довідки Windows, що можуть використовуватися в розроблювальному додатку.
Файли зображень або графічні файли (.wmf, .bmp, .ico)	Ці файли звичайно використовуються в додатках Windows для створення привабливого і дружнього інтерфейсу.

У заголовку вікна “Редактор Коду” відображається ім'я поточного файлу, того, з яким виконується робота. Часто в ***C++ Builder*** приходиться працювати з декількома файлами. Зокрема, звичайно крім файлу реалізації модуля (***.cpp***) приходиться працювати з заголовним файлом модуля (***.h***) . Можна завантажити заголовний файл у "Редактор Коду", клацнувши у вікні редактора правою клавішею миші і вибрати з контекстного меню команду ***Open Source/Header File***. Якщо в цей момент у вікні "Редактор Коду"

знаходився текст файлу реалізації модуля форми, то в редактор додається автоматично окремий лист вікна з закладкою і у нього завантажиться заголовний файл модуля форми. У нижній частині редактора будуть видні корінці закладок. За допомогою корінців закладок можна швидко переходити з одного файлу в іншій. Якщо якийсь з файлів більше не потрібний, то можна закрити цю сторінку з файлом, вибравши в контекстному меню команду ***Close Page***. Можна також відкрити додаткове вікно в редакторі коду через команди: у меню команд ***View/New Edit Window*** або через праву кнопку миші аналогічну команду з контекстного меню. Це дозволяє працювати одночасно з декількома модулями та з різними фрагментами одного модуля форми.

У нижній частині вікна "Редактор Коду" знаходиться рядок стану. В самій лівій позиції видно індикатор рядка і стовпчика, що вказує розташування курсору. Цей індикатор допомагає швидко зрозуміти, у якому місці тексту з файлу ви знаходитеся. Другий елемент рядка стану - це індикатор модифікацій. Якщо вносилися зміни в текст файлу і команда ***File / Save*** не виконувалась, то в індикаторі видне повідомлення ***Modified*** тобто код тексту відрізняється від файлу, збереженого на диску. Третій елемент рядка стану – це індикатор режиму вставки. Цей індикатор показує, будуть символи, що вводяться, або вставлятися в текст чи записуватися поверх тексту. Переключення режиму вставки виконується клавішею ***Insert.***

Вікно “Редактор Коду” на екрані може містити вбудоване вікно "Дослідник класів" (***ClassExplorer***). Дослідник класів показує для завантаженого проекту дерево всіх типів, класів властивостей, методів глобальних перемінних і глобальних функцій, що містяться у файлі модуля чи форми іншому файлі. За замовчуванням вікно "Дослідник Класів" з'являється автоматично вбудованим у вікно "Редактор Коду". Це розташування за замовчуванням може бути зміненим відключенням опції ***Automaticaly show Explorer*** на сторінці ***ClassExplorer*** при виконанні команди ***Tools / Environment Options***. У цьому випадку при необхідності можна викликати "Дослідник Класів" командою ***View / ClassExplorer.***

В інтегрованому середовищі , як і у віконних компонентах C++ Builder, можна використовувати технологію ***Drag & Doc*** - перетаскування і вбудовування вікон. На ряду з вбудованим вікном ***ClassExplorer*** також може бути вбудоване: вікно "Менеджера проектів" (***Project Menager***), вікно величин, що спостерігаються, (***Watch List***) і ін. вікна. Вікно, що вбудовується, можна відрізнити від звичайного вікна по наступним ознаках:

- ✓ Скорочена смужка системного заголовка вікна з одною кнопкою закриття вікна;
- ✓ Наявність у меню, що спливає у вікні при щиглику правою кнопкою миші, перемикача ***Dockable*** - що вбудовується. Якщо зняти мітку з цього перемикача, вікно перестане бути таким, що вбудовується. Надалі можна знову позначити цей перемикач, і вікно знову стане що вбудовується.

При перетаскуванні вікна, що вбудовується, розміри його рамки змінюються, якщо вікно переміщається в межах іншого вікна. Вбудовування вікон дозволяє ощадливо використовувати площу екрана. Для того, щоб перемістити вікно, що вбудовується, треба потягнути курсором миші за подвійну лінію над однією з границь вікна, що вбудовується. При цьому можна витягти його з вікна - контейнера і зробити самостійним вікном - так називаним вікном, що плаває. Для перетворення вбудованого вікна в стан, що плаває, не обов'язково тягти за подвійну виступаючу лінію - досить зробити два щигликів на цій подвійній лінії.

Вікно ***ClassExplorer*** можна розмістити в нижній частині вікна редактора коду, щоб можна було бачити велику довжину тексту коду програми. Найбільше зручно умонтувати вікно ***ClassExplorer*** у вікно "Інспектор Об'єктів" (***Object Inspector***). При цьому утворяться окремі сторінки, корінці яких показуються угорі вікна ***Object Inspector***. У цьому випадку вікно ***ClassExplorer*** зовсім не займає на екрані додаткового місця і, коли необхідно переглянути необхідне вікно, досить клацнути відповідну закладку корінець. Оскільки не усі закладки убудованих вікон можуть

уміститися в заголовку вікна ***Object Inspector***, то в цьому випадку біля корінців закладок з'являються зі стрільцями кнопки для переміщення корінців закладок.

В ***C++ Builder*** "Редактор Коду" має такі вбудовані засоби для розробки вихідних текстів коду:

– інструмент - "Знавець Коду" (***Code Insight***);

– засоби швидкої навігації по тексту файлу і коректування коду.

Робота з Code Insight.

Цей інструмент вбудований у вікно редактора коду і може надавати велику допомогу при написанні і налагодженні кодів. У багатьох випадках ***Code Insight*** підкаже імена властивостей, методів, подій, типи аргументів, типові синтаксичні конструкції і багато чого іншого. ***Code Insight*** може застосовуватися в двох режимах: автоматичному та не автоматичному. Автоматичний режим включений за замовчуванням. Можна відключити автоматичний режим роботи і його викликати при необхідності за допомогою таких клавіш: "Ctrl +Shift + пробіл" чи "Ctrl + пробіл" у залежності від того, до яких можливостей ***Code Insight*** потрібно звернутися. Використовується “Знавець Коду” для таких функцій:

✓ *Завершення коду*

Автоматично дописується закінчення коду по першим набраним символам команди:

– Якщо ви написали у своєму додатку ім'я компоненти, поставили після нього символи стрілки (->) і небагато затрималися з введенням наступного тексту, то з'явиться вікно, що містить список усіх властивостей, методів і подій класу, до якого належить даний компонент. Можна з цього списку вибрати необхідну назву або почати писати перші символи властивості методу, а потім натиснути ***Enter***, і в початий код додається відповідне ім'я. Так працює ***Code Insight*** в автоматичному режимі. Якщо автоматичний режим відключений, то можна викликати ту ж саму підказку, якщо,

набравши символи стрілки (->) після імені компонента, натиснете “Ctrl + пробіл” ;

– Якщо ви написали символ операції присвоювання (=) і натиснете "Ctrl + пробіл", то буде показаний список можливих аргументів, сумісних по типу з перемінною, до котрої буде привласнювання значення. Аналогічним образом можна одержати підказки по аргументах функцій і процедур.

✓ *Параметри функцій, процедур, методів*

Якщо ***Code Insight*** працює в автоматичному режимі, то після того, як буде написано ім'я функції чи методу і буде поставлена відкриваюча дужка, тоді з'явиться список параметрів і їхніх типів. Причому, по мере того, як будуть вводитися значення аргументів, знавець коду буде висвічувати тип наступного параметра. Якщо автоматичний режим відключений, то цю підказку можна одержати натисканням клавіш "Ctrl +Shift + пробіл".

✓ *Шаблони коду*

У помічнику ***Code Insight*** записана безліч шаблонів стандартних структур мови С++. Причому сам програміст зможе додавати чи видаляти ці шаблони. Виклик шаблона виконується натисканням клавіш "Ctrl + J". Зі списку, що випадає, можна вибрати потрібний шаблон. Наприклад, якщо обраний шаблон циклу ***for*** , то в текст коду додасться:

```
    for(  ;  ; )
{
        }
```

Залишається лише тільки заповнити заголовок циклу і написати тіло циклу.

✓ *Оцінка виразів*

Ця здатність застосовується при налагодженні ***C***++ програми. ***Code Insight*** дозволяє при зупинці для покрокового виконання програми підвести курсор у вікні “Редактора Коду” до імені будь-якої змінної чи виразу та побачити значення оцінюваної величини.

✓ *Інформація про ідентифікатори– Code browser*

Якщо задане автоматичне виконання цього режиму ***Code Insight***, то при переміщенні курсору миші у тексті ***C***++ програми над будь-якою змінною автоматично з'являється інформація про оголошення, тип і про модуль і про номер рядка, що містить це оголошення. Можливості ***Code Insight*** істотно розширюються, якщо натиснути клавішу ***Ctrl*** і не відпускати при перегляді коду тексту. У випадку переміщення курсору мишки над будь-яким ідентифікатором, коли ідентифікатор виділяється кольором і підкресленням, курсор прийме вид руки. Якщо зробити щиглик на виділеному ідентифікаторі, то у вікні "Редактор Коду" відкриється файл, що містить оголошення і курсор встановиться на рядок цього оголошення. Причому ця можливість виконується незалежно від автоматичного і не автоматичного режиму ***Code Insight***. Інформацію про нові ідентифікатори не збережені у файлах ***Code browser*** знаходити не може.

Виконання швидкої навігації.

✓ *Контекстні меню редактора коду.*

У контекстних меню зосереджені багато інструментальних функцій:

– Якщо в момент щиглика курсор був на імені компоненти, властивості, методу, константи, перемінної, функції і т.д., тоді в контекстному меню з'являється розділ ***Find Declaration*** - знайти оголошення. Вибравши цей розділ, можна бачити оголошення даного елемента коду. Якщо цей об'єкт (змінна, функція, і т.д.) визначені в модулі, завантаженому у вікно редактора коду, то курсор просто переміститься на рядок, що містить оголошення.

– Якщо ж об'єкт визначений у якомусь іншому модулі, то вікно редактора коду буде завантажений відповідний файл модуля і курсор установиться на оголошення. Потім при необхідності можна вивантажити файл модуля через контекстне меню командою ***Close Page***;

– Якщо помістити курсор на імені заголовного файлу, що підключається до проекту директивою #***include***, тоді можна зробити щиглик правою клавішею миші і вибрати в контекстному меню роздягнув ***Open File at Cursor***, тоді у

вікно редактора коду буде завантажений відповідний файл для перегляду оголошень функцій, констант, макросів і ін. об'єктів;

✓ *Закладки коду, що позначаються*

Забезпечується швидке переміщення і навігація по тексту коду ***C***++ програми. Щоб створити закладку потрібно установити курсор на необхідному рядку і викликати контекстне меню, де вибрати команду ***Toggle Bookmarks***. В результаті з'явиться список можливих закладок. У цьому списку можна позначити закладку, яку необхідно прив'язати до даного рядка, клацнувши на ній лівою клавішею мишки. Якщо потрібно видалити раніше позначену закладку, необхідно на ній клацнути правою клавішею мишки. Для швидкого переходу по тексту коду за допомогою закладки необхідно викликати контекстне меню і вибрати в ньому команду ***Goto Bookmarcs***, при цьому з'явиться список закладок, з якого потрібно вибрати необхідну закладку. Якщо потрібні закладки, щоб спостерігати паралельно два різних фрагменти коду, наприклад, звірити чи оператори скопіювати, чи оператори перенести з одного фрагмента тексту коду в іншій, то в цьому випадку спочатку випливає контекстне меню і командою ***New Edit Window*** відкрити додаткове вікно редагування того ж файлу. Тоді можна в одному вікні перейти до однієї закладки, а в іншім вікні до іншої закладки і одночасно працювати з обома фрагментами коду.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Постановка задачі по програмуванню у лабораторній роботі

Необхідно для ***Windows*** створити прикладні програми до наступних задач:

Задача А. Прикладна C++ програма повинна формувати зображення до залежності функції ***F*(*x*)** і для заданого значення ***x*** розраховувати числове значення для ***F*(*x*)**. Після введення коефіцієнтів для функції ***F*(*x*)** і щиглика на кнопці "ПУСК" вікно програми повинно мати вигляд, як показано на рис.3-1. При повторних введеннях інших коефіцієнтів до функції ***F*(*x*)** і щигликах на

кнопці "ПУСК" програма виконує нові розрахунки значення ***F(x)***. Задача А програмується і виконується по двох варіантах проектування:

- для першого варіанта виконуємо задачу А1, у якій створюється проект файлів у вигляді пакета часу виконання для одержання виконуючого файлу прикладної ***C***++ програми;
- по другому варіанту виконуємо задачу А2, де налаштовуємо ***C***++ Builder на одержання автономного виконуючого файлу та за допомогою програми ***tdump.exe*** порівнюємо розмір автономного виконуючого файлу з виконуючим файлом з пакету часу виконання.

Задача Б. Необхідно створити для ***Windows*** прикладну ***C***++ програму у вікні, якої буде кнопка з назвою "Розрахувати" і після щиглика мишкою по кнопки програма в залежності від обраного зі списку числового методу інтегрування визначить(розрахує) значення інтегралу для сформованого зображення до залежності функції ***F(x),*** як це показано на рис. 3-2.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Завдання для аналізу і обробки результатів роботи запрограмованої C++ програми

Після запуску в роботу ***C***++ програми лабораторної роботи № 3 для захисту результатів програмування необхідно виконати таке:

- нарисувати блок-схему алгоритму з роботи програм у задачі А(А1,А2) лабораторної роботи № 3;
- нарисувати блок-схему алгоритму з роботи програм у задачі Б лабораторної роботи № 3;
- нарисувати блок-схему алгоритму по виконанню операторів функції, з обробки в програмі задачі Б події для кнопки «Розрахувати»;
- нарисувати блок-схему алгоритму по виконанню операторів функції, з розрахунків по визначенню оптимального шагу ***h*** для обчислення інтегралу функції.

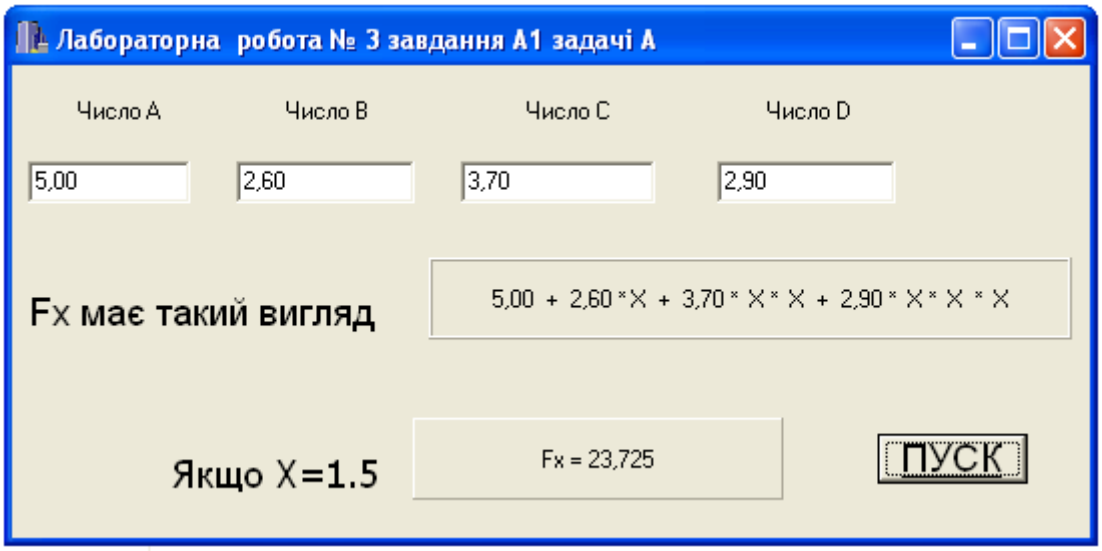


Рис. 3-1. Результат роботи *C*++ програми при виконанні задач А1 та А2.

Методика виконання задачі А1.

***Порядок дій і команд при виконанні програмування задачі А1 зі створенням для програми проекту файлів у вигляді пакету часу виконання*:**

Крок 1. Відкриваємо новий проект для програми задачі А1 і визначаємо назву проекту ***P_Lab*3_A1**:

- Виконайте в керуючому вікні команду ***File/New Application***, щоб з'явилася чиста форма ***Form*1**;
- Перейдіть у вікно інспектора об'єктів і у полі ***Caption*** задайте назву лабораторної роботи "Лаб. робота № 3 задача А1", щоб у заголовку вікна форми змінилась назва ***Form*1.**

Крок 2. Збережіть порожню форму з новим заголовком вікна для активізації нового проекту файлів і для запам'ятовування ***C*++ *Builder*** шляху для швидкого збереження змін у проекті:

- Збережете файли проекту на ***D*:*LA_NN**LAB*_3\A1, де ***NN*** - номер навчальної групи;
- У меню ***File*** виберіть команду ***Save Project As*** і з появою запиту на збереження замініть назву файлу ***Unit*1** на файл ***U_Lab*3_A1.cpp**, а

назву проекту ***Project1.bpr*** замініть на ***P_Lab3_*A1.bpr.**

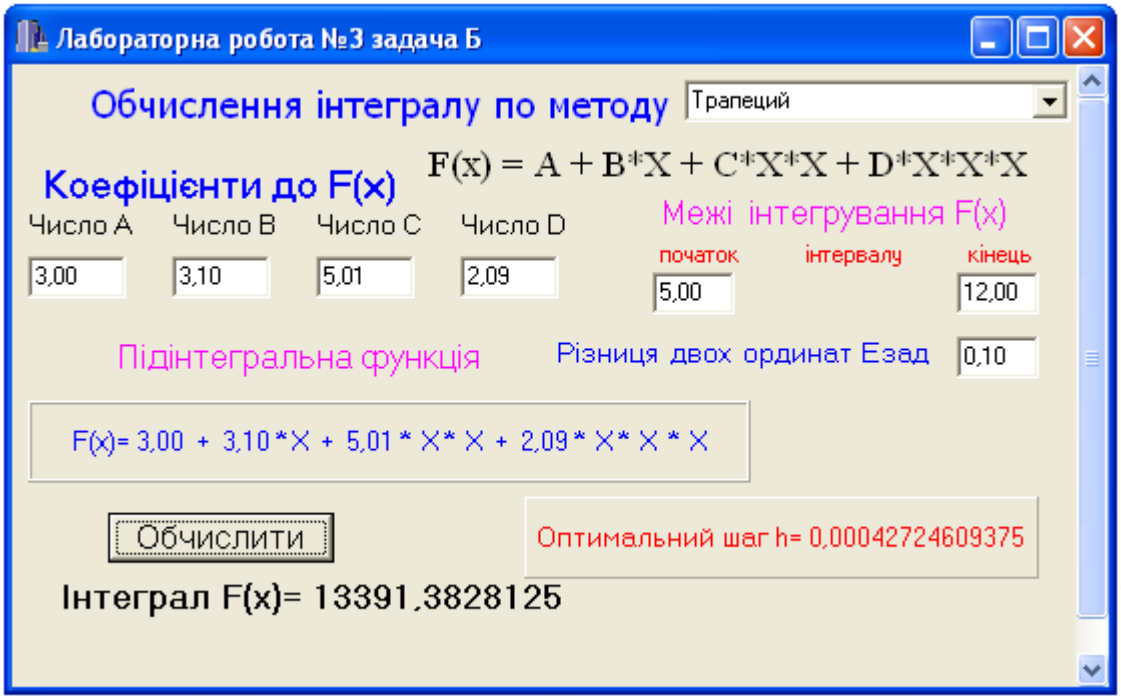


Рис. 3-2. Результат роботи програми *C++* при виконанні задачі Б.

Крок 3. Встановлюємо на форму програми (рис.3-3) компоненти для введення коефіцієнтів ***A, B, C*** та ***D***, щоб можна було для сформувати зображення у вигляді такої функції

$$F(x) = A + B*X + C*X*X + D*X*X*X$$

- Встановить на форму такі компоненти: ***Label*1, *Label*2, *Label*3, *Label*4**, де будуть написи підказки - який коефіцієнт варто вводити;
- Під назвами ***Label*** встановить компоненти ***Edit*1, *Edit*2, *Edit*3, *Edit*4** для введення числових даних до коефіцієнтів функції ***F*(*x*)**;
- Встановить компоненти ***Label*5** та ***Label*6** відповідно до рисунка 3-3;
- Встановить компоненти ***Panel*1, *Panel*2** і кнопку ***Button*1**.

Крок 4. Змінюємо назви ***Label*1, *Label*2, *Label*3** та ***Label*4** на написи: Число ***A***, Число ***B***, Число ***C***, Число ***D***:

- На компоненту Panel1 установить мишкою маркерну рамку;
- Перейдіть у вікно інспектора об'єктів (***Object Inspector***) і на сторінці властивостей об'єкта (***Properties***) у виділеному полі ***Caption*** встановить

курсор на напис ***Panel*1** і цю назву змінить на напис "Число ***A***". Аналогічно зробіть заміни для ***Label*2, *Label*3** та ***Label*4**.

Рис. 3-3. Структура і розташування компонент на формі для завдання А1.

Крок 5. У компонентах ***Edit*** замінюємо написи: ***Edit*1, *Edit*2, *Edit*3** та ***Edit*4** на **0,00** для підказки користувачу в якому форматі потрібно вводити числові значення коефіцієнтів для функції (ціла частина числа відокремлюється комою і перехід робіть клавішею "***Tab***").

Крок 6. На компоненті ***Label*5** задаємо такий напис "***F*(*x*)** має такий вигляд".

Крок 7. Для компоненти ***Label*6** набираємо такий напис "Якщо ***X*** = **1.5**".

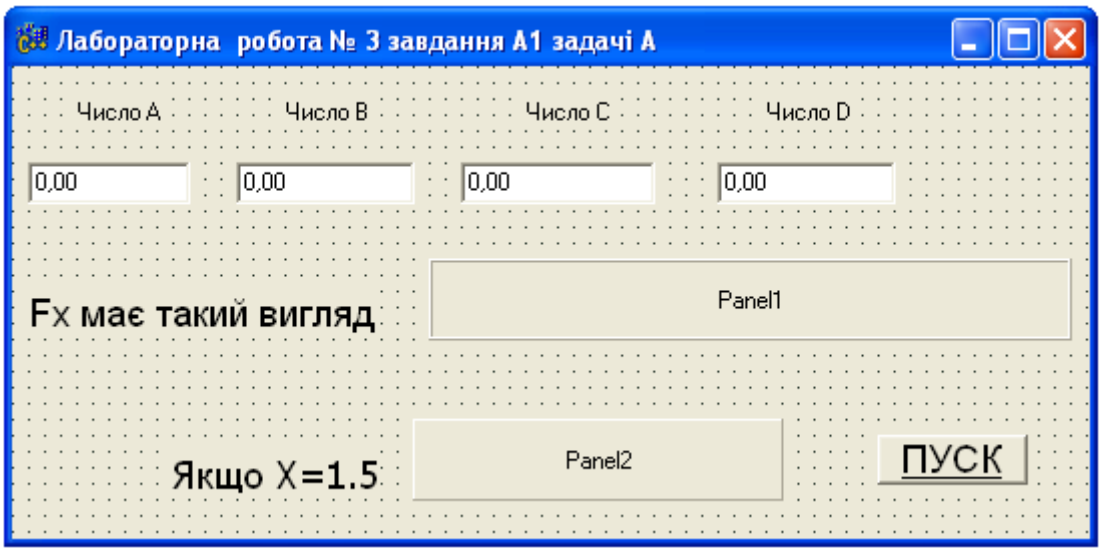


Рис. 3-4. Вигляд форми до завдання А1 у задачі А перед компіляцією файлів.

Крок 8. Змінюємо на формі вікна вигляд у панелі ***Panel*1**:

- Установить властивість ***Bevelinner*** в значення ***bvLowered***.

Крок 9. Змінюємо назву кнопки ***Button*1**:

- Встановіть маркерну рамку на кнопку ***Button*1** та назвіть її "ПУСК".

Крок 10. Визначаємо подію для щиглика на кнопці "ПУСК", щоби на полі компоненти ***Panel*1** у зображенні залежності функції ***F*(*x*)** заповнювалися числові значення коефіцієнтів, а на полі ***Panel*2** повинне з'являтися розрахункове значення до функції ***F*(*x*),** якщо ***X*** = 1.5 (дивись рис. 3-1):

- Виконаєте подвійний щиглик мишкою на кнопці "ПУСК" і перейдіть у вікно інспектора об'єктів на сторінку подій (***Events***);
- У вікні “Редактор Коду ” в функцію ***TForm*1*::Button*1*Click*** додайте такі оператори:

```
//---------------------------------------------------------------------------------------
        Panel1->Caption = Edit1->Text + "  " + "+" + "  " + Edit2->Text + " " +"*" +
" " + "X" + "  " + "+" + "  " + Edit3->Text + " " + "*" + "  " + "X" + " " + "*" +"  " + "X"
+ "  " + "+" +"  " + + Edit4->Text + " " + "*" + "  " + "X" + " " + "*" +"  " +"X" + "  " +
"*" +"  " + “X”;
 Panel2->Caption = "Fx = "+FloatToStr(StrToFloat(Edit1->Text)+ StrToFloat(Edit1-
>Text)*1.5+ StrToFloat(Edit3->Text)*1.5*1.5 + StrToFloat(Edit4->Text)*1.5*1.5*1.5 );
//---------------------------------------------------------------------------------------------
```

Крок 11. Вигляд форми програми перед виконанням компіляції проекту ***P_ Lab*3_A1** повинний бути таким, як показано на рис.3-4.

Крок 12. Компілюємо файли з проекту ***P_Lab*3_A1** і запускаємо на виконання командами ***Run/Run*** або клавішею ***F*9.** Якщо всі налаштування до компонент на формі програми були зроблені правильно, то у вікні програми після введення чисел для коефіцієнтів і щиглика на кнопці "ПУСК" на екрані зображення буде мати вигляд, як показано на рис. 3-1.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

***Порядок дій і команд при виконанні програмування задачі A2 (другого варіанту задачі A) зі створенням для програми проекту файлів у вигляді автономного пакету*:**

Крок 1. Створюємо автономний виконуючий файл проекту ***P_ Lab*3_A2**:

- Виконаєте команди ***Project / Options*** і перейдіть на сторінку (закладку) ***Packages***, де в нижній частині вікна "***Project Options***" видалить прапорець індикатора режиму ***"Build with runtime Packages"*** і натисніть ОК;
- Копіюємо файли з ***D***:***LA_NN\LAB***_3\A1 на ***D***:***LA_NN\LAB***_3\A2;
- Створіть автономний виконуючий файл командами ***Project / Build/P_ Lab*3_A2 (** рис. 3-5**);**
- Після закінчення компіляції та виконання програми згортаємо основне вікно ***C***++ ***Builder***.

Визначимо для порівняння розмір отриманого автономного виконуючого файлу ***P_ Lab*3_A2 *.exe.*** Для цієї мети запускаємо програму "Провідник " і входимо в папку ***Lab*3** і каталог A1, де зберігається виконуючий файл проекту. В меню "Вид" задаємо режим перегляду "Таблиця". Розмір виконуючого файлу прикладної ***C***++ програми запишіть до протоколу лабораторної роботи № 3.

Крок 2. Створимо виконуючий файл ***P_ Lab*3_A2*.exe*** у складі пакету часу виконання:

- Налаштуйте ***C***++ ***Builder*** на компіляцію проекту в режимі ***Packages***. Виконайте ***Project / Options*** і перейдіть на сторінку (закладку) ***Packages***, де в нижній частині вікна "***Project Options***" установить прапорець індикатора в режим ***"Build with runtime Packages"*** і натисніть ОК;
- Компілюємо проект файлів у вигляді пакету. Виконайте команди ***Project / Build /P_ Lab*3_A_2**;
- Після закінчення компіляції згорніть вікно ***C***++ ***Builder*** для визначення в пакеті часу розміру виконуючого файлу ***P_ Lab*3_A_2*.exe***. Для цієї мети запускаємо програму "Провідник" і входимо в папку ***Lab*3_A_2,** де зберігається виконуючий файл проекту. В меню "Вид" задаємо режим перегляду "Таблиця";

- Розмір виконуючого файлу з пакету до прикладної програми запишіть у протокол лабораторної роботи № 3 та виконуйте наступне:

- визначить список динамічних бібліотек, яки використовує виконуючий файл ***P_ Lab3_A_2.exe***;
- закрийте вікна ***C++ Builder*** ;
- увійдіть в каталог ***Lab3_A_2***, де зберігається файл ***P_ Lab3_A_2.exe***;
- наберіть у командному рядку наступне

tdump P_ Lab3_A_2.exe > dump.txt

і натисніть ***Enter***. У список файлів буде доданий текстовий файл ***dump.txt*** з даними виконуючого файлу з пакету у часі виконання;

- переглянемо текст файлу ***dump.txt*** та перемістимо курсор у розділ ***Imports,*** де буде показуватися список застосованих динамічних бібліотек.

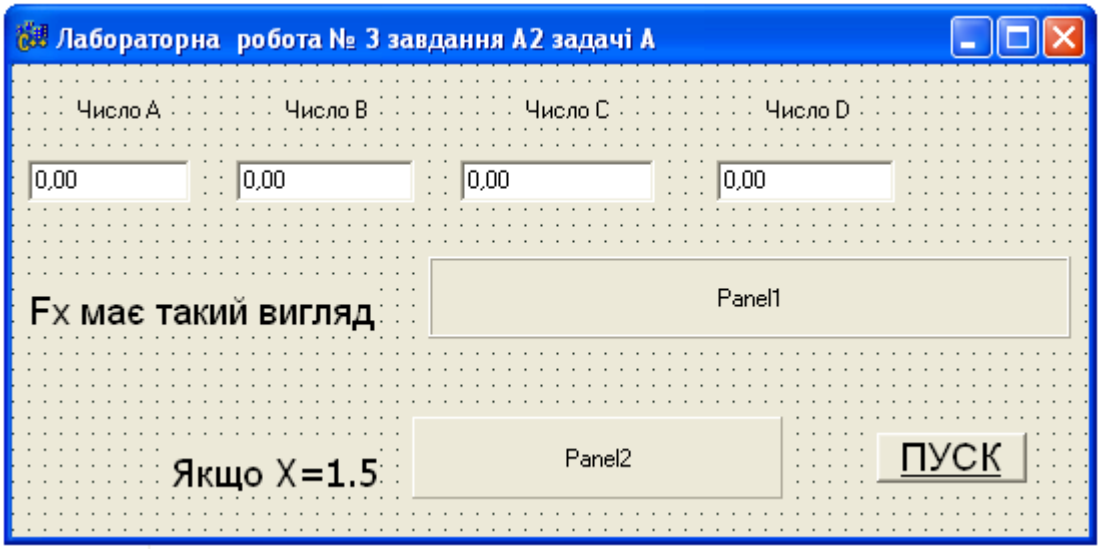


Рис. 3-5. Вигляд форми до завдання А2 задачі А перед компіляцією файлів.

Методика виконання задачі Б.

Порядок дій і команд при виконанні програмування задачі Б по створенню проекту файлів для програми з зображенням до функції F(x):

Крок 1. Установить на форму компоненту ***Label1*** і задайте такі властивості:

- У полі ***Caption*** наберіть такий текст "Обчислення інтеграла по методу";

- Задайте ***Font*** (колір-синій, шрифт-***Tahoma***, розмір 14).

Крок 2. Установить на форму компоненту ***ComboBox*** і виконайте для неї налагодження:

- У властивості ***Text*** видалите напис ***ComboBox*** ;
- У властивості ***Items***(***TStrings***) заповните список названій до числових методів інтегрування для їхнього вибору:

– метод трапецій;

– метод лівих прямокутників;

– метод правих прямокутників.

Для переходу на новий рядок використовуйте клавішу ***Enter*** і збережіть список методів кнопкою ОК.

- Виконайте компіляцію через клавішу ***F*9** або командами ***Run*/*Run***.

Крок 3. Установить на форму компоненту ***Label*2** і задайте такі властивості:

- У полі ***Caption*** наберіть наступний текст "Коефіцієнти до";
- Задайте ***Font*** (колір-синій, шрифт-***Tahoma***, розмір 14).

Крок 4. Установить на форму компоненту ***Label*3** і задайте такий напис

«F(x)»= A + B*X + C*X*X + D*X*X*X »:

- Виконайте через ***F*9** компіляцію і збережіть файли проекту командою "***Save All***".

Крок 5. Установить на форму компоненти ***Label*4, *Label*5, *Label*6** та ***Label*7** і задайте у властивостях ***Caption*** такі відповідні назви: "Число A", "Число B", "Число C", "Число D" з розміром шрифту 10.

Крок 6. Установить на форму програми компоненти ***Edit*1, *Edit*2, *Edit*3** та ***Edit*4** для введення числових значень коефіцієнтів до функції ***F*(*x*)**:

- У властивості ***Text*** до компонент ***Edit*** задайте значення 0,00;
- Виконайте компіляцію і збережіть файли командою "***Save All***".

Крок 7. Установить на форму компоненту ***Label*8** і задайте такі властивості:

- У полі ***Caption*** наберіть наступний текст "Межі інтегрування F(x)";
- Задайте ***Font*** (колір, шрифт розміром 12).

Крок 8. Установить на форму компоненту ***Label*9** і задайте такі властивості:

- У полі ***Caption*** наберіть наступний текст "Різниця ординат Езад" та задайте ***Font***;

Крок 9. Установить на форму компоненту ***Label*10** і задайте напис "Підінтегральна функція":

- Виконаєте компіляцію і збережіть файли командою "***Save All***".

Крок 10. Установите на форму компоненту ***Panel*1** для показу зображення сформованої функції ***F*(*x*)**.

Крок 11. Установить на форму компоненту ***Panel*2** для показу значення до оптимального кроку ***h*** , по якому будуть розраховуватися ординати функції.

Крок 12. Установить на форму компоненту ***Label*12** для показу повідомлення "Інтеграл F(x) = " (дивися рис. 3-6):

- У полі ***Caption*** видалить ім'я ***Label*12**.

Крок 13. Установить на форму компоненту кнопки ***Button*1** і задайте для неї напис "Розрахувати":

- Розташуєте на формі "Лабораторна робота № 3 задача Б " компоненти як показано рис. 3-6.

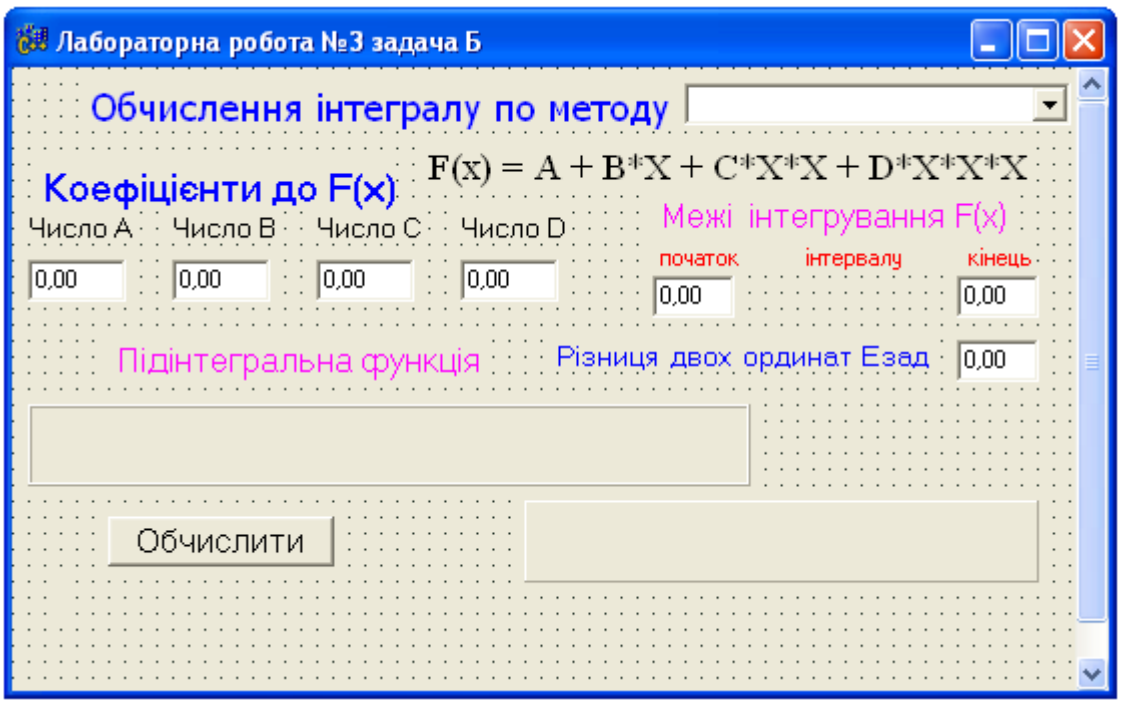


Рис. 3-6. Розташування компонент на формі *C*++ програми до задачі Б.

Крок 14. У шаблон файлу ***U_ Lab3_Б.cpp*** потрібно додати оператори, котрі необхідні для обчислення значення інтеграла в залежності від обраного числового методу та введених значень до коефіцієнтів функціональної залежності ***F(x)***:

```
    /* U_Lab3__Б.cpp */
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "U_ Lab3_Б .h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{  ComboBox1->ItemIndex = 0;
}
//--- Оголошення змінних задачі------
      float h,a,ah,x;
      float S = 0,IL,IR,ITR;
//---------------------------------------------------------------------------


void __fastcall TForm1::Button1Click(TObject *Sender)
{
      Panel1->Caption = "F(x)= " + Edit1->Text + "  " + "+" + "  " + Edit2->Text + " "
+"*" + " " +
"X" + "  " + "+" + "  " + Edit3->Text + " " + "*" + "  " + "X" + " " + "*" +"  " +
"X" + "  " + "+" +"  " + Edit4->Text + " " + "*" + "  " + "X" + " " + "*" +"  " +
"X" + "  " + "*" + "  " + "X";
//------формуємо макрос для F(x)
#define F(x)  (StrToFloat(Edit1->Text)+ StrToFloat(Edit2->Text)*(x)+
StrToFloat(Edit3->Text)*(x)*(x) + StrToFloat(Edit4->Text)*(x)*(x)*(x))
//--визначення оптимального кроку h
   h = (StrToFloat(Edit6->Text) - StrToFloat(Edit5->Text))/2;
   a =  StrToFloat(Edit5->Text);
   ah = a + h;
  M1:   if((F(ah) - F(a)) <= StrToFloat(Edit7->Text))
           goto M2;
        else
        h = h/2;
        ah = a + h;
           goto M1;
 M2:    Panel2->Caption = "Оптимальний шаг h = " + FloatToStr(h);
```

```
//---------------------------------------------------------
 if(ComboBox1->ItemIndex == 1)
{        x = a;   //--- визначення інтегралe по лівим ординатам
   M3:    S = S + F(x);
         if(x == (StrToFloat(Edit6->Text) - h))
               goto M4;
         else
           x += h;
              goto M3;
  M4:   IL = S*h;
   Label12->Caption = "Інтеграл F(x)= " +  FloatToStr(IL);
//-----------------------------------------------------
}
else
   if(ComboBox1->ItemIndex == 2)
  { x = a + h ; //--- визначення інтегралу по методу правих ординат
           S = 0;
   M5:    S = S + F(x);
         if(x == StrToFloat(Edit6->Text))
               goto M6;
         else
           x += h;
              goto M5;
  M6:   IR = S*h;
   Label12->Caption = "Інтеграл F(x)= " +  FloatToStr(IR);
//-----------------------------------------------------------
}
  else
  {
            x = a;  //--- визначення інтегралу по методу трапецій
  M7:    S = S + F(x);
         if(x == StrToFloat(Edit6->Text))
               goto M8;
           else
            x += h;
               goto M7;
  M8:   ITR = S*h - 0.5 * h*(F(a)) - 0.5 * h * StrToFloat(Edit6->Text);
     Label12->Caption = "Інтеграл F(x)= " +  FloatToStr(ITR);
   }
}
//---------------------------------------------------------------------------
```

Результати виконання прикладної ***C***++ програми до задачі Б при обчисленні інтегралу заданим числовим методом дивися по рис. 3-2.

Контрольні запитання до лабораторної роботи № 3

1. Які пакети (***packages***) файлів може формувати ***C++ Builder***.
2. Пояснить, як у ***C++ Builder*** можна одержати файл реалізації модуля (***.cpp***) і об'єктний файл модуля (***.obj***).
3. Які файли з розширенням (***.h***) застосовуються в даних ***C++*** програмах.
4. Які елементи показуються в рядку стану редактора коду ***C++*** програми.
5. Пояснить, як ***C++*** програма обробляє подію для кнопки "***Button1***".

Лабораторна робота № 4

Правила налагодження компоненти MainMenu для створення меню команд прикладної C++ програми

Ціль виконання лабораторної роботи. Кожна прикладна ***C++*** програма має відповідне меню команд, яке можна створювати в ***C++ Builder*** за допомогою компоненти ***MainMenu*** і тому задачі даної роботи складаються у вивченні:

- властивостей компоненти ***MainMenu*** та техніки використання для побудування меню команд і залежних підкоманд;
- інструментів, методів підключення значків та програмних функцій до відповідних команд з меню команд ***C++*** програми;
- функцій та операторів для вбудовування заданих графічних зображень на форму та значків до команд при розробці меню команд прикладної ***C++*** програми.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Компонента-конструктор MainMenu для створення в прикладних C++ програмах основного меню команд та меню залежних підкоманд

Практично будь-яки прикладні ***C++*** програми повинні мати відповідне меню команд, оскільки саме меню команд дає найбільш зручний інтерфейс для виконання команд і роботи користувача з програмою. Також у меню команд можна бачити перелік припустимих виконуваних дій користувача в

прикладній ***C***++ програмі. Інтерфейс команд в ***C***++ програмах для ***Windows*** реалізується такими видами меню:

- головне меню команд зі списком команд на формі при виконанні прикладної програми;

- каскадні меню, у яких до розділу первинного списку команд меню ставиться у відповідність список підрозділів, набору команд нижнього рівня;

- спливаючі або контекстні меню, що з'являються, якщо користувач клацне правою кнопкою мишки на якомусь елементі програми.

Основна вимога до меню це їхня стандартизація. Ця вимога відноситься до багатьох аспектів меню: місце розміщення заголовків меню і їхніх розділів; форми самих заголовків меню; позначення графічного логотипу команди; вказівка на швидкі клавіші та ін.. Основна мета стандартизації меню це полегшити користувачу роботу з прикладною ***C***++ програмою. Треба щоб меню читко визначало для користувача його дії і не було потрібно додатково уточнювати, де шукати команди. Також стандартизація розташування меню команд на полі форми виробляє в користувача упевнену роботу з прикладною програмою.

Для розробки меню команд прикладної програми в ***C***++ ***Builder*** маються таки компоненти: ***MainMenu, ActionList, ImageList*** та ін.. Зручно розробляти меню команд за допомогою диспетчеризації дій і подій при допомозі компоненти ***ActionList***. Компонента ***ActionList,*** не додаючи ніяких принципових можливостей, дозволяє систематизувати і упорядкувати розробку об'єктно-орієнтованих прикладних програм. Також застосування ***ActionList*** дозволяє заощаджувати час на проектування меню команд. Ця компонента містіть список дій програми, що передбачаються командами в меню. Дія - це деяка реакція програми на команду (вплив) користувача, наприклад, щиглик на кнопці або на напису команди в меню. При проектуванні програми необхідно скласти список тих дій, що прийдеться виконувати користувачу в процесі роботі з програмою. Первинне складання списку дій майбутньої програми виконується через

проектовану на форму невізуальну компоненту ***ActionList***. Після щиглику на значку даної компоненти на формі відкривається редактор дій, який дозволяє додавати дії та їх упорядковувати (рис. 4-1).

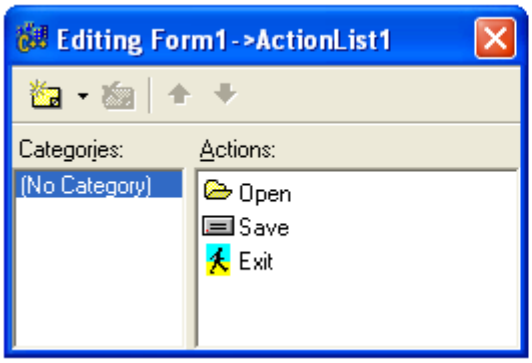


Рис. 4-1.

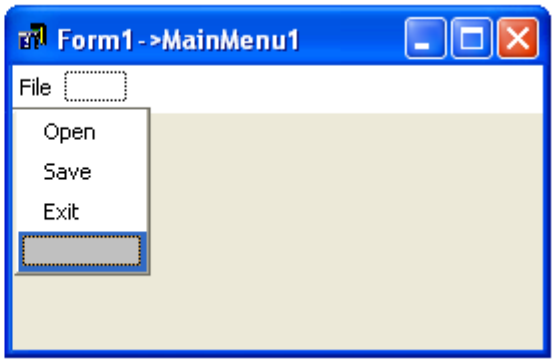


Рис. 4-2.

Звичайно перед назвою кожної команди в меню показується значок, який допомагає користувачу краще розрізняти і сприймати команду. Для формування набору значків відповідним командам необхідно застосовувати компоненту ***ImageList***, яка вибирається на сторінці ***Win*32**. Ця компонента дозволяє організувати ефективне і ощадливе керування багатьма піктограмами майбутніх команд ***C***++ програм для ***Windows***. Після установки ***ImageList*** на форму потрібно подвійним щигликом мишки на значку компоненти активізувати вікно для компонування значків.

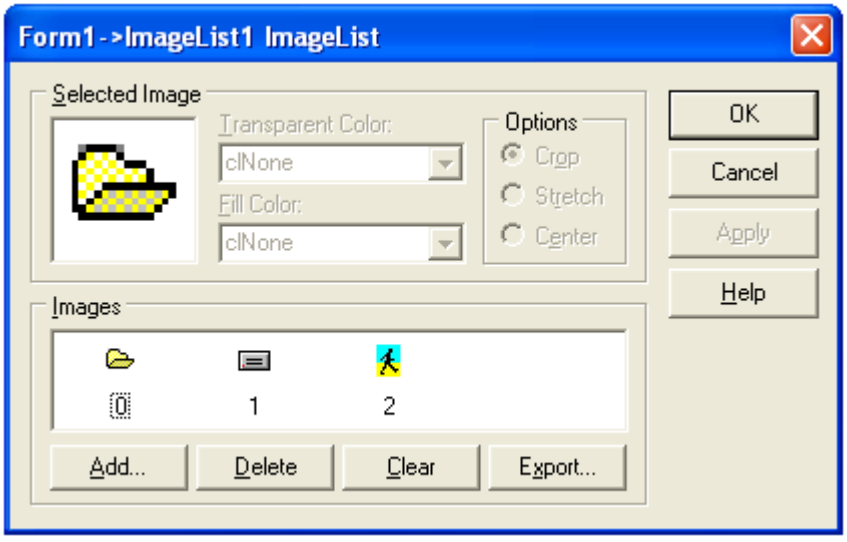


Рис. 4-3.

Файли зі значками вибираються за допомогою кнопки ***Add*** і можна вставляти з файлу на форму кілька значків, а зайві віддаляються кнопкою ***Delete*** (рис. 4-3). При додаванні в список значків автоматично задається цифровий індекс для прив'язки до назви дії (команди меню).

Формування меню команд і налагодження властивостей для дій програми виконуються за допомогою компоненти ***MainMenu***, обираємой на сторінці ***Standart*** . Компонента ***MainMenu*** (рис. 4-2) невізуальна і після її установки на форму потрібно подвійним щигликом мишки на значку активізувати конструктор меню команд. Поле для заповнення найменування команди показується автоматично пунктирною курсорною рамкою.

Більшість програм для ***Windows*** містять малюнок - логотип (вбудоване графічне зображення). При розробці програми можна рисунок на форму вставити і закріпити за допомогою компоненти ***Image***, яка вибирається на сторінці ***Additional***. Після подвійного щиглика на формі в полі ***Image*** відкривається діалогове вікно для вибору кнопкою ***Load*** необхідного файлу з рисунком. Якщо файл обраний, то в цьому випадку автоматично рисунок фіксується на формі і його місце розташування визначається переміщенням маркерної рамки компоненти ***Image.***

Загальні зведення з компонент для вбудовування рисунків і значків у прикладну C++ програму

Більшість ***C++*** програм для ***Windows*** містять рисунок або логотип (вбудоване графічне зображення). Для відображення графічної інформації можна вибрати в ***VCL*** базові шість компонент та крім цього також можна графічні рисунки відображати на інших компонентах ***C++ Builder***, у яких мається властивість ***Canvas***. Для розташування рисунка на формі прикладної ***C++*** програми зручно використовувати компоненту ***Image***. Компонента ***Image*** вибирається у бібліотеці ***VCL*** на сторінці ***Additional***. Після подвійного щиглика мишкою у полі ***Image*** на формі відкривається діалогове вікно для вибору кнопкою ***Load*** необхідного файлу з рисунком. Якщо файл обраний, то

в цьому випадку автоматично рисунок фіксується на формі і його місце розташування визначається координатами компоненти ***Image*** та переміщеннями маркерної рамки. Після завантаження рисунка з файлу в компоненту ***Image*** цей рисунок не тільки відображається компонентою, але і зберігається в прикладній програмі.

Для компоненти ***Image C++ Builder*** підтримує три типи файлів - бітові матриці, піктограмки та мета файли. Усі ці три типи файлів зберігають рисунок і їх відмінність полягає лише в способі їхнього збереження зображення у середині файлів та в засобах доступу до них. Бітова матриця, це рисунок у файлі з розширенням ***.bmp***, який відображає колір кожного пікселя у зображенні рисунка. При цьому інформація зберігається таким чином, що будь-який комп'ютер може відобразити зображення рисунка з кількістю кольорів, відповідно до його конфігурації.

Піктограмки (файли з розширенням ***.ico***) - бітові матриці іконок у вигляді значків. Вони повсюди використовуються для позначення значків програм, у швидких кнопках, у пунктах меню команд та у різних списках. Спосіб збереження зображення у піктограмках схожий зі збереженням інформації в бітових матрицях, але маються і особливості, зокрема, неможливо піктограмку масштабувати, бо вона зберігає той розмір, у якому була створена.

Метафайли (***Metafiles***) зберігають не послідовність біт зображення, а інформацію про спосіб створення рисунка. Файли зберігають послідовності команд рисування, яки можуть бути повтореними при відтворенні зображення. Це робить такі файли більш компактними, ніж бітові матриці.

Компонента ***Image*** основну властивість ***Picture*** активізує у вікні інспектора об'єкта або на формі подвійним щигликом мишки на значку компоненти ***Image***. Ця компонента має і інший набір властивостей. Якщо установити властивість ***AutoSize*** у значення ***true***, то розмір компоненти ***Image*** буде автоматично налаштовуватися під розмір завантаженого в неї графічного зображення. Якщо ж властивість ***AutoSize*** установити в значення

false, то зображення рисунку може не поміститися в компоненту, або площа компоненти буде на багато більше площі зображення рисунка.

Властивість - ***Stretch*** дозволяє підганяти не компоненту під розмір рисунка, а рисунок під розмір компоненти. Не завжди реально установити розміри ***Image*** точно пропорційними розміру рисунка, що викликає деформацію зображення. Властивість ***Stretch*** має сенс встановлювати в ***true*** для візерунків, а не для рисунків і картинок. Властивість ***Stretch*** не діє на зображення піктограмок, тому що вони не можуть змінювати своїх розмірів.

Установка в ***true*** властивості - ***Center***, центрує зображення на полі ***Image***, якщо розмір компоненти більше розміру рисунка.

Прозорість рисунка визначається властивістю - ***Transparent***. Якщо ***Transparent*** дорівнює ***true***, то зображення тільки бітових матриць стає прозорими.

У властивості ***Picture*** мається підвластивість, яка вказує на графічний об'єкт, що зберігається. Якщо в ***Picture*** зберігається бітова матриця, то на неї указує властивість ***Picture.Bitmap***, на піктограмку - ***Picture.Icon*** і відповідно на метафайл - ***Picture.Metafile***. Інтегроване середовище C++ Builder має свій вбудований редактор зображень - ***Image Editor***, що викликається командою ***Tools/Image Editor***. Цей простий редактор може редагувати зображення у вигляді бітових матриц, піктограм, зображень курсорів та зберігати створені зображення у вигляді файлів, а також відразу включати їх у файл ресурсів прикладної програми ***Resource File*** (***.res***), а також і у файл ресурсів компоненти ***Component Resource File*** (***.dcr***). Редактор зображень ***Image Editor*** може створити файл бітової матриці (***.bmp***), файли піктограмок (***.ico***) і файл з зображенням курсору (***.cur***).

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Постановка задачі по програмуванню у лабораторній роботі

Необхідно створити прикладні ***C***++ програми для роботи у ***Windows*** і для цього виконати наступні завдання:

Завдання №1 (задача А). Створить ***C***++ програму з вбудованим рисунком на формі в компоненту ***Image***.

Завдання №2 (задача Б). Потрібно створити ***C***++ програму і за допомогою компоненти ***MainMenu*** сконструювати таке меню команд:

– ***Open*** (можна буде переглядати файли з рисунками);

– ***Save As...*** *(*можна буде зберегти рисунок у файл із заданим ім'ям);

– ***Exit*** *(*можна вийти і закрити програму).

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Завдання для аналізу і обробки результатів роботи запрограмованої C++ програми

Після запуску в роботу ***C***++ програми з лабораторної роботи № 4 для захисту результатів програмування необхідно виконати таке:

- нарисувати блок-схему алгоритму по виконанню ***C***++ програми до завдання № 1(задача А) з лабораторної роботи № 4;
- по результатам виконання завдання № 2 (задача Б) з лабораторної роботи № 4 нарисувати блок-схему алгоритму до методики проектування меню команд в ***C***++ програмі за допомогою компоненти ***MainMenu*** та невізуальних компонент ***C***++ ***Builder***.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

***Порядок дій і команд при виконанні програмування задачі завдання № 1(задача А) по створенню C++ програми з вбудованим рисунком в компоненту Image*1

Крок 1. Перейдіть на форму нової програми і у властивості ***Caption*** напишіть назву роботи "Лабораторна робота № 4 задача А", щоб цей текст з'явився у заголовку форми:

- Виконаєте команду ***File/Save Project As...*** . На диску ***D:***\ створіть папку ***Lab_*4** і в цій папці сформуйте внутрішню папку ***Lab_*4_1** для файлів проекту ***P_work_*4_1.*bpr*** та файлу модуля форми ***U_work_*4_1.*cpp*.**

Крок 2. В бібліотеці компонент на сторінці ***Additional*** виберіть і розмістить на форму програми компоненту ***Image*1** та потім маркерною рамкою

визначить розмір під рисунок.

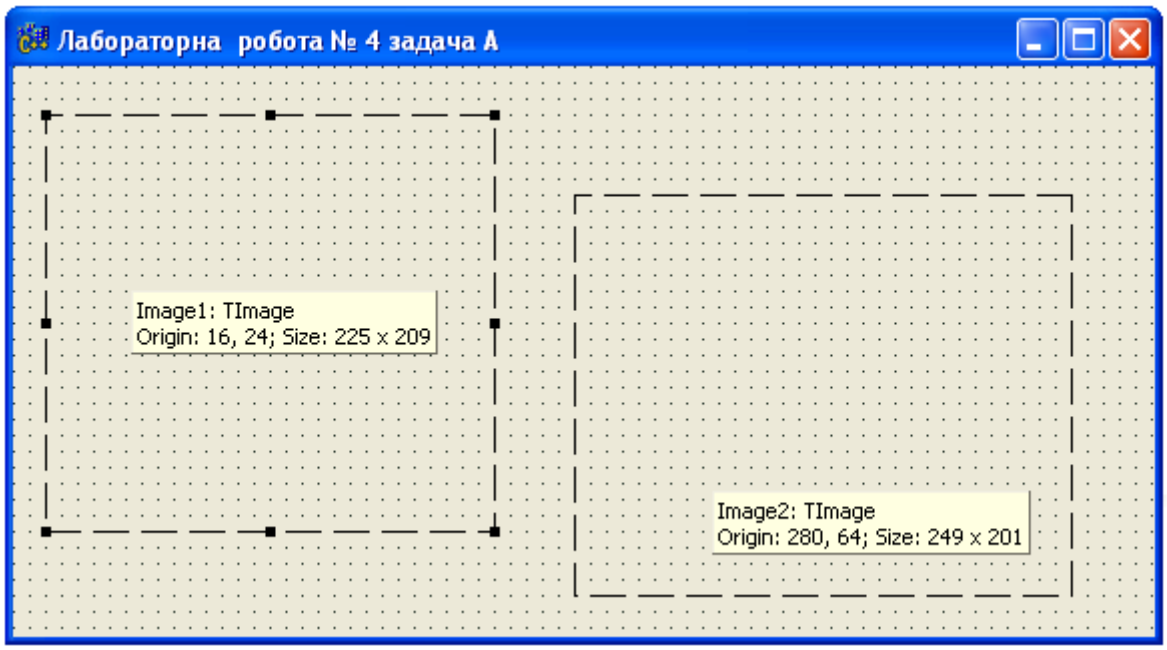


Рис. 4-4.

Крок 3. Потрібно відкрити вікно діалогу для вибору файлу з рисунком:

- Для цього у вікні інспектора об'єктів у властивості ***Picture*** відкрийте кнопку з крапками, щоб активізувалося вікно "***Picture Editor***", потім через кнопку ***Load*** виберіть файл із рисунком ***athena.bmp*** з папок ***Program Files\Common Files\Borland Shared\Images\Splash\ 16Color***.

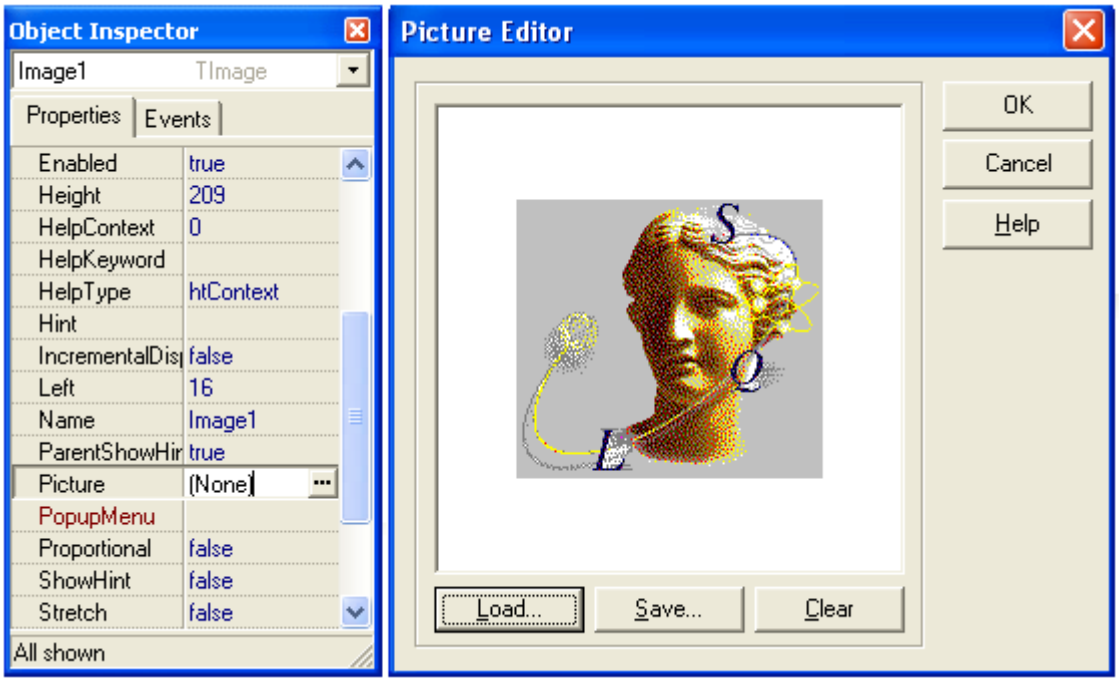


Рис. 4-5.

Крок 4. Виконайте команду ***Run*** і перегляньте результат роботи програми.

Крок 5. У бібліотеці компонентів ***VCL*** на сторінці ***Additional*** виберіть і помістить на форму компоненту ***Image2*** та маркерною рамкою визначить розмір полю для другого рисунку.

Крок 6. Для ***Image2*** потрібно відкрити вікно діалогу для вибору файлу з рисунком:

- Для цього у вікні інспектора об'єкта у властивості ***Picture*** відкрийте кнопку з крапками, щоб активізувалося вікно "***Picture Editor***", потім через кнопку ***Load*** виберіть файл ***earth.bmp*** рисунка із папок ***Program Files\Common Files\Borland Shared\Images\Splash\ 16Color\ athena.b*** .

Крок 7. У вікні інспектора об'єктів у властивостях:

– ***Stretch*** змінить режим ***false*** на ***true***;

– ***Transparent*** змінить режим ***false*** на ***true***;

– ***AutoSize*** змінить режим ***false*** на ***true***;

– ***Center*** змінить режим ***false*** на ***true***.

Крок 8. Виконайте команду ***Run*** і перегляньте результат роботи програми.

Рис. 4-6.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

<u>Порядок дій і команд</u> при виконанні програмування завдання № 2 (задача Б) по створенню С++ програми з меню команд на основі компоненти MainMenu

<u>Крок 1.</u> Перейдіть на форму програми і у властивості ***Caption*** напишіть назву роботи "Лабораторна робота № 4 задача Б", щоб цей текст з'явився у заголовку форми. Виконаєте команду ***File/Save Project As...*** . На диску ***D***:\ у папці ***Lab_4*** створіть внутрішню папку ***Lab_4_2*** для файлів проекту ***P_work_4_2.bpr*** та файлу модуля форми ***U_work_4_2.cpp***.

<u>Крок 2.</u> В бібліотеці ***VCL*** на сторінці ***Standart*** виберіть компоненту диспетчер дій ***ActionList.***

<u>Крок 3.</u> В бібліотеці компонент на сторінці ***Win*32** виберіть ***ImageList*** для завдання значків командам в меню команд програми.

<u>Крок 4.</u> По значку ***ImageList*1** виконаєте подвійного щиглика мишкою. У вікні, що відкриється, через кнопку ***Add...*** з папок ***Program Files\Common Files\ Borland Shared\ Images\Buttons*** , відповідно до рис 4-1 та рис. 4-2, виберіть такі файли: ***fldropen.bmp, harddisk.bmp, picture.bmp*** як значків до меню ***File*** з наступними командами:

Open

Save

Exit

<u>Крок 5.</u> В бібліотеці компонент на сторінці ***Dialogs*** виберіть і установить на форму невізуальні компоненти ***OpenPictureDialog*** та ***SavePictureDialog*** .

<u>Крок 6.</u> Далі необхідно зв'язати диспетчер дій зі списком призначених значків для команд і тому виконайте наступне:

- Активізуйте маркерну рамку на значку ***ActionList***;
- У вікні інспектора об'єктів у полі ***Image*** вкажіть ***ImageList*1**.

<u>Крок 7.</u> На значку ***ActionList*** у полі форми подвійним щигликом активізуйте редактор дій для меню команд:

- Клавішею ***Insert*** у полі ***Action*** додається елемент ***Action1*** і у властивостях на полі ***Name*** змінюємо назву на ***Open***;
- У властивостях ***Caption*** та ***Hint*** укажіть назву ***Open***. У властивості ***Imageindex*** зі списку вкажіть номер значка до команди ***Open***;
- Через кнопку ***New Action(Ins)*** додаємо в список назву ***Action*****1** і у властивостях на полі ***Name*** змінюємо назву на ***Save***;
- У властивостях ***Caption*** та ***Hint*** укажіть напис ***Save***. У властивості ***Imageindex*** зі списку вкажіть номер значка для команди ***Save***;
- Через кнопку ***New Action(Ins)*** додаємо в список назву ***Action*****1** і у властивостях на поле ***Name*** змінюємо назву на ***Exit..*** .У властивостях ***Caption*** та ***Hint*** укажіть напис ***Exit***. У властивості ***Imageindex*** зі списку вкажіть номер значка для команди ***Exit***;
- Перевіряємо свої результати в ***ActionList*** по рис. 4-1.

Крок 8. В бібліотеці компонент на сторінці ***Standad*** виберіть компоненту-конструктор ***MainMenu*** та установить на форму програми.

Крок 9. Задайте для властивості ***Image*** значення ***ImageList*****1**.

Крок 10. На формі програми по значку ***MainMenu*** виконуємо подвійний щиглик. Відкриється вікно з курсорною рамкою:

- У властивості ***Caption*** набирайте назву меню ***File*** і потім натисніть клавішу ***Enter*** ;
- У властивості ***Name*** з'явиться назва ***File*****1**, а в курсорній рамці вікна з'явиться напис ***File*** і будуть додані дві курсорні рамки для наступних заповнень.

Крок 11. У проектованому меню команд виділить нижній курсор і потім у властивості ***Caption*** наберіть ***Open*** та натисніть клавішу ***Enter***. Перевірте записи у властивостях: ***Imageindex, Name, Action***.

Крок 12. У проектованому меню команд нижній курсор виділить і потім у властивості ***Caption*** наберіть ***Save*** і натисніть клавішу ***Enter***. Перевірте записи у властивостях: ***Imageindex, Name, Action***.

Крок 13. У проектованому меню команд нижній курсор виділить і у властивості ***Caption*** наберіть ***Exit*** і натисніть клавішу ***Enter***. Перевірте записи у властивостях: ***Imageindex, Name, Action***. Конструктор (редактор) компоненти ***MainMenu*** закрийте і на формі з'явиться напис ***File,*** на який установить покажчик мишки, щоб з'явився список команд зі значками, як це видно на рис. 4-7.

Шаг 14. Перейдіть у вікно редактора кодів і перегляньте, що вже мається в модулі файлу ***U_work_4_2.cpp*** .

Крок 15. На формі програми по значку ***ActionList*1** виконаєте подвійного щиглика мишкою для активізації на екрані списку дій. Далі потрібно визначити події для записаних дій:

- На назві ***Open*** зробіть подвійного щиглика мишкою і на закладці ***Events***, у полі ***OnExecute*** автоматично заповниться подія ***OpenExecute*** і одночасно в модулі форми буде вставлений шаблон коду;
- У шаблон коду ***OpenExecute*** необхідно додати наступні оператори:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::OpenExecute(TObject *Sender)
{
if(OpenPictureDialog1->Execute() )
{
Image1->Picture->LoadFromFile(OpenPictureDialog1->FileName);
Form1->Caption ="Перегляд рисунка" + OpenPictureDialog1->FileName;
}
}
//---------------------------------------------------------------------------
```

- Виконайте команду ***Run*** і перегляньте стан меню команд в програмі;
- На назві ***Save*** зробіть подвійного щиглика мишкою і на закладці ***Events***, у полі ***OnExecute*** автоматично заповниться подія ***SaveExecute*** і одночасно в модулі форми буде вставлений шаблон коду;
- У шаблоні коду ***SaveExecute*** необхідно додати наступні оператори:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::SaveExecute(TObject *Sender)
{
 if(SavePictureDialog1->Execute() )
{ Image1->Picture->SaveToFile( SavePictureDialog1->FileName);
```

```
}
}
//---------------------------------------------------------------------------
```

- Виконайте команду ***Run*** і перегляньте стан меню команд в програмі;
- На назві ***Exit*** зробіть подвійного щиглика мишкою і на закладці ***Events,*** у полі ***OnExecute*** автоматично заповниться подія ***ExitExecute*** і одночасно в модулі форми буде вставлений шаблон коду;
- У шаблоні коду ***ExitExecute*** необхідно додати наступні оператори:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::ExitExecute(TObject *Sender)
{
 Form1->Close();
}
//---------------------------------------------------------------------------
```

- Виконаєте команду ***Run*** і переглянете стан меню команд в програмі.

Крок 16. В бібліотеці ***VCL*** виберіть сторінку ***Additional*** і розмістить на форму компоненту ***Image*****1**, потім маркерною рамкою визначить розмір під рисунок.

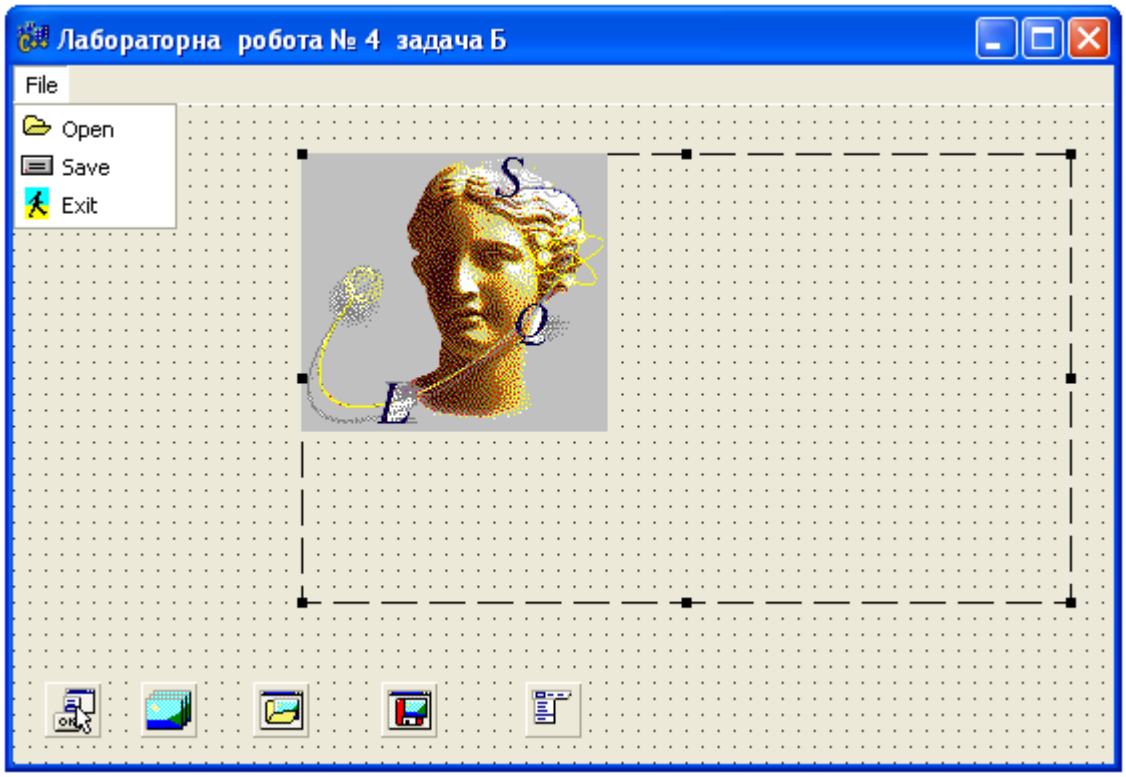


Рис. 4-7.

Крок 17. Виконайте команду ***Run*** для ***C***++ програми з формою вікна і меню команд (рис. 4-7) та перевірте в меню команд виконання команд, сформованих компонентою ***MainMenu***. Файли до рисунків вибирайте з папок по такому шляху ***Program Files\Common Files\Borland Shared\Images\ Splash\16Color\.....***.

Контрольні запитання до лабораторної роботи № 4

1. Яким чином застосовується компонента ***MainMenu*** при розробці меню команд для ***C***++ програми.
2. Покажіть порядок дій, якщо додається або редагується команда в меню команд програми.
3. Покажіть порядок дій, якщо додаються або редагуються рисунки значків у меню команд програми.
4. Необхідно продемонструвати для яких цілей застосовується компонента ***ImageList***.
5. Пояснить та покажіть призначення компоненти ***ActionList***.
6. Пояснить техніку вбудовування малюнків на форму програми.

Лабораторна робота № 5

Техніка використання та контролю з обробки даних структурного типу в С++ програмі

Ціль виконання лабораторної роботи. В ***C***++ програмах обробляється і використовується багато даних структурного типу (***struct***) і тому основна задача в даній лабораторній роботі складається у вивченні:

- техніки і методики використання в прикладній ***C***++ програмі даних типу «структура»;
- правил та команд для контролю обробки даних структурного типу за допомогою застосування програмного відладчіка ***C***++ ***Builder*** ;

- техніки пошуку помилок програмування в текстах файлів, створюваних до програмних модулів, прикладної ***C***++ програми;
- техніки одночасної роботи редактора коду та програмного відладчіка при пошуках помилок програмування в операторах нестандартних функцій, яки використовуються прикладною ***C***++ програмою.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Структурні типи даних в С++ програмах

Структури - це складений тип даних, створюваний програмістом, що будується з використанням інших типів даних. Структури являють собою об'єднаний загальним ім'ям набір даних різних типів. Саме цим відрізняються структури і що в них можуть зберігається дані різних типів, наприклад, у масивах зберігаються дані одного типу.

Дані структури називаються елементами або полями структури. Формат оголошення структури має такий вигляд

```
struct <ярлик_структури> {
  тип_даних перемінна1;
  тип_даних перемінна2;
  ……..;
  }<ім'я_перемінної>;
```

Ярлик структури дозволяє повідомляти нові перемінні структурного типу з зазначеним ярликом. В оголошенні структури обов'язково повинен бути заданий "ярлик_структури" або "ім'я_перемінної". Наприклад, можна оголосити таку структуру

```
struct point {
  float x;
  float y;
  float z; };
```

або структуру в такому вигляді

```
struct  {
  float x;
```

```
    float y;
    float z; } P1;
```

В мові програмування ***C***++ структури наділені додатковими можливостями:

- перше – це можна як елемент структури вказувати визначення функції

```
struct point {
    float x;
    float y;
    float z;
    show()
{ printf(”/n %f”, x); }
};
```

- друге – можна вказувати в структурі специфікатори доступу до даних в елементах (полях) і функціям-елементів, як це робиться в класах.

Дозволяється застосовувати специфікатори ***public*** (відкритий) та ***private*** (закритий). Закриті елементи структури можуть бути доступні тільки для функцій-елементів цієї структури. Ні через об'єкт, ні через покажчик на об'єкт доступ до них неможливий.

Відкриті елементи структури можуть бути доступні для будь-яких функцій у ***C***++ програмі. Наприклад, у наступному оголошенні структури

```
struct MyStr{
    int x,y;
  int Get( );
   private:
    int a,b;
  void F( );
                    };
```

дані ***x*** та ***y*** і функція ***Get( )*** - відкриті і можуть використовуватися при роботі зі структурою, а дані ***a*** та ***b*** і функція ***F( )*** - закриті і ними може

користуватися тільки функція ***Get( ).*** Структура може мати елементи з побітовими полями, т. е. елемент може займати задану кількість біт

```
struct pole_bit {
unsignit char x :  3;
unsignit char y :  5;
};
```

Постановка задачі по програмуванню у лабораторній роботі

Необхідно створити за допомогою ***Consol Wizard*** файл прикладної програми з виконанням таких дій:

- в структуру даних задаються (вводяться с клавіатури) значення координат для трьох меток (крапок) на площині;
- по значенням до координат, записаних в структуру даних, визначаються масштабні коефіцієнти і з урахуванням їх значень будується поле площини для розміщення меток (крапок);
- по значенням до координат, записаних в структуру даних, розміщуються метки (крапки) на зелену площину з урахуванням визначених відповідних масштабних коефіцієнтів;
- за допомогою програмного відладчіка ***C++ Builder*** виконується по шагах перевірка правильності виконання операторів в функціях програми;
- за допомогою програмного відладчіка ***C++ Builder*** виконується контроль значень розрахованих масштабних коефіцієнтів при розміщенні меток (крапок) на зелену площину.

Завдання для аналізу і обробки результатів роботи запрограмованої C++ програми

Після запуску в роботу ***C++*** програми лабораторної роботи № 5 для захисту результатів програмування необхідно виконати таке:

- нарисувати блок-схему алгоритму по виконанню ***C++*** програми лабораторної роботи № 5;

- нарисувати блок-схему алгоритму до команди вікна ***Watch List*** (відладчіка) по визначенню і по замінам значень масштабних коефіцієнтів меток (крапок) на площині.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

***<u>Порядок дій і команд</u> при виконанні програмування задачі*:**

<u>Крок 1.</u> На диску ***D***:\ створіть папку ***Lab_*5.**

<u>Крок 2.</u> Після запуску у роботу ***C++ Builder*** у меню ***File*** виконайте команду ***New***. В результаті відкриється вікно ***New Items***, де виберіть ***Consol Wizard***.

<u>Крок 3.</u> У вікні "Редактор Коду" наберіть наступний текст програми і виконайте компіляцію:

```
 /*****************************************************/
//     Трикутник на площині, заданий своїми
//     вершинами P1(x1,y1), P2(x2,y2), P3(x3,y3)
/*****************************************************/
//---------------------------------------------------------------------------

#pragma hdrstop

//---------------------------------------------------------------------------
#include <stdio.h>
#include <math.h>
#include <conio.h>
#include <alloc.h>
#define Xmax 52     /* 52 */
#define Xmin 5      /* 1 */
#define Ymax 21     /* 25 */
#define Ymin 3      /* 1 */
 struct point {
    float x,y;
        } k[2] ;
 /* функція завдання координат вершин трикутника*/
 point * scan_xy( void )
 {
  gotoxy(44,1);
 printf("Введіть координати вершин трикутника:");
    gotoxy(62,2);
 printf(" вершина P1(x1,y1) ");
    gotoxy(66,3);
 printf(" x1=");    scanf("%f",&k[0].x);
    gotoxy(66,4);
 printf(" y1=");    scanf("%f",&k[0].y);
    gotoxy(62,5);
```

```
 printf(" вершина P2(x2,y2) ");
    gotoxy(66,6);
 printf(" x2=");    scanf("%f",&k[1].x);
    gotoxy(66,7);
 printf(" y2=");    scanf("%f",&k[1].y);
    gotoxy(62,8);
 printf(" вершина P3(x3,y3) ");
    gotoxy(66,9);
 printf(" x3=");    scanf("%f",&k[2].x);
    gotoxy(66,10);
 printf(" y3=");    scanf("%f",&k[2].y);
    return k;
  }


/* функція обчислення коефіцієнтів для */
/* масштабування координат рисунка */
 float * m_ko( struct point k[] )
  {
    float xma,xmi,ymi,yma;
    static  float * mk;
       int i;
   xma = k[0].x;   xmi = k[0].x;
   yma = k[0].y;   ymi = k[0].y;
   for(i=0; i<=2; i++)
    {
     if( k[i].x > xma)
          xma = k[i].x;
     if( k[i].x < xmi)
          xmi = k[i].x;
    }
   for( i=0; i<=2; i++)
    {
     if( k[i].y > yma)
          yma = k[i].y;
     if( k[i].y < ymi)
          ymi = k[i].y;
    }
     mk = ( float *) calloc( 2, sizeof( float ));
          * mk= yma;
          *( mk + 1 ) = xma;
      return mk;
   }

/* функція розрахунку масштабованих координат*/
/* вершин трикутника та їхній показ на рисунку*/
 void out_point( struct point k[], float mk[])
 {    int i;
```

```
    double x,y;
    float s;
    for( i=0; i<=2; i++)
     {
       y = Ymax - ((k[i].y * (Ymax - Ymin))/mk[0]);
       x = Xmin + ((k[i].x * (Xmax - Xmin))/mk[1]);
 gotoxy( floor(x), floor(y));  textcolor(MAGENTA);
 textbackground(WHITE); textcolor(BLACK );
 cprintf("o"); gotoxy( floor(x) + 1, floor(y));
 textmode(LASTMODE); textbackground(WHITE);
 textcolor(BLACK);
 cprintf("P%d\a",i + 1);
 textmode(LASTMODE);
s = k[0].x *(k[1].y-k[2].y) + k[1].x *(k[2].y-k[0].y);
s = ( s + k[2].x *(k[0].y-k[1].y))/2;
  gotoxy(40,23);
printf("Площа трикутника S=%f",fabs(s));
  textbackground(WHITE);
  gotoxy(1,23);  textcolor(BLACK);
printf("  Лабораторна"); gotoxy(1,24);
printf("   робота N5");
  textbackground(WHITE);
  gotoxy(50,24); textcolor(BLACK);
printf(" Натиснути будь-яку клавішу");
  textmode(LASTMODE);   gotoxy(3,22);
  printf("0.00\t|\t|\t|\t|\t%f ---> X    ",mk[1]);
  gotoxy(1,2);
  printf("%f",mk[0]);
  gotoxy(2,3); printf("Y ^");
  gotoxy(4,4); printf("|");
  gotoxy(4,5); printf("|");
  gotoxy(4,6); printf("-");
  gotoxy(4,10); printf("-");
  gotoxy(4,15); printf("-");
  gotoxy(4,20); printf("-");
      gotoxy(78,24);
    }
   }

/*функція фарбування полю під рисунок*/
  void pole( void)
    {
      int x,y;
         for( y = Ymin; y <= Ymax; y++ )
         {   for( x = Xmin; x <= Xmax; x++ )
            {
              gotoxy( x,y ); textbackground(GREEN   );
              cprintf("    ");
```

```
            }
          }
      }

//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
 clrscr();
         scan_xy();
       pole();
    out_point( k, m_ko(k));
       getch();
        return 0;
}
//---------------------------------------------------------------------------
```

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Види команд C++ Builder для налагодження та пошуку помилок у текстах операторів програмних модулів прикладної C++ програми

Компіляція файлів ***C***++ програми може виконуватися декількома способами.

Варіант перший. Компіляція з наступним виконанням програми здійснюється командою ***Run/Run***, чи відповідною швидкою кнопкою, чи "гарячою" клавішею ***F9***. У цьому випадку виконується компіляція ***C***++ програми, її компонування, створюється виконуючий модуль (***.exe***) і він запускається на виконання. Створення модуля (***.exe***) і виконання буде проводитись тільки у випадку, якщо при компіляції і компонуванні не виявлені помилки. У процесі компіляції і компонування на екрані активізується вікно стану процесу компіляції. Верхній рядок вікна показує ім'я проекту файлів, який компілюється. У наступному рядку відображається поточна обробка, тобто компіляція визначеного модуля компонування (***Linking***). В рядку нижче відображається поточний рядок модуля (***Current line***), оброблюваний компілятором, і загальне число рядків у модулі (***Total lines***). У самому нижньому рядку відображається число зауважень(***Hints***) виявлених на даний момент, попереджень (***Warnings***) і помилок (***Errors***). Клавіша ***Cancel*** унизу вікна дозволяє перервати процес компіляції і

компонування. Якщо при компіляції у файлі зустрілися непоправні помилки, виконуючий файл не буде створений. Якщо помилок немає, файл створюється, але і в цьому випадку у компілятора можуть бути попередження і зауваження, які необхідно уважно вивчити. При компіляції проекту файлів, що складається з декількох модулів, компілюються тільки ті модулі, тексти яких були змінені з моменту попереднього компонування проекту. Це істотно заощаджує час компіляції.

Варіант другій. При виконанні команди ***Run*** можна задати командний рядок, якщо в програмі передбачена передача якихось параметрів. Для цього треба виконати команду ***Run/Parameters*** і у вікні, що відкрилося, написати необхідний командний рядок.

Не завжди треба компілювати проект і відразу виконувати. Часто важливіше просто перевірити, чи не містять останні зміни коду якихось помилок. У цьому випадку не треба гаяти час на виконання проекту і краще скористатися іншими командами меню: ***Project/Compile Unit***, ***Project/Make Project*** або ***Project/Build Project***.

Варіант третій. Команда ***Compile*** виконує компіляцію тільки того модуля, що виділений у вікні "Редактор коду" або у вікні "Менеджері Проектів". Ця команда дозволяє найбільш швидко перевірити наявність помилок і зауважень при компіляції модуля, тому що не здійснюється компонування програми і не компілються ніякі інші модулі. Якщо компіляція пройшла успішно, створюється об'єктний файл (***.obj***) відкомпільованого модуля.

Варіант четвертий. Команда ***Make*** виконує компіляцію усіх тих модулів, тексти яких були змінені з моменту попередньої компонування проекту. Якщо компіляція пройшла успішно, то створюються об'єктні файли модулів (***.obj)*** і здійснюється компонування програми. Якщо і вона пройшла, то створюється виконуваний модуль (***.exe)***. У такий спосіб відмінність ***Make*** від ***Run*** тільки в тім, що після компонування частин програма не виконується.

Команда ***Build*** подібна команді ***Make*** за одним виключенням - компілюються всі модулі, незалежно від того, коли вони востаннє змінювалися. Виконання цієї команди вимагає найбільшого часу.

Крім описаних команд компіляції мається ще дві: ***Project/Make All Projects*** та ***Project/Build ALL Projects***. Вони подібні розглянутим командам ***Make*** і **Build**, але використовуються при роботі з групою проектів і відносяться не до одного, а до всіх проектів групи.

Якщо в процесі компіляції виявлена помилка, то це активізує вікно з повідомленнями про знайдені помилки компіляції. Звичайне вікно з повідомленнями відкривається в нижній частині вікна редактора кодів. Вікно повідомлень є вікном, що вбудовується, і його можна витягнути з вікна редактора і зробити самостійним чи вікном умонтувати, наприклад, у вікно "Інспектор Об'єктів ". Найбільш зручно вбудовувати вікно з повідомленнями про помилки у вікно "Редактор Коду". Якщо при маніпуляціях вбудовування вікно "повідомлень" загублено, то потрібно у вікні "Редактор Коду " клацнути правої клавіші миші і виконати команду ***Message View***. Помилка компіляції також викликає появу виділення кольором рядка модуля, що компілюється.

В інтегрованому програмувальному середовищі ***C++ Builder*** мається набір спеціальних програмних засобів для налагодження (пошук помилок) програмних модулів прикладної ***C++*** програми:

– **Майстер оцінки виразів (*Tooltip Expression Evaluation*).** Цей майстер дозволяє підвести курсор миші до ідентифікатора перемінної і на екрані з'являється числове значення цієї перемінної. Таким способом можна довідатися про значення перемінних ***C++*** програми в даний момент. Деякі перемінні не вдається спостерігати, тому що компілятор який оптимізує код, видалив їх з результуючого коду. Цей інструмент дозволяє контролювати тільки окремі перемінні;

– **Вікно спостережень (*Watch List*).** Цей влаштований програмний інструмент забезпечує можливість бачити значення декількох перемінних

відразу, щоб їх можна було порівняти і зрозуміти причину помилки коду. Для відкриття вікна ***Watch List*** потрібно виконати команди ***View/Debug Windows/Watches***. Також досить підвести курсор миші до перемінної та натиснути ***Ctrl+F*5**. При цьому вікно спостереження автоматично відкриється і у ньому з'явиться ім'я перемінної і її значення (значення змінної буде видно тільки при зупинці виконання програми і переході в ***C++ Builder***). Потім можна підвести курсор до іншої перемінної, знову натиснути ***Ctr+F*5** і у вікні спостережень з'явиться новий рядок. Більш того можна виділити курсором якесь вираження, натиснути ***Ctr+F*5** і у вікні спостереження побачити значення цього вираження. Іноді перемінні не вдається спостерігати, тому що компілятор з оптимізацією видалив їх з результуючого коду і помістив відповідні значення в системні регістри. Це прискорює виконання обчислень у програмі, заощаджує пам'ять, але перешкоджає спостереженню перемінних у процесі налагодження. У таких випадках можна оголосити відповідну перемінну з ключовим словом ***volatile***. Наприклад: ***volatile int x*1 .**

Специфікатор ***volatile*** показує компілятору, що дану перемінну не можна зберігати в регістрі. У цьому випадку інший кращий варіант досягти тієї ж мети, якщо виконати команду ***Project/Options***, і у вікні, що відкрилося, на сторінці ***Advanced Compiler*** виключити опцію ***Register Variables***. Обов'язково після налагодження обоє ці варіанти потрібно зняти, щоб не знижувати ефективність роботи прикладної програми. Якщо перейти у вікно спостережень і там клацнути правою клавішею миші та у меню вибрати такі команди: зокрема ***Edit Watch*** (відредагувати вираз, що спостерігається,) чи ***AddWatch*** (уставити новий вираз, що спостерігається,). В обох випадках активізується вікно ***Watch Properties*** , що також можна відкрити через ***Ctrl+F*5**. В полі редагування ***Expression*** можна записати ім'я будь-який змінний чи будь-яке вираз, що містить змінні, константи, функцію. Поле редагування ***Repeat count*** використовується при спостереженні масивів і дозволяє задати число елементів масиву, що спостерігаються. Наприклад, якщо мається в програмі масив ***X*[ ],** то можна просто вказати в полі

Expression ім'я масиву ***X***. Тоді у вікні спостережень будуть відображатися всі елементи масиву ***X***. Але можна вказати в полі ***Expression*** ім'я елемента ***X*[0] ,** а в полі ***Repeat count*** написати , наприклад, **5**. Тоді у вікні спостережень будуть відображатися тільки перші **5** елементів масиву. Поле редагування ***Digits*** визначає число виведених значущих розрядів чисел після коми, що плаває. Індикатор ***Enabled*** дозволяє відключити висновок у вікно спостережень відповідного вираження під час виконання програми. Це підвищує продуктивність процесу виконання. Після того, як програма зупинена і потрібно все-таки подивитися дані вираження у вікні спостереження, то виділить його у вікні і зробіть на ньому подвійний щиглик. Відкриється вікно ***Watch Properties*** із завантаженим у нього вираженням і потім потрібно включити індикатор ***Enabled*** і клацнути ОК.

Індикатор ***Allow Side Effects*** дозволяє або забороняє відображення таких виразів, що здатні викликати побічні ефекти. Наприклад, можна записати в поле ***Expression*** вираз **++A**. Якщо індикатор ***Allow Side Effects*** виключений (він виключений за замовчуванням), то у вікні спостережень поруч з виразом **++A** покажеться текст:"***Side effects are not allowed***" (побічні ефекти заборонені). Радіокнопки в нижній частині вікна ***Watch Properties*** задають формат висновку значення змінної чи виразу. За замовчуванням включена кнопка ***Default***. У цьому випадку формат визначається автоматично по типу відображуваного виразу. Але можна вибрати і інший формат. Наприклад, можна скористатися цими радіокнопками, щоб відображати деяку цілу змінну один раз у десятковому вигляді, а іншого разу в шістьнадцатирічному вигляді. Список, що випадає, у полі редагування ***Expression*** дозволяє вибрати вираження з тих, котрі використовувалися раніше і при необхідності його відредагувати. Наприклад, якщо потрібно вивести значення

Form*1->*Label*1->*Caption*, *Form*2->*Label*2->*Caption*, *Form*3->*Label*3->*Caption

то досить один раз написати цей вираз, а надалі брати його зі списку, що випадає, і тільки змінювати в ньому цифру.

– **Вікно оцінки і модифікації (*Evaluate/Modify*).** Це вікно дозволяє в процесі налагодження не тільки спостерігати, але і змінювати значення змінних. Зробити це вікно видимим можна командою ***Run/Evaluate/Modify***. Відповідно команду ***Debug/ Evaluate/Modify*** можна також вибрати з контекстного меню, що спливає при щиглику правою клавішею миші у вікні "Редактор Коду".

У верхнім полі ***Expression*** задається ім'я перемінної чи вираження. Після цього потрібно клацнути кнопку ***Evaluate*** і в полі ***Result*** з'явиться поточне значення. Якщо в полі ***Expression*** задане ім'я перемінної, то стає доступної кнопка ***Modify*** що дозволяє змінити значення, тобто можна втрутитися в процес виконання програми шляхом завдання нового значення в поле ***New value*** і потім необхідно натиснути кнопку ***Modify***. У результаті значення перемінної в додатку зміниться, що можна бачити в полі ***Result***. Також зміниться значення у вікні спостережень - ***Watch List*** якщо в нього перейти.

– **Трасування програми по кроках.** Вище були розглянуті вікна, що дозволяють бачити статичну інформацію в момент, коли відбулася помилка або зупинено виконання програми. Але не завжди така інформація дає повну картину виконання операторів в програмі. Часто, щоб знайти причину помилки в програмі, треба виконати по кроках якийсь фрагмент (шматочок) програми, спостерігаючи за змінами перемінних при виконанні кожної команди.

Команди для проходу фрагмента програми по кроках. Таблиця 5-1.

КОМАНДА	"Гарячі" клавіші	ДІЯ КОМАНДИ
Step Over (По кроках без заходу в….)	**F8**	Покрокове виконання рядків програми, вважаючи виклик функції за один рядок, тобто вхід у функцію не виконується.
Trace Into (Трасування з заходом у...)	**F7**	Покрокове виконання рядків програми з заходом у викликувані функції.
Trace to Next Source Line (Трасування до наступного рядка)	**Shift+F7**	Перехід до наступного рядку програми, що виконується.
Run toCursor (Виконати до курсору)	**F4**	Виконується програма до того місця (виконуваного оператора), де розташовано курсор у вікні редактора коду.

Run Until Return (Виконати до виходу з функції)	**Shift+F8**	Виконання програми до виходу з викликаної функції з зупинкою на операторі, що випливає за викликом функції.
Show Execution Point (Показати крапку виконання)	-	Міститься курсор на оператор, що буде виконуватися наступним.

При трасуванні фрагмента програми зміни значень спостерігаються у вікні ***Watches***. Для зручної роботи потрібно вмонтувати вікно ***Watches*** у вікно Інспектора Об'єктів, його активізувати щигликом на корінці закладки і задати список перемінних для спостереження.

– **Крапки переривання (*breakpoint*).** Могутнім інструментом налагодження програми є розміщення крапок переривань у програмі. Щоб ввести просту (безумовну) крапку переривання, досить у вікні "Редактор коду" клацнути мишею на смужці зліва від коду необхідного рядка програми. Рядок програми офарблюється в червоний колір і на смужці ліворуч з'явиться червона крапка. Перевага крапок переривання полягає в тім, що їх можна одночасно вказати безліч у різних місцях програми і у різних модулях. Програма виконується доти, поки керування не перейде до першої крапки переривання. Для того щоб забрати крапку переривання, досить клацнути мишею на червоній крапці зліва від коду, відзначеного крапкою переривання. Крапки переривання можна встановлювати тільки на виконуваних операторах. Якщо, наприклад, установити переривання на рядку оголошення перемінної, то в момент запуску додатка в червоній крапці з'явиться хрестик. Так ***C++ Builder*** попереджає, що переривання не буде, оператор не виконуваний. Аналогічний хрестик з'являється і у тому випадку, якщо компілятор у процесі оптимізації коду забрав його з коду і розмістив у регістрах.

Debuger надає можливість виконати умовне переривання. Затримайте курсор над червоною крапкою ліворуч у рядку, де задане переривання. У результаті спливе ярличок з характеристикою даної крапки переривання. За замовчуванням ніяких умов для зупинки не задається.

Наприклад, потрібно задати крапку переривання і визначити умову, зупинка повинна відбуватися на 10 циклі виконання оператора циклу. Для цього встановлюється переривання і потім на червоній крапці ліворуч виконується щиглик правою клавішею миші і у меню яке з'являється вибираємо команду ***Breakpoint properties***. Відкривається вікно властивостей переривання, де у верхніх рядках ***Filename***(ім'я файлу) і ***Line Number***(номер його рядка) заповнилися автоматично в момент завдання крапки переривання. Поле ***Condition***(умова) дозволяє ввести де яки необхідні умови. Переривання буде відбуватися тільки в тому випадку, якщо значення умови стане дорівнювати ***true*** . Наприклад, можна вказати умову зупинки в такому вигляді ***i* = = 10.**

<u>Контрольні запитання до лабораторної роботи № 5</u>

1. Пояснить, у яких задачах потрібні змінні типу "структура".
2. Покажіть можливі варіанти оголошення структури даних.
3. Пояснить правила запису і читання даних у полях змінної структурного типу і покажчика на структуру.
4. Яка структура даних використовуються в ***C***++ програмі лабораторної роботи.
5. Покажіть варіанти використання крапок **(*breakpoint*)** для ***C***++ програми**.**

Лабораторна робота № 6

Техніка використання функцій при обробках команд мишки до подій в прикладних C++ програмах

<u>Ціль виконання лабораторної роботи</u>. Інтегроване програмувальне середовище ***C***++ ***Builder*** має відповідний набір функцій для обробки команд маніпулятора «МИШКА» до подій яки виникають в прикладних програмах і тому задача даної роботи складається у вивченні:

- методики завдання та управління функціями з обробки команд мишки і подій при виконанні прикладної ***C***++ програми;

- техніки роботи з рисунками на формі та правил використання влаштованого редактора зображень ***Image Editor*** для створення рисунків курсорів нестандартного вигляду та їх підключення у файл ресурсів в проекті файлів прикладної програми;
- команд по управлінню замінами зображень курсору мишки при зміщеннях по полю вікна прикладної програм.

Загальні зведення о функціях для обробки команд мишки і подій в прикладних С++ програмах

У більшості прикладних програм при їх виконанні у ***Windows*** команди і події визначаються при допомозі маніпулятора (мишки). Для обробки команд мишки та подій в програмах ***C++ Builder*** має набір функцій для обробки визначених користувачем команд до подій при роботі прикладної програми.

Функції для обробки команд мишки та їх призначення. Таблиця 6.1

Позначення функцій з обробки команд мишки	**Опис призначення функції при обробках команд мишки до подій**
OnClick	Щиглик мишки на компоненті та деякі інші дії користувача при роботі з програмою
OnDblClick	Два щиглика мишки на компоненті.
OnMouseDown	Натискання клавіші мишки над компонентою з можливістю визначення нажатої кнопки і координат курсору мишки.
OnMouseMove	Зміщення курсору мишки над компонентою з можливістю визначення нажатої кнопки і координат курсору мишки.
OnMouseUp	Відпускання раніше натиснутої кнопки мишки над компонентою з можливістю визначення натиснутої кнопки і координат курсору мишки
OnStartDrag	.Початок процесу переміщення об’єкту з можливістю визначення об’єкту.
OnDragOver	Переміщення об’єкту над компонентою з можливістю визначення об’єкта та координат курсору мишки.
OnDragDrop	Відпускання раніше натиснутої кнопки мишки після переміщення об’єкту з можливістю визначення об’єкту та координат курсору мишки.
OnEndDrag	Ще одна подія при відпусканні раніше натиснутої кнопки мишки після переміщення об’єкту з можливістю визначення об’єкту та координат курсору мишки.

OnEnter	Подія у момент отримання елементом фокуса в результаті маніпуляції мишкою(натискання клавіші табуляції або програмної передачі фокуса).
OnExit	Подія у момент загублення фокусу в результаті маніпуляцій мишкою (натискання клавіші мишки, табуляції або програмної передачі фокуса).
OnMouseWheel	Подія при обертанні колеса мишки в будь якому напрямку.
OnMouseWheelUp	Обертання колеса мишки в напрямку у верх, якщо подія не оброблена OnMouseWheel.
OnMouseWheel Down	Обертання колеса мишки в напрямку у низ, якщо подія не оброблена OnMouseWheel.

В прикладних програмах найбільше використовується для команд мишки подія ***OnClick*** . Така подія виникає, коли користувач зробив щиглика на компоненті, тобто була натиснута і потім відпущена клавіша мишки, якщо курсор був розташований на компоненті. Така подія виникає також і при інших діях користувача програми. Подія для мишки виникає і обробляється, якщо:

- ✓ Користувачем обрано елемент: на сітці форми; у дереві; у списку; в меню команд або натиснув кнопку зі стрілкою;
- ✓ Користувач натиснув клавішу пробіл, коли кнопка або індикатор були у фокусі програми;
- ✓ Користувач натиснув клавішу ***Enter***, або активна форма має по замовчанню кнопку з властивістю ***Default.***
- ✓ Користувач натиснув клавішу ***Esc*** , або активна форма має по замовчанню кнопку з властивістю ***Cancel***.
- ✓ Користувач натиснув клавішу швидкого доступу до кнопки або індикатору. Наприклад, якщо властивість ***Caption*** індикатора записано як «& - напівжирний» і символ ‘П’ підчеркнуто , тоді натискання користувачем клавіш ***Alt*** + **П** викликає виконання події ***OnClick*** в цьому індикаторі;
- ✓ Програма встановила в ***true*** властивість ***Checked*** радіокнопки ***RadioButton***;
- ✓ Програма виконала заміну у індикатора властивості ***Checked;***
- ✓ Виконано метод ***Click*** для елемента в меню команд;

✓ Для форми подія **OnClick** виникає, коли користувач зробив щиглика на пустому місці форми або на недоступній компоненті.

При проектуванні ***C***++ програми визначається послідовність табуляції віконних компонент. Під цим треба розуміти послідовність, у який перемикався фокус з компоненти на іншу компоненту та коли користувач натискав клавішу ***Tab***. Властивість форми ***ActiveControl*** в процесі проектування, визначає, яка з розміщених компонент буде у фокусі в перший момент при виконанні програми. В процесі виконання програми ця властивість змінюється і показує ту компоненту, у який в даний момент знаходиться фокус.

Послідовність табуляції задається для компонент властивістю ***TabOrder***. Початкова послідовність табуляції визначається просто тією послідовністю, в який розміщувались керуючи елементи на формі. У першого елемента на формі значення ***TabOrder***, дорівнює **0**, другому відповідно **1** і так далі. Значення ***TabOrder***, нуль, визначає, що при перший появі форми на екрані у фокусі буде перша компонента, яка була встановлена на форму (якщо для форми не встановлено властивість ***ActiveControl***).

Всі ці функції по обробках подій, яки пов'язані з маніпуляціями мишки передається параметр ***Sender*** типу ***Tobject*** . Цей параметр є указником на компоненту до якої виникла подія. Параметр **Sender** можна використовувати для роз позначення причини події. Наприклад, записати такий оператор

```
if (Sender = = Image1)
……;
```

Окрім параметру ***Sender*** в функції з обробки подій ***OnMouseDown*** та ***OnMouseUp*** передаються параметри, яки дозволяють розпізнати натиснуту кнопку мишки, разом з додатковими клавішами, а також визначити координати курсору мишки. Заголовок функції з обробки події ***OnMouseDown*** може мати, наприклад, наступний вигляд:

```
void_fastcall TForm1:: Image1MouseDown(Tobject * Sender, TMouseButton
Button, TShiftState Shift, int X, int Y)
```

Додатково до параметру ***Sender*** в функцію з обробки події передаються параметри ***Button***, ***Shift***, ***X*** та ***Y***. Параметр ***Button*** у момент події визначає натиснуту кнопку мишки. Тип ***TmouseButton*** –визначається наступним чином :

```
enum TmouseButton{ mbLeft, mbRight, mbMiddle};
```

Значення ***mbLeft*** відповідає натисканню лівої кнопки мишки, значення ***mbRight*** – правої кнопки мишки, а значення ***mbMiddle*** – середньої кнопки мишки. Наприклад, якщо ви бажаєте щоби функція обробних події реагувала на натискання лівої кнопки мишки, тоді можна записати такий оператор:

```
if (Button ! = mbLeft) return;
```

і в даному випадку , якщо значення ***Button*** не дорівнює ***mbLeft*** , тобто була натиснута не ліва кнопка мишки, тоді обробка події не виконується.

Параметр ***Shif*** типа ***TshiftState*** визначає, яки додаткові клавіші на клавіатурі були натиснути у момент події – натискання кнопки мишки.

В усіх подіях, пов’язаних з командами мишки також передаються до курсору координати **X** и **Y.** Дані параметри визначають координати курсору в клієнтській площини компоненти. Це дає можливість визначати різну реакцію програми в залежності від розташування курсору мишки на формі.

Редактор зображень ***Image Editor***. Інтегроване програмувальне середовище C++ Builder має влаштований редактор зображень ***Image Editor***, який викликається командою ***Tools/ Image Editor*** . Даний редактор дозволяє створювати зображення у вигляді бітових матріц, піктограмок та зображень до курсорів і зберігати одночасно у файл і у файл ресурсів програми. Цим відрізняється редактор ***Image Editor*** від других більших графічних редакторів.

Робота в редакторі ***Image Editor*** починається з меню ***File***, в якому можливо обрати розділ ***Open*** – відкрити новий файл зображення чи файл ресурсів або розділ ***New*** – створити новий файл. Якщо обрано команду ***New,*** тоді пропонується обрати вид файлу, який потрібно створювати:

Resource File (***.res***)	Файл ресурсів
Component Resource File (***.dcr***)	Файл ресурсів компоненти
Bitmap File (***.bmp***)	Файл бітової матриці
Icon File (***.ico***)	Файл піктограми
Cursor File (***.cur***)	Файл зображення нового курсору

В прикладній ***C***++ програмі, розробленої в ***C***++ ***Builder***, автоматично створюється глобальний об'єкт ***Screen*** (экран) типу ***TScreen***, властивості якого визначаються з інформації ***Windows*** про дисплей комп'ютера на якому виконується прикладна програма. Об'єкт ***Screen*** має властивість ***Cursor***, яка визначає вид курсору над компонентою. Якщо властивість ***Cursor*** має значення ***crDefault*** , тоді вид курсору при переміщеннях над компонентами визначається встановленими в них властивостями ***Cursor*** . Якщо властивість ***Cursor*** об'єкту ***Screen*** відрізняється від значення ***crDefault*** , тоді відповідні властивості компоненти відміняються і курсор буде мати глобальний вид, котрий задано в ***Screen***. Це можна використовувати для зміни виду курсору на інший вид, наприклад, на зображення «писчані часи» при виконанні довгих розрахунків або інших довгих операцій. Для цього потрібно записати наступне:

```
Screen->Cursor  =  crHourGlass;
try
{ //--- выполняются длинные операции
  ……
catch (…..)
{
   Screen->Cursor = crDefault; //--восстановление курсора
    throw

}
   Screen->Cursor = crDefault;
```

При успішному або аварійному закінченні довгих операцій зображення курсору в будь якому випадку стане відповідати значенню ***crDefault.*** Також має місце властивість ***Cursors[i]*** яка представляє собою список доступних у ***C***++ програмі курсорів. Можна створити і використовувати також свій нестандартний вид курсору. Створюється такий нестандартний курсор і включається у ресурс програми за допомогою редактора ***Image Editor***.

Реєстрація нового нестандартного курсору виконується за допомогою функції ***LoadCursor***. Наприклад, якщо створено новий курсор ***NEWCURSOR*** (ім'я обов'язково записувати великими буквами) тоді необхідно створити і глобальну константу яка буде вказувати на нестандартний новий курсор. Це виконується таким чином:

```
Const crMyCursor = 11;
```

В такому випадку необхідно в обробник подій форми ***OnCreate*** записати оператор для реєстрації нестандартного курсору у властивість ***Cursors***:

```
Screen->Cursor = crMyCursor;
```

При необхідності можна відновлювати значення стандартного глобального курсору таким оператором:

```
Screen->Cursor = crDefault;
```

Новий зареєстрований курсор можна також використовувати як локальний, наприклад, тільки до панелі ***Panel*1** і для цього потрібно записати такий оператор:

```
Panel1->Cursor = crMyCursor;
```

При створюванні нестандартного курсору потрібно в основному меню команд редактора ***Image Editor*** обрати команду ***Cursor*** та обов'язково для ***Set Hot Spot*** потрібно в пікселях задати значення точки указника курсору. Точка указника курсору – це та точка зображення нестандартного курсору координати якої (***X*** і ***Y***) передаються функції з обробки подій мишки.

Постановка задачі по програмуванню у лабораторній роботі

Необхідно створити для ***Windows*** прикладну ***C***++ програму де у вікні форми повинно бути:

- ✓ Три нестандартні вимкнуті кнопки (***Button*_123_*Off***) відповідно до зображення рис. 6-1;

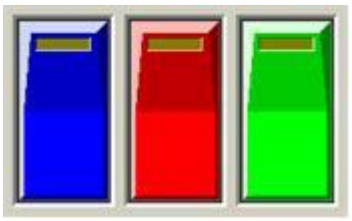
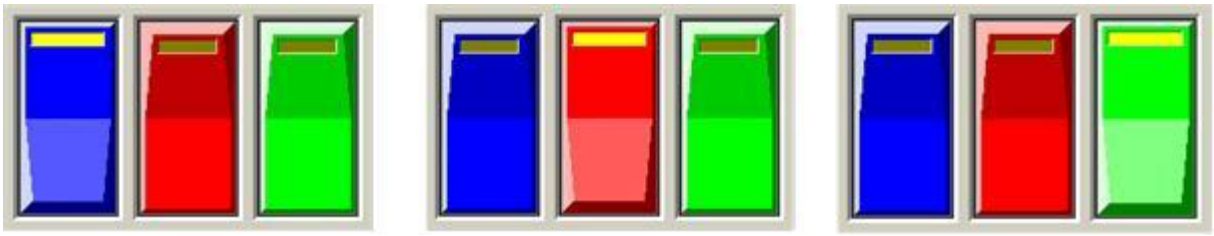


Рис. 6-1. Нестандартні кнопки у вимкнутому стані.

- ✓ При натисканні лівой клавіши мишки над зображенням нестандартної першої кнопки (лівої) зображення ціх кнопок повинно змінюватись на зображення (***Button_1_On***) відповідно до рис. 6-2;



Button_1_On ***Button_2_On*** ***Button_3_On***

Рис. 6-2. Нестандартні кнопки з різним станом включення.

- ✓ При натисканні лівой клавіши мишки над зображенням нестандартної другої кнопки (середнеї) зображення ціх кнопок повинно змінюватись на зображення (***Button_2_On***) відповідно до рис. 6-2;
- ✓ При натисканні лівой клавіши мишки над зображенням нестандартної третьої кнопки (правої) зображення ціх кнопок повинно змінюватись на зображення (***Button_3_On***) відповідно до рис. 6-2;

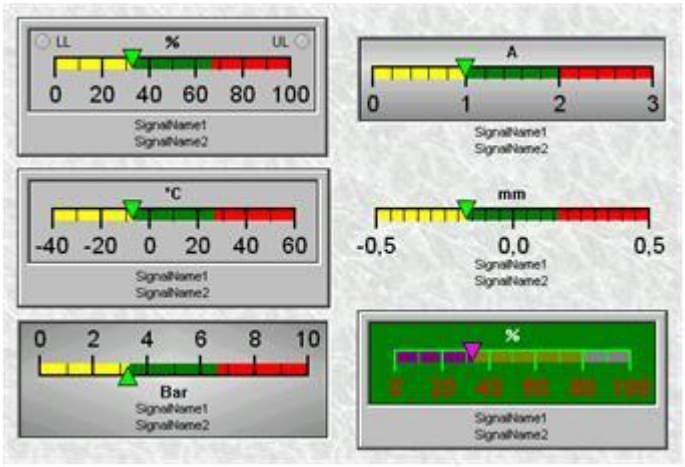


Рис.6-3.

✓ Після натискання клавіші мишки над однією з нестандартних кнопок форми ***C***++ програма повинна одночасно з перемиканням зображень стану нестандартних кнопок також в полі ***Image*1** переключати зображення приладів, яки показані на рис. 6-3 та 6-4;

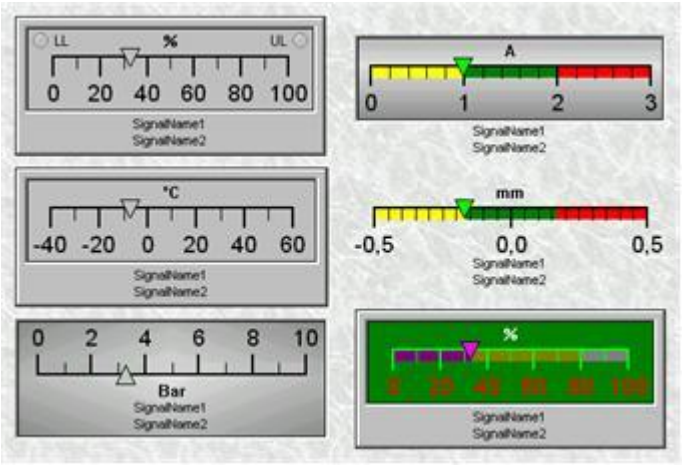


Рис. 6-4.

✓ Необхідно поля компонент ***Image*1, *Image*2** та ***Image*3** на формі вікна програми розташувати і визначити розміри відповідно до рис. 6-5;

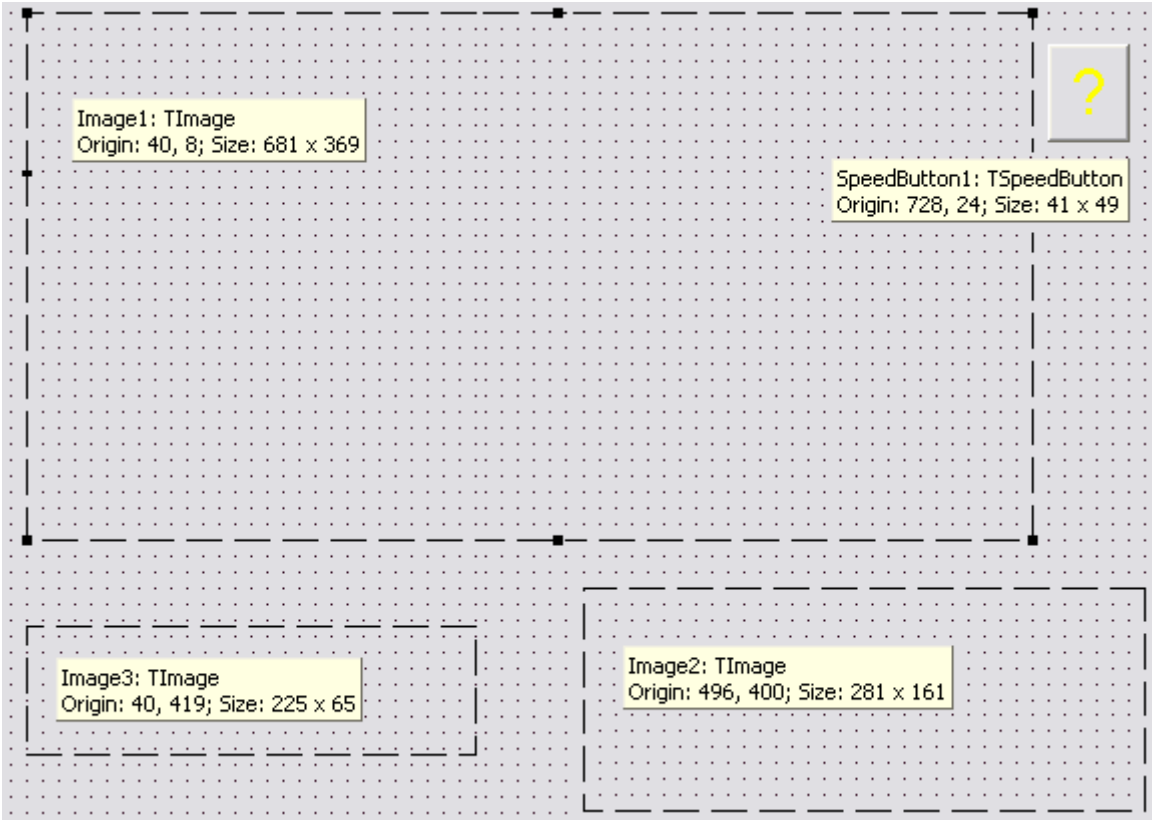


Рис. 6-5. Розміри до компоннет ***Image*1, *Image*2** та ***Image*3** на формі вікна.

- ✓ При розташуванні указника мишки над нестандартними кнопками зображення стандартного курсору «стрілка» повинно замінюватися на інший курсор у вигляді «рука»;
- ✓ Необхідно в редакторі зображень ***Image Editor*** створити нестандартний курсор і підключити у файл ресурсів прикладної ***C***++ програми для обробки команд мишки до подій;
- ✓ При переміщеннях нестандартного курсора по зображенням рисунків приладів на рис. 6-3 та рис. 6-4 повинні з'являться у полі ***Image*3** повідомлення про назву приладу, який обрано мишкою.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Завдання для аналізу і обробки результатів роботи запрограмованої С++ програми

Після запуску в роботу С++ програми лабораторної роботи № 6 для захисту результатів програмування необхідно виконати таке:

- нарисувати блок-схему алгоритму з роботи програми лабораторної роботи № 6;
- нарисувати блок-схему алгоритму виконання команд при змінах зображень курсора мишки на екрані дисплея комп'ютера;
- нарисувати алгоритм оброки команд мишки у вікні С++ програми (рис. 6-6) при зміщеннях курсору по зображенням приладів рисунка (6-3).

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Порядок дій і команд для виконання програмування задачі:

Крок 1. Перейдіть на форму програми і у властивості ***Caption*** напишіть назву лабораторної роботи "Лабораторна робота № 6. Техніка обробки команд мишки", щоби цей текст з'явився в заголовку вікна форми програми перед збереженням файлів проекту ***C***++ програми:

- ➢ Для цього виконайте команду ***File/Save Project As…*** . На диску D:\ створіть папку ***Lab*_6** і в цій папці створіть внутрішню папку ***Lab*_6_1** для файлів проекту ***P_work*_6** та файлу ***U_work*_6.*cpp*** до модуля форми програми.

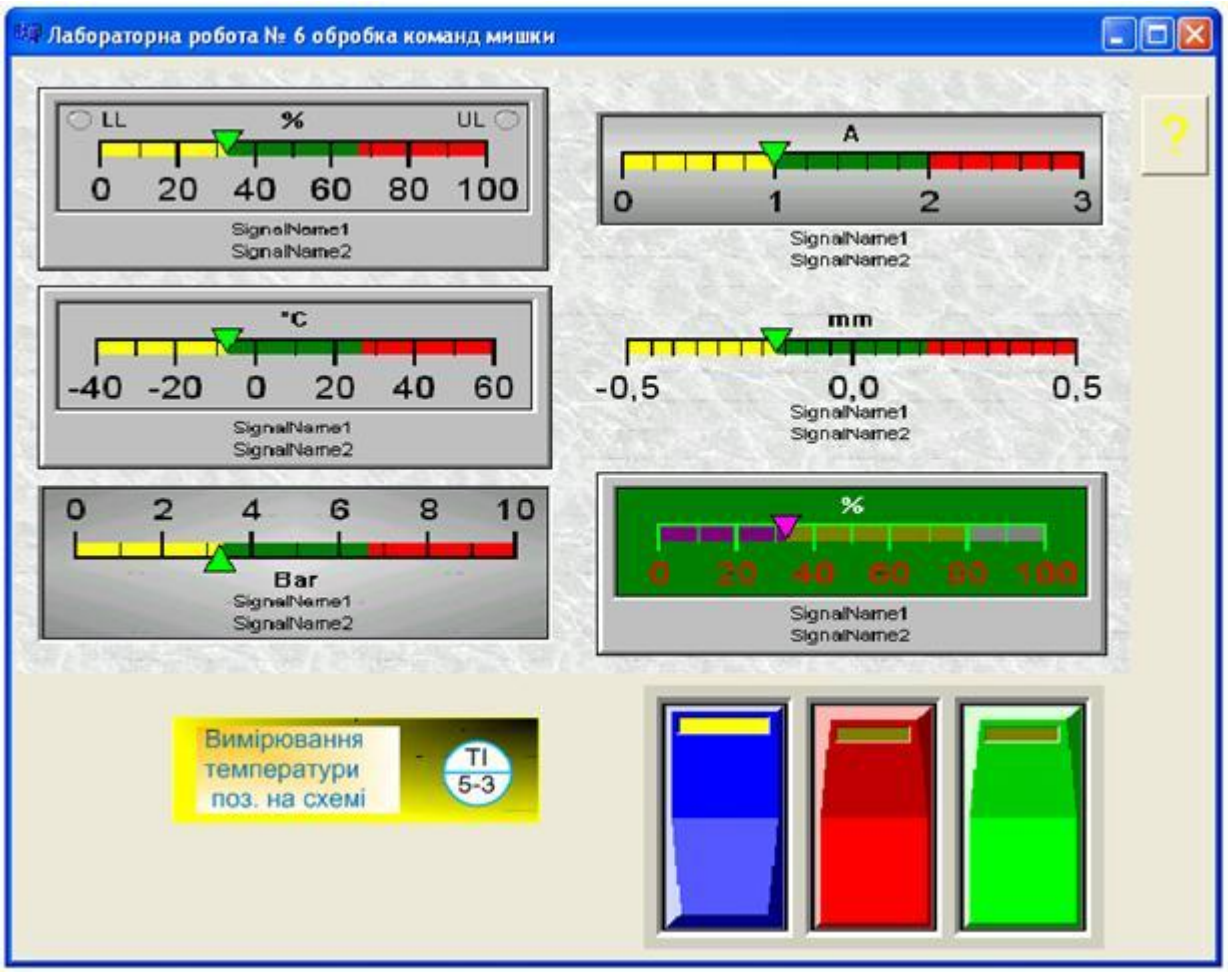


Рис. 6-6.

Крок 2. Виберіть в бібліотеці компонент ***VCL*** сторінку ***Additional*** і з неї розташуйте на форму компоненти ***Image*1, *Image*2** та ***Image*3** і для них маркерною рамкою визначить розміри відповідно до рис. 6-5.

Крок 3. Підключаємо курсор виду ***crHandPoint*** до компонент ***Image*:**

- Встановіть маркерну рамку спочатку на полі ***Image*1** потім на ***Image*2** і у властивості ***Cursor*** виберіть зі списку назву курсору ***crHandPoint*** (рука).

Крок 4. Встановлюємо курсор на сітку форми програми і у списку подій відкриваємо шаблон ***OnCreate*** , щоби можна було задати відповідні початкові рисунки та необхідні змінні у форму програми.

Крок 5. Установить курсор на компоненту ***Image*1** і відкрийте шаблон для події ***OnMouseDown*** та заповніть відповідні команди (дивись файл модуля форми ***U_work*_6.*cpp***).

Крок 6. Установить курсор на компоненту ***Image*2** і відкрийте шаблон для події ***OnMouseDown*** (переключення трьох кнопок) та заповніть відповідні команди (дивись файл модуля форми ***U_work*_6.*cpp***).

Крок 7. На форму C++ програми встановіть кнопку ***SpeedButton*1** і для неї задайте подію ***OnClick*** , де визначить вид курсору ***crHandPoint*** (дивись файл модуля форми ***U_work*_6.*cpp***).

Крок 8. При допомозі влаштованого редактора ***Image Editor*** створіть нестандартний тимчасовий курсор і його зареєструйте у файл ресурсів створюваної прикладної ***C***++ програми. Для цього виконайте:

- Виберіть в меню таки команди ***Tools/ ImageEditor***;
- Виберіть в меню таки команди ***File/ Open*** і знайдіть для створюваної прикладної ***C***++ програми файл ресурсів проекту (**.res).** У результаті відкриється вікно зі структурою та вменю буде додано ***Resource***;
- Виберіть таки команди ***Resource/ New/ Cursor*** , щоби додати нову групу ***Cursor*** і нове ім'я ***Cursor*1**.
- На ім'ям ***Cursor1*** зробіть щиглика правою клавішею мишки і виберіть команду ***Rename*** та задайте нове ім'я нестандартного курсору.
- На ім'ям нового курсору зробіть двійного щиглика мишкою і створіть свою піктограму (рисунок) нестандартного курсору.
- При допомозі команд ***File / Save*** збережіть новий курсор у файл ресурсів проекту, створюваної ***C***++ програми.

Крок 9. У файл ***U_work*_6.*cpp*** збережіть для модуля прикладної програми таки тексти: до вказівок препроцесору; до об'яв змінних; до операторів записаних у шаблони з обробки подій мишки:

```
/*   Текст файлу U_work_6.cpp  .   */
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
```

```
//--- стан рисунків при їх переключенні
int pic1_On_Off, pic2_On_Off, pic3_On_Off, pic4_On_Off;
//--- указники для завантаження рисунків
TImage *PictK1;
TImage *PictP1;
TImage *PictIm3;
const int crLi = 1; //--- константа для реєстрування нестандартного курсору
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner): TForm(Owner)
{
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
TImage *Pict = new TImage(Form1);
Pict->AutoSize = true;
// -- завантаження рисунків у поля Image
Pict->Picture->LoadFromFile("Pic_1.bmp");
Image1->Canvas->CopyRect(Image1->ClientRect, Pict->Canvas,
       Rect(0,0,Pict->Width,Pict->Height));
Pict->Picture->LoadFromFile("But1_On.bmp");
Image2->Canvas->CopyRect(Image2->ClientRect, Pict->Canvas,
       Rect(0,0,Pict->Width,Pict->Height));
Pict->Picture->LoadFromFile("Image3.bmp");
Image3->Canvas->CopyRect(Image3->ClientRect, Pict->Canvas,
       Rect(0,0,Pict->Width,Pict->Height));
pic1_On_Off = 1;
delete Pict;
//--- завантаження і реєстрування тимчасового курсору до приладів
Screen->Cursors[crLi] = LoadCursor(HInstance,"LI");
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
void __fastcall TForm1::Image1MouseDown(TObject *Sender,
     TMouseButton Button, TShiftState Shift, int X, int Y)
{
TImage *PictIm3 = new TImage(Form1);  //--- до повідомлення назв приладів
PictIm3->AutoSize = true;
 if((Sender == Image1) && (Screen->Cursor == crHandPoint) )
 Image1->Cursor = crLi;     /* задаємо нестандартний курсор, створений в редакторі
Image Editor */
else
{PictIm3->Picture->LoadFromFile("Image3.bmp");
Image3->Canvas->CopyRect(Image3->ClientRect, PictIm3->Canvas,
       Rect(0,0,PictIm3->Width,PictIm3->Height));
}
if((X < Image1->Width / 2) && (Y < Image1->Height / 3))
```

```
{
PictIm3->Picture->LoadFromFile("LI.bmp");
Image3->Canvas->CopyRect(Image3->ClientRect, PictIm3->Canvas,
      Rect(0,0,PictIm3->Width,PictIm3->Height));
}
if((X > Image1->Width / 2) && (X < 2*(Image1->Width / 2)) && (Y < Image1->Height
/ 3))
{
PictIm3->Picture->LoadFromFile("EI.bmp");
Image3->Canvas->CopyRect(Image3->ClientRect, PictIm3->Canvas,
      Rect(0,0,PictIm3->Width,PictIm3->Height));
}
if((X < Image1->Width / 2) && (Y > Image1->Height / 3) && (Y < 2*(Image1->Height
/ 3)))
  {
PictIm3->Picture->LoadFromFile("TI.bmp");
Image3->Canvas->CopyRect(Image3->ClientRect, PictIm3->Canvas,
      Rect(0,0,PictIm3->Width,PictIm3->Height));
  }
if((X > Image1->Width / 2) && (X < Image1->Width ) && (Y > Image1->Height / 3)
                                    && (Y < 2*(Image1->Height / 3)))
{
PictIm3->Picture->LoadFromFile("GE.bmp");
Image3->Canvas->CopyRect(Image3->ClientRect, PictIm3->Canvas,
      Rect(0,0,PictIm3->Width,PictIm3->Height));
}
if((X < Image1->Width / 2) && (Y > 2*(Image1->Height / 3)))
{
PictIm3->Picture->LoadFromFile("PI.bmp");
Image3->Canvas->CopyRect(Image3->ClientRect, PictIm3->Canvas,
      Rect(0,0,PictIm3->Width,PictIm3->Height));
}
if((X > Image1->Width /2 ) && (Y > 2*(Image1->Height / 3) ) && (Y < Image1-
>Height ))
{
PictIm3->Picture->LoadFromFile("GEK.bmp");
Image3->Canvas->CopyRect(Image3->ClientRect, PictIm3->Canvas,
      Rect(0,0,PictIm3->Width,PictIm3->Height));
}
}

//---------------------------------------------------------------------------
void __fastcall TForm1::Image2MouseDown(TObject *Sender,
    TMouseButton Button, TShiftState Shift, int X, int Y)
{
TImage *PictK1 = new TImage(Form1);  //---для кнопок
PictK1->AutoSize = true;
TImage *PictP1 = new TImage(Form1);  //---для приладів
```

```
PictP1->AutoSize = true;
      switch(1+X /(Image2->Width /3) + 3*(Y /(Image2->Height/2)))
      {
case 1: PictK1->Picture->LoadFromFile("But1_On.bmp");
Image2->Canvas->CopyRect(Image2->ClientRect, PictK1->Canvas,
      Rect(0,0,PictK1->Width,PictK1->Height));
if(pic1_On_Off ==0) /*--- перевіряємо, щоби не було мигатиння при повторних
включеннях */
{
PictP1->Picture->LoadFromFile("Pic_1.bmp");
Image1->Canvas->CopyRect(Image1->ClientRect, PictP1->Canvas,
      Rect(0,0,PictP1->Width,PictP1->Height));

pic1_On_Off =1; pic2_On_Off =0; pic3_On_Off =0; pic4_On_Off =0;
 } break;
case 2: PictK1->Picture->LoadFromFile("But2_On.bmp");
Image2->Canvas->CopyRect(Image2->ClientRect, PictK1->Canvas,
      Rect(0,0,PictK1->Width,PictK1->Height));
if(pic2_On_Off == 0)  //-- перевіряємо, щоби не було мигатиння при повторних
включеннях
{
PictP1->Picture->LoadFromFile("Pic_2.bmp");
Image1->Canvas->CopyRect(Image1->ClientRect, PictP1->Canvas,
      Rect(0,0,PictP1->Width,PictP1->Height));
pic1_On_Off =0;pic2_On_Off =1;pic3_On_Off =0;pic4_On_Off =0;
} break;
case 3: PictK1->Picture->LoadFromFile("But3_On.bmp");
Image2->Canvas->CopyRect(Image2->ClientRect, PictK1->Canvas,
      Rect(0,0,PictK1->Width,PictK1->Height));
if(pic3_On_Off == 0) //--- перевіряємо, щоби не було мигатиння при повторних
включеннях
{
PictP1->Picture->LoadFromFile("Pic_3.bmp");
Image1->Canvas->CopyRect(Image1->ClientRect, PictP1->Canvas,
      Rect(0,0,PictP1->Width,PictP1->Height));
 pic1_On_Off =0;pic2_On_Off =0;pic3_On_Off =1;pic4_On_Off =0;
 } break;
case 4: PictK1->Picture->LoadFromFile("But123_Off.bmp");
Image2->Canvas->CopyRect(Image2->ClientRect, PictK1->Canvas,
      Rect(0,0,PictK1->Width,PictK1->Height));
if(pic4_On_Off == 0)//--- перевіряємо, щоби не було мигатиння при повторних
включеннях
{
PictP1->Picture->LoadFromFile("Pic_4.bmp");
Image1->Canvas->CopyRect(Image1->ClientRect, PictP1->Canvas,
      Rect(0,0,PictP1->Width,PictP1->Height));
 pic1_On_Off =0;pic2_On_Off =0;pic3_On_Off =0;pic4_On_Off =1;
 }  break;
```

```
case 5: PictK1->Picture->LoadFromFile("But123_Off.bmp");
Image2->Canvas->CopyRect(Image2->ClientRect, PictK1->Canvas,
      Rect(0,0,PictK1->Width,PictK1->Height));
if(pic4_On_Off == 0) //--- перевіряємо, щоби не було мигатиння при повторних включеннях
{
PictP1->Picture->LoadFromFile("Pic_4.bmp");
Image1->Canvas->CopyRect(Image1->ClientRect, PictP1->Canvas,
      Rect(0,0,PictP1->Width,PictP1->Height));
      pic1_On_Off =0;pic2_On_Off =0;pic3_On_Off =0;pic4_On_Off =1;
 }   break;
case 6: PictK1->Picture->LoadFromFile("But123_Off.bmp");
Image2->Canvas->CopyRect(Image2->ClientRect, PictK1->Canvas,
      Rect(0,0,PictK1->Width,PictK1->Height));
if(pic4_On_Off == 0) /* перевіряємо, щоби не було мигатиння при повторних включеннях */
{
PictP1->Picture->LoadFromFile("Pic_4.bmp");
Image1->Canvas->CopyRect(Image1->ClientRect, PictP1->Canvas,
      Rect(0,0,PictP1->Width,PictP1->Height));
      pic1_On_Off =0;pic2_On_Off =0;pic3_On_Off =0;pic4_On_Off =1;
 }
      }
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
void __fastcall TForm1::SpeedButton1Click(TObject *Sender)
{
Screen->Cursor = crHandPoint;
}
//---------------------------------------------------------------------------
//---------------------------------------------------------------------------
void __fastcall TForm1::Image1MouseUp(TObject *Sender, TMouseButton Button,
    TShiftState Shift, int X, int Y)
{
Screen->Cursor = crDefault;
}
//---------------------------------------------------------------------------
```

При виконаннанні лабораторної роботи № 6 вікно ***C***++ програми повинно мати зображення, яке показано на рис. 6-6.

Контрольні запитання до лабораторної роботи № 6

1. Пояснить, яким чином підключається файл нестандартного курсора у файл ресурсів прикладної ***C***++ програми.

2. Покажіть оператори в ***C***++ програмі котрі забезпечують вивід повідомлень у компоненту ***Image*3.**
3. Пояснить структуру операторів, записаних у перемикач ***switch*( )** при обробці команд мишки.
4. Поясніть заголовки функцій, яки обробляють команди мишки над зображеннями приладів на рис. 6-3 та рис. 6-4.
5. Поясніть оператори, забезпечуючих заміни курсора мишки на інший вид при виконанні команд мишки над зображеннями приладів та нестандартних кнопок у полі ***Image*2.**

Лабораторна робота № 7

Компоненти C++ Builder та інструменти і функції для побудування графічних елементів до зображень у вікні прикладної програми

Ціль виконання лабораторної роботи. Інтегроване програмувальне середовище ***C***++ ***Builder*** має відповідний набір компонент, інструментів, властивостей, методів і функцій для побудування графічних елементів до зображень на формі прикладної ***C***++ програми і тому задача даної роботи складається у вивченні:

- методики використання з ***VCL*** компонент для побудування графічних елементів на формі програми;
- техники роботи з ***Canvas*** – канвою форми прикладної програми;
- методики рисування кольорових елементів до графічних зображень на формі програми;
- правил налагодження компоненти для рисування відповіного типу лінії: суцільної лінії; штрихової ліні; пунктірної лінії та штрих-пунктірної лінї;

- техніки рисування базових графічних фігур: дуга чи круг або еліпс; сегмент або сектор; прямокутник та інші фігури за допомого інструмента – перо ***Pen***.

Загальні зведення до компонент та інструментів для рисування графічних елементів на формі програми

Відображати і вводити графічну інформацію можна на поверхні будь-якої віконної компоненти, що має властивість ***Canvas*** – канва (полотно) та ще в бібліотеці ***C++ Builder*** для відображення графічної інформації маються компоненти, список яких наведено в таблиці 7.1.

Компоненти для відображення графічних зображень на формі. Таблиця 7.1

Піктограма	Найменування компонента	Сторінка бібліотеки	Призначення
	**IMAGE** (зображення)	***Additional***	Використовується для відображення графіки: піктограм, бітових матриць і метафайлів
	**DBImage** (зображення даних)	***Data Controls***	Зв'язаний з даними аналог ***Image***
	***PaintBox*** (вікно для малювання)	***System***	Використовується для створення на формі деякої області, у якій можна малювати

Канва (***Canvas***) не є компонентою. Багато компонентів у ***C++ Builder*** мають властивість ***Canvas***, що представляє собою область компоненти, на якій можна малювати або відображати готові зображення. Цю властивість мають форми, графічні компоненти I***mage***, ***DBimage***, ***PaintBox*** і багато інших. Канва містить властивості та методи, що істотно спрощують роботу з графікою при розробках прикладних програм в ***C++ Builder*** і всі складні взаємодії графічних зоражень із системою заховані для користувача.

Кожна крапка канви має координати ***X*** та ***Y*** . Система координат канви починається з лівого верхнього кута області полотна. Кордината ***X*** зростає при змінах від ліворуч до праворуч, а кордината ***Y*** змінюється зверху до низу. Кординати змінюються у пикселях. ***Пиксель*** – це найменший елемент у графічному зображенні, яким можна маніпулювати. Найважливіша

властивість пикселя - це колір. Для опису кольору використовується тип ***Tcolor***. Малювати на канві можна різними способами. ***Перший варіант (Pixels)*** – рисування по окремим пикселям. ***Другий варіант (Pen)*** – рисування пером ***Pen***. Властивість ***Pixels*** у канві являє собою двовимірний масив, що визначає кольори крапок на канві:

Cavas->Pixels[*int X*][*int Y*]

З масивом пикселей можна роботать як з будь-якою властивістю: змінювати колір, задавати пикселю нове значення, чи визначати його колір константою. Наприклад, (*Cavas->Pixels*[*int X*][*int Y*] = *clBlue;)* - тут задається пикселю синій колір. При необхідності намалювати деякий графік функції ***F(x)*** на канві компоненти ***Image*** можна, якщо задані значення ***Ymin, Ymax*** і ***Xmin, Xmax*** та якщо записати наступні оператори у програмі:

```
//-------------------------------------------------------------
                    float x, y;       //---координати функції
                    float PX, PY;     //---координати пикселей
                    for( PX=0; PX <= Image1->Width;  PX ++ )
{  //--x – координата, що відповідає пикселю з координатою PX
            x = Xmin + PX*(Xmax - Xmin) / Image -> Width;
            y = F(x);
//--PY –координата пикселя, що відповідає координаті y
PY = Image1->Width – (y - Ymin)* Image->Height/(Ymax - Ymin);
//---установлюється чорний колір обраного пикселя
            Image1->Canvas->Pixels[PX][PY] = clBlack;
}
//-------------------------------------------------------------
```

У канви мається властивість ***Pen*** - перо. Це об'єкт, у свою чергу має ряд властивостей. Властивість ***Color*** – колір, якою наноситься малюнок, властивість ***Widht*** - ширина лінії, що малюється. Ширина лінії задається в пикселях і за замовчуванням дорівнює **1**. Властивість ***Style*** – визначає вид лінії. Ця властивість може приймати такі значення:

psSolid	Суцільна лінія
psDash	Штрихова лінія
psDot	Пунктирна лінія
psDashDot	Штрих-пунктирна лінія

psDashDotDot	Лінія, щочергує штрих і два пунктири
psClear	Відсутність лінії
psInsideFrame	Суцільна лінія і при Widht > 1 допускаються кольори відмінні від кольорів Windows

Усі стилі зі штрихами і пунктирами доступні тільки при **Widht = 1**. У противному випадку лини цих стилів малюються як суцільні. Стиль ***psinsideFrame*** - єдиний, котрий допускає довільні кольори. Колір лінії при інших стилях округляється до найближчого з палітри ***Windows***. Властивість ***PenPos*** - визначає в координатах канви поточну позицію пера. Зміна значень ***PenPos***, тобто переміщення пера без промальовування лінії виконується за допомогою методу канви ***MoveTo(X,Y)***. Значення (***X,Y***) – це координати крапки, у яку переміщується перо. Ця поточна крапка стає вихідної, від якої методом ***LineTo(X,Y)*** можна провести лінію в крапку з координатами ***(X,Y)***. При цьому поточна крапка переміщується в кінцеву крапку лінії і новий виклик ***LineTo(X,Y)*** буде проводити лінію з цієї нової поточної крапки.

Перо ***Pen*** може малювати не тільки прямі лінії, але і базові фігури. Нижче перераховані деякі з методів канви, що використовують перо ***Pen*** для малювання фігур:

Arc	Малює дугу чи окружності еліпса
Chord	Малює замкнуту фігуру, обмежену дугою чи окружності еліпса і хордою
Ellipse	Малює еліпс або окружність
Pie	Малює сектор окружності або еліпса
Polygon	Малює замкнуту фігуру з кусочно-лінійною границею
Polyline	Малює кусочно-лінійну криву
Rectangle	Малює прямокутник
RoundRect	Малює прямокутник з округленими кутами

У канви також мається властивість ***Brush*** - кисть. Ця властивість визначає тіло (фон) і заповнення замкнутих фігур на канві. ***Brush*** - це об'єкт,

що має у свою чергу ряд властивостей. Властивість ***Color*** визначає колір заповнення. Властивість ***Style*** - визначає шаблон заповнення (штрихування).

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Постановка задачі по програмуванню у лабораторній роботі

Задача А. Необхідно створити прикладну ***C***++ програму, яка буде демонструвати побудування графіку функції ***Sin*(*X*)** двома способами:

- перший варіант - рисування графіку на канві за допомогою ***Pixels***;
- другий варіант - рисування графіку за допомогою інструмента ***Pen*** методами ***MoveTo(X,Y)*** та ***LineTo(X,Y)*** .

Задача Б. Необхідно створити прикладну ***C***++ програму, яка буде демонструвати побудування фігур за допомогою інструмента ***Pen*** у режимі рисування стандартних графічних фігур.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Завдання для аналізу і обробки результатів роботи запрограмованої C++ програми

Після запуску в роботу ***C***++ програми лабораторної роботи № 7 для захисту результатів програмування необхідно виконати таке:

- нарисувати блок-схему алгоритму з роботи програми до задачі А лабораторної роботи № 7;
- нарисувати блок-схему алгоритму з роботи програми до задачі Б лабораторної роботи № 7;
- нарисувати блок-схему алгоритму по виконанню операторів з рисування графіку на канві за допомогою ***Pixels***;
- нарисувати блок-схему алгоритму по виконанню операторів з рисування графіку за допомогою інструмента ***Pen.***

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Методика виконання задачі А.

***Порядок дій і команд при виконанні програмування задачі А*:**

Крок 1. На диску ***D***:\ створіть папку ***Lab***_7 і в цій папці сформуйте внутрішню папку ***Sin*_7** для файлів проекту ***P_Sin*_7**.

Крок 2. Виберіть у бібліотеці компонентів сторінку ***Additional*** і на форму розмістить компоненту ***Image*1**, а потім маркерною рамкою визначить розмір під малюнок графіку.

Крок 3. На сторінці ***Standart*** виберіть кнопку і установить на форму кнопку ***Button*1** і замінить назву на напис "Розрахувати".

Крок 4. Визначить подію ***OnClik*** для кнопки ***Button*1**.

Крок 5. У шаблон ***TForm*1::*Button*1*Click*(*TObject *Sender*)** для обробки події додайте код, щоби будувався графік функції ***Sin*(*X*)** за допомогою зміни координат ***Pixels*** :

```
 //---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "U_Sin_7.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
}
//---------------------------------------------------------------------------
#include <math.h>
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 #define Pi 3.14159
 float X,Y;        // ----координати функції
 int PX,PY;        // ---координати пикселей
 for (PX = 0; PX <= Image1->Width; PX++)
  {
//------X - координата, що відповідає пикселю з координатою PX
   X = PX * 4 * Pi / Image1->Width;
   Y =  sin(X);
//-----PY - координата пикселя, що відповідає координаті Y
   PY = Image1->Height - (Y+1) * Image1->Height / 2;
//-----Встановлюється колір обраного пикселя
   Image1->Canvas->Pixels[PX][PY] = clRed;
  }
}
//---------------------------------------------------------------------------
```

Крок 6. Для побудування інструментом ***Pen*** другого графіку функції ***Sin*(*X*)** додайте на форму компоненту ***Image*2** .

Крок 7. Необхідно сформувати два однакових поля під графіки:

- Утримуючи натиснутої клавішу ***Shift*** замаркіруйте ***Image*1**, ***Image*2** і виконаєте команду правою клавішею на формі.
- В меню виберіть команду ***Size*** і установить опцію ***Grow to largest*** чи ***Shrink to smallest***.
- Змінить в оброблювачу події ***TForm*1::*Button*1*Click*(*TObject *Sender*)** код програми на такий текст:

```
//---------------------------------------------------------------------------
#include <math.h>
//---------------------------------------------------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 #define Pi 3.14159

 float X,Y;        // ----координати функції
 int PX,PY;        // ---координати пикселей
Image2->Canvas->MoveTo(0,Image2->Height/2);

 for (PX = 0; PX <= Image1->Width; PX++)
  {
//-----X - координата, що відповідає пикселю з координатою PX
   X = PX * 4 * Pi / Image1->Width;
   Y =  sin(X);
//----PY - координата пикселя, що відповідає координаті Y
   PY = Image1->Height - (Y+1) * Image1->Height / 2;
//-----Установлюється колір обраного пикселя
   Image1->Canvas->Pixels[PX][PY] = clRed;
//-----Проводиться лінія на другому графіку чорним кольором
   Image2->Canvas->LineTo(PX,PY);
  }
}

//---------------------------------------------------------------------------
```

Крок 8. Виконаєте компіляцію ***C***++ програми та спостеригайте результат рисування функції у вигляді рис. 7-1.

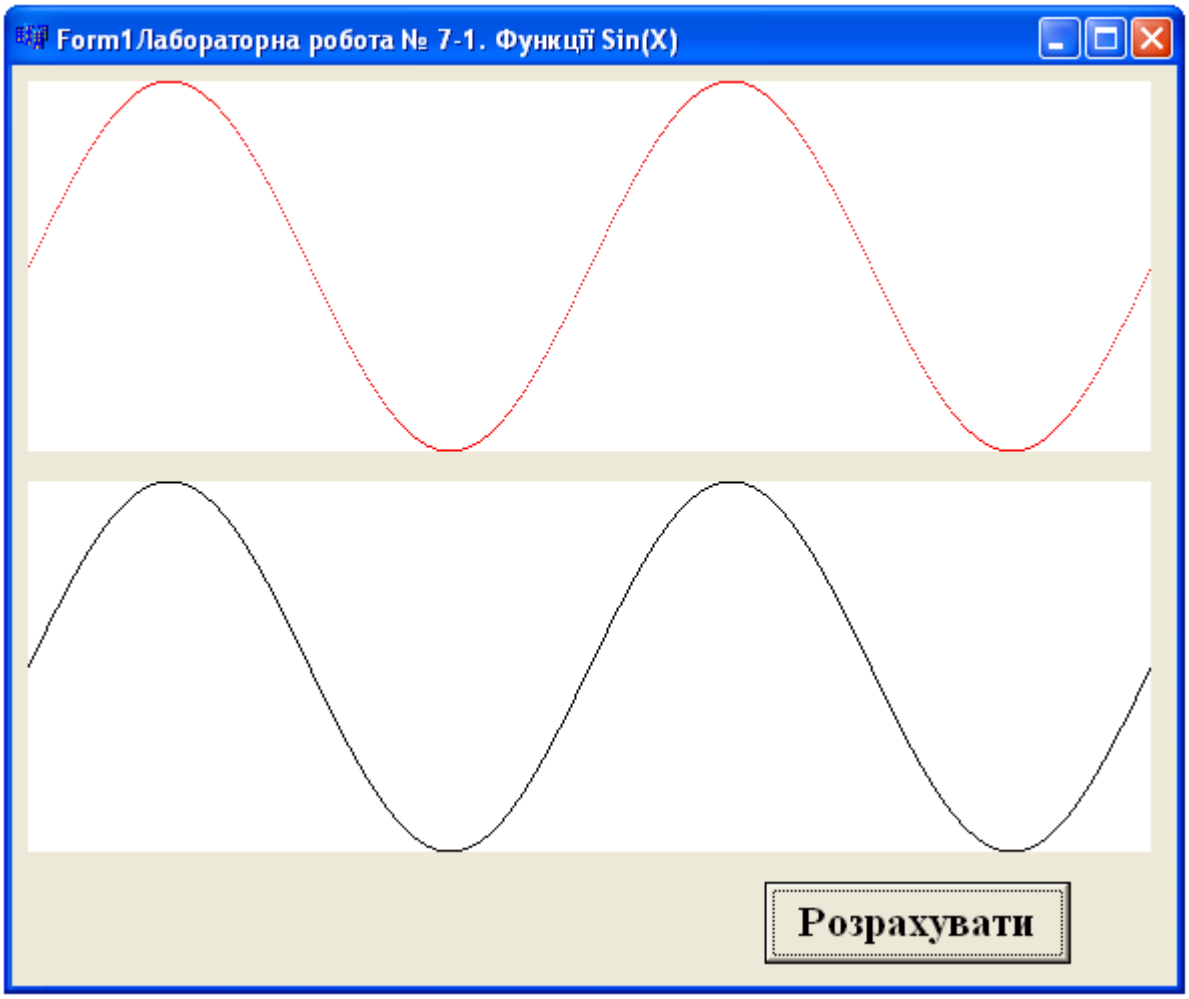


Рис. 7-1.

Методика виконання задачі Б.

Порядок дій і команд при виконанні програмування задачі Б:

Крок 1. На диску ***D*:** , у створеній папці ***Lab*_7** , потрібно сформувати внутрішню папку ***Figures*_7** для файлів проекту ***P_Figures*_7**.

Крок 2. На формі задайте поле ***Image*1** і заповнить в програмний модуль ***U_Figures*_7** наступний текст коду програми:

```
//---------------------------------------------------------------------------
    #include <vcl.h>
    #pragma hdrstop

    #include "U_figures_7.h"
    //---------------------------------------------------------------------------
    #pragma package(smart_init)
    #pragma resource "*.dfm"
    TForm1 *Form1;
    //---------------------------------------------------------------------------
```

```
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
 Image1->Canvas->Font->Style << fsBold;
 Image1->Canvas->Arc(10,10,90,90,90,50,10,50);
 Image1->Canvas->TextOut(40,60,"Arc");
 Image1->Canvas->Chord(110,10,190,90,190,50,110,50);
 Image1->Canvas->TextOut(135,60,"Chord");
 Image1->Canvas->Ellipse(210,10,290,50);
 Image1->Canvas->TextOut(230,60,"Ellipse");
 Image1->Canvas->Pie(310,10,390,90,390,30,310,30);
 Image1->Canvas->TextOut(340,60,"Pie");
 TPoint points[5];
 points[0] = Point(30,150);
 points[1] = Point(40,130);
 points[2] = Point(50,140);
 points[3] = Point(60,130);
 points[4] = Point(70,150);
 Image1->Canvas->Polygon(points,4);
 Image1->Canvas->TextOut(30,170,"Polygon");
 points[0].x += 100;
 points[1].x += 100;
 points[2].x += 100;
 points[3].x += 100;
 points[4].x += 100;
 Image1->Canvas->Polyline(points,4);
 Image1->Canvas->TextOut(130,170,"Polyline");
 Image1->Canvas->Rectangle(230,120,280,160);
 Image1->Canvas->TextOut(230,170,"Rectangle");
 Image1->Canvas->RoundRect(330,120,380,160,20,20);
 Image1->Canvas->TextOut(325,170,"RoundRect");
}
//---------------------------------------------------------------------------
```

Крок 3. Виконайте компіляцію ***C***++ програми і спостеригайте результат рисування у вигляді рис. 7-2.

Рис.7-2.

<u>Контрольні запитання до лабораторної роботи № 7</u>

1. Пояснить по листигну ***C***++ програми, оператори до алгоритму рисування графіка функції методом ***Pixels.***
2. Пояснить по листигну ***C***++ програми, оператори до алгоритму рисування графіка функції методом ***Pen.***
3. Пояснить складові у запису оператора:

   Image1->Canvas->Pixels[PX][PY] = clBlack; .
4. Пояснить складові у запису оператора:

   Image2->Canvas->MoveTo(0,Image2->Height/2); .
5. Пояснить складові у запису оператора:

   Image2->Canvas->LineTo(PX,PY);

Лабораторна робота № 8

Техніка створювання графічної анімації в C++ програмі на основі використання компоненти «Animate»

<u>Ціль виконання лабораторної роботи</u>. Інтегроване програмувальне середовище ***C***++ Builder має можливості для розробки прикладних ***C***++

програм з відеороликами та мультипликаціями на основі компоненти «***Animate***» і тому задача даної роботи складається у вивченні:

- техніки і методики використання компоненти "***Animate***" при побудованні прикладної C++ програми с запуском відеоролика на формі вікна;
- техніки і методики використання компоненти "***Animate***" при побудованні прикладної ***C***++ програми с запуском мультипликації на формі вікна;
- правил до налагодження властивостей компоненти "***Animate***" для запуска відеоролика на формі вікна;
- правил до налагодження властивостей компоненти "***Animate***" для запуска мультипликації на формі вікна.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Загальні зведення до компоненти «Animate» з програмувального середовища C++ Builder

Компонента ***Animate*** забезпечує просту без звуку графічну анімацію і значок якої знаходиться в бібліотеці ***VCL*** на закладці ***Win*****32** і додається на форму прикладної ***C***++ програми стандартним діями на формі. Після встановлення компоненти ***Animate*** на форму вікна програми потрібно виконати налагодження властивостей – задати до них відповідні значення.

Компонента ***Animate*** забезпечує показ анімації на формі програми з файлів формату ***.avi*** (***Audio Video Interleaved***) без звукового запису. Файли формату ***.avi*** є послідовностями кадрів зображень у вигляді побітової матриці.

Список властивостей компоненти ***Animate*** наводиться в таблиці № 8-1.

Таблиця № 8-1.

Назва властивості	Опис властивості
Name	Ім'я компоненти використовується для доступу к властивостям та для управління роботою.
FileName	Ім'я AVI-файлу, в якому знаходиться анімація і буде відображатися компонентою.

FrameWidth	Ширина кадрів анімації.
FrameHeight	Висота кадрів анімації.
FrameCount	Кількість кадрів анімації.
AutoSize	Признак для автоматичного встановлення відповідного розміру компоненти з розмірами кадрів.
Center	Признак центрування кадрів у полі компоненти. Якщо значення властивості завдано ***true*** і розмір компоненти більше розміру кадрів (AutoSize =false), кадри анімації розташовуються у центрі поля компоненти.
StartFrame	Номер кадру, з якого починається відображення анімації.
StopFrame	Номер кадру, на якому зупиняється анімація.
Active	Признак активізації відображення анімації.
Color	Цвет фона компоненти, на якому відображається анімація.
Transparent	Режим використання «прозорого» кольору при відображенні анімації.
Repetitions	Кількість повторів у анімації.
CommonAVI	Налаштовується стандартна Windows.

Для показу анімації у компоненти ***Animate*** властивість можна налаштувати:

- на значення ***FileName*** для вибору відповідного файлу ***.avi*** для анімації методом ***Play;***

- на значення ***CommonAVI*** для анімації стандартних кліпів передбачених у ***Windows*** , встановленому на комп'ютері.

Властивість ***CommonAVI*** має тип ***TCommonAVI*** і визначено у вигляді значень наступного об'єднання:

```
enum TCommonAVI { aviNone, aviFindFolder,
                  aviFindFile, aviFindeComputer,
                  aviCopyFiles, aviCopyFile,
                  aviRecycleFile, aviEmptyRecycle,
                  aviDeleteFile }.
```

У компоненти ***Animate*** передбачені обробки в програмі таких подій: ***OnClose, OnOpen, OnStart*** та ***OnStop***, яки генерують у відповідні моменти закриття і відкриття компоненти та початок і закінчення анімації на формі вікна прикладної програми.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Постановка задачі по програмуванню у лабораторній роботі

Задача А. Необхідно створити C++ програму для роботи у ***Windows***, в який у вікні форми буде виконуватись анімація рисунків компонентою ***Animate*** при властивості ***FileName*** після натискання кнопки **"ЗАПУСК"**. Також у ***C***++ програмі повинна виконуватись безперевна анімація та анімація по окремим кадрам з відповіного файлу, який обирається у діалозі з каталогу файлів при натисканні кнопки «ОБРАТИ».

Задача Б. Необхідно створити ***C***++ програму для роботи у ***Windows***, в який на формі вікна повинні виконуватись стандартні кліпи компонентою ***Animate*** при налагодженні властивості ***CommonAVI*** .

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Завдання для аналізу і обробки результатів роботи запрограмованої C++ програми

Після запуску в роботу ***C***++ програми лабораторної роботи № 8 для захисту результатів програмування необхідно виконати таке:

- нарисувати блок-схему алгоритму з роботи програми до задачі А лабораторної роботи № 8;
- нарисувати блок-схему алгоритму з роботи програми до задачі Б лабораторної роботи № 8;
- нарисувати блок-схему алгоритму дій ***C***++ програми при налагодженні у компоненти ***Animate*** властивості ***StartFrame***;
- нарисувати блок-схему алгоритму дій ***C***++ програми при налагодженні у компоненти ***Animate*** властивості ***StopFrame.***

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Методика виконання задачі А.

***<u>Порядок дій і команд</u> при виконанні програмування задачі А*:**

<u>Крок 1.</u> Активізуємо проект файлів для нової ***C***++ програми та виконайте команду для зберігання їх на диску в задану папку (каталог):

- Виконайте команду ***File/New Application***, щоб з'явилася нова чиста форма ***Form*1**;
- Перейдіть у вікно інспектора об'єкту і у властивості ***Caption*** задайте назву «Лабораторна робота № 8 **“Анімація кадрів з рисунками“**», щоб на заголовку вікна форми ***C***++ програми цей напис з’явився замість назви ***Form*1**;

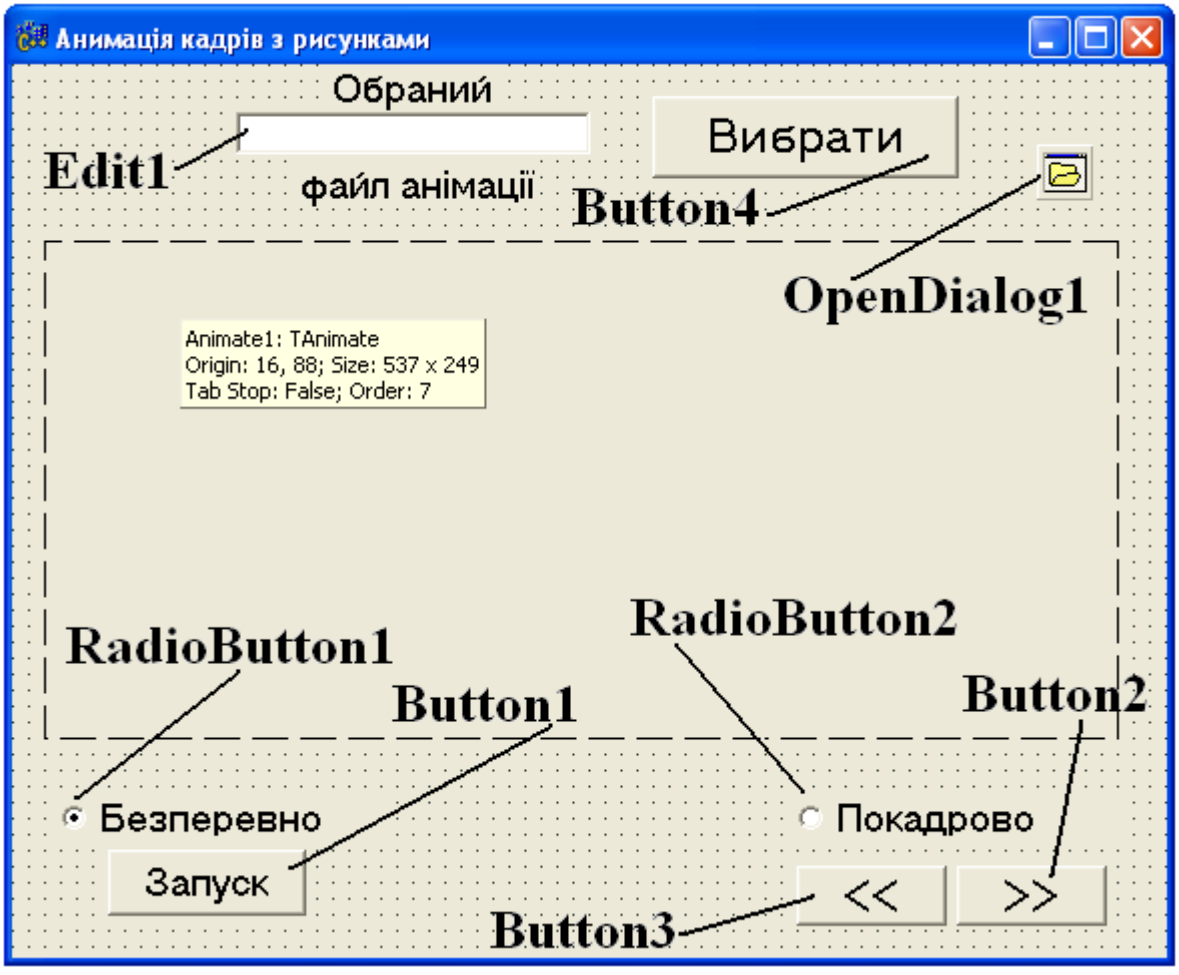


Рис. 8-1. Форма *C*++ програми для виконання анімації з обраного файлу.

- Збережить новий проект файлів разом з порожньою формою і змінами в заголовку вікна. Це автоматично активізує новий проект файлів та ***C***++ Builder запам'ятає шлях для швидкого збереження змін у файлах, розробляємої ***C***++ програми. Для збереження файлів проекту відкрийте

на диску таки папки ***D:\LA_NN\LAB-A_Animate-AVI***, де ***NN*** - номер навчальної групи;

- Далі у меню ***File*** виберіть команду ***Save Project As*** та при появі запиту на збереження змінюємо назву файлу ***Unit*1** на файл ***U_Animate-AVI.cpp***, а пропонуєму назву проекту ***Project1.bpr*** змінюємо на ***P_ Animate-AVI.bpr***.

Крок 2. Далі потрібно створити форму прикладної ***C***++ програми з елементами (компонентами), показаними на рис. 8-1.

Крок 3. Після закінчення розроби по рис. 8-1 форми ***C***++ програми потрібно у вікні редактора текстів сформувати наступний листинг команд для обробок подій при виконанні анімації AVI-файлу:

```
//---------------------------- U_Animate_avi.cpp ---------------------------------
#include <vcl.h>
#pragma hdrstop
#include "U_Animate_avi.h"
//--------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
int CFrame = 0;
//--------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
      : TForm(Owner)
{
}
//--------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
TSearchRec sr; // знаходиться інформація о файлі, знайденому функцією FindFirst
// пошук AVI-файлу в активному каталозі
      if(FindFirst("*.avi",faAnyFile,sr) == 0)
      {Edit1->Text = sr.Name;
// анімація без звукової доріжки
try
      {Animate1->FileName = sr.Name;
      }
catch(Exception &e)
      {
      return;
      }
```

```
          RadioButton1->Enabled = true;
          RadioButton2->Enabled = true;
        Button1->Enabled = true;
         }
}
//---------------------------------------------------------------------------
// обробка події до кнопки "Вибрати"----------------------
void __fastcall TForm1::Button4Click(TObject *Sender)
{
//відкриття каталогу, з якого виконано запуск програми
//OpenDialog1->InitialDir = "D:\00new\filecopy.avi\";
// вивід списку з AVI-файлами
 OpenDialog1->FileName = "*.avi";
 if( OpenDialog1->Execute())
 // обрано файл і натиснута кнопка "Відкрити"
 try
 {
  Animate1->FileName = OpenDialog1->FileName;
  }
       catch(Exception &e)
       {Edit1->Text - "";
  // блокування кнопок керування при avi-файлу зі звуком
  RadioButton1->Enabled = false;
   RadioButton2->Enabled = false;
   Button1->Enabled = false;
   Button2->Enabled = false;
   Button3->Enabled = false;
  //повідомлення про помилуку
  AnsiString msg = "Помилка відкриття файлу" +
                   OpenDialog1->FileName +
                   "\n Можливо анімаціця має звук.";
   ShowMessage(msg);
   return;
   }
   Edit1->Text = OpenDialog1->FileName; //відображення назви файлу
   RadioButton1->Checked = true;//режим безперевного перегляду
   Button1->Enabled = true; //кнопка "Пуск" доступна
   Button2->Enabled = true; // кнопка "Попередній кадр" недоступна
   Button3->Enabled = true; //кнопка "Наступний кадр" недоступна
   }
//------- щелчок по кнопці "ПУСК/СТОП"-----------------------------------
void __fastcall TForm1::Button1Click(TObject *Sender)
{
 if( Animate1->Active)
 {
 //анімація відображається, потрибен щелчок на кнопці "СТОП"
 Animate1->Active = false;
 Button1->Caption = "Пуск";
```

```
 RadioButton2->Enabled =true;
 }
 else //щелчок на кнопці "ПУСК"
 {
 //активізувати відображення анімації
 Animate1->StartFrame = 1; //з першого кадру
 Animate1->StopFrame = Animate1->FrameCount; //по останьому кадру
 Animate1->Active = true;
 Button1->Caption = "Стоп";
 RadioButton2->Enabled = false;
 }
}
//---------------------------------------------------------------------------
//---вибір режиму перегляду повної анімації--------------------------------

void __fastcall TForm1::RadioButton1Click(TObject *Sender)
{
 Button1->Enabled = true; //кнопка "ПУСК/СТОП" доступна
 // блокування кнопок для просмотру по кадрам
 Button2->Enabled = false;
 Button3->Enabled = false;
  Animate1->Active = false;
}
//вібір режиму перегляду по кадрам ------------------------------------

void __fastcall TForm1::RadioButton2Click(TObject *Sender)
{
  Button1->Enabled = false; //кнопка "ПУСК/СТОП" недоступна
  Button2->Enabled = true;   //кнопка "Наступний кадр" доступна
  Button3->Enabled = false; //кнопка "Попередній кадр" недоступна
  //відображення першого кадру
  Animate1->StartFrame = 1;
  Animate1->StopFrame = 1;
  Animate1->Active = true;
  CFrame = 1; //запоминання номеру відображаємого кадру
}
//---------------------------------------------------------------------------
//щелчок на кнопуі "Наступний кадр"-----------------------------------------
void __fastcall TForm1::Button2Click(TObject *Sender)
{
CFrame = CFrame + 1;
//відображення кадру
Animate1->StartFrame = CFrame;
Animate1->StopFrame = CFrame;
Animate1->Active = true;
if(CFrame > 1)
Button3->Enabled = true;
if(CFrame == Animate1->FrameCount) // відображено останній кадр
```

```
Button2->Enabled =false;
}
//---------------------------------------------------------------------------
 //щелчок на кнопці "Попередній кадр"----------------------------------
void __fastcall TForm1::Button3Click(TObject *Sender)
{
if(CFrame == Animate1->FrameCount) //останній кадр
Button2->Enabled = true;
CFrame--;
//показ кадру
Animate1->StartFrame = CFrame;
Animate1->StopFrame = CFrame;
Animate1->Active = true;
if(CFrame == 1)
Button3->Enabled = false; //кнопка "Наступний кадр" недоступна
}
//---------------------------------------------------------------------------

//------------------------------ U_Animate_avi.h -----------------------------------
//---------------------------------------------------------------------------
#ifndef U_Animate_aviH
#define U_Animate_aviH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published: // IDE-managed Components
        TRadioButton *RadioButton1;
        TRadioButton *RadioButton2;
        TButton *Button1;
        TButton *Button2;
        TButton *Button3;
        TEdit *Edit1;
        TButton *Button4;
        TAnimate *Animate1;
        TOpenDialog *OpenDialog1;
        TLabel *Label1;
        TLabel *Label2;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall Button4Click(TObject *Sender);
        void __fastcall Button1Click(TObject *Sender);
```

```
        void __fastcall RadioButton1Click(TObject *Sender);
        void __fastcall RadioButton2Click(TObject *Sender);
        void __fastcall Button2Click(TObject *Sender);
        void __fastcall Button3Click(TObject *Sender);

private:        // User declarations
public:                 // User declarations
        __fastcall TForm1(TComponent* Owner);
};

//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif

//------------------------------ P_Animate_avi.cpp ----------------------------------
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("U_Animate_avi.cpp", Form1);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                Application->Initialize();
                Application->CreateForm(__classid(TForm1), &Form1);
                Application->Run();
        }
        catch (Exception &exception)
        {
                Application->ShowException(&exception);
        }
        catch (...)
        {
                try
                {
                        throw Exception("");
                }
                catch (Exception &exception)
                {
                        Application->ShowException(&exception);
                }
        }
        return 0;
}
//---------------------------------------------------------------------------
```

Крок 4. Для кнопки ***RadioButton*1** необхідно задати у властивості ***Checked*** значення ***true*** , щоб на формі з'явився значок чорної крапки.

Крок 5. Виконуємо ***C*++** програму з різними файлами анімації та при різних налаштуваннях з таблиці № 8-1 властивостей компоненти ***Animate***.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Методика виконання задачі Б.

Порядок дій і команд при виконанні програмування задачі Б:

Крок 1. Активізуємо проект файлів для нової ***C*++** програми та виконуємо команду для зберігання їх на диску в задану папку (каталог):

- Виконайте команду ***File/New Application***, щоб з'явилася нова чиста форма ***Form*1**;
- Перейдіть у вікно інспектора об'єкту і у властивості ***Caption*** задайте назву **"Анімація стандартних кліпів Windows "**, щоб на заголовку вікна форми ***C*++** програми замість назви ***Form*1** з'явився напис " **Анімація стандартних кліпів Windows** ";
- Збережить новий проект файлів разом з порожньою формою і змінами в заголовку вікна. Це автоматично активізує новий проект файлів і ***C*++** Builder запам'ятає шлях для швидкого збереження змін у файлах, розробляємої ***C*++** програми. Для збереження файлів проекту відкрийте на диску таки папки ***D:\LA_NN\LAB-Б_Animate-AVI***, де ***NN*** - номер навчальної групи;
- Далі у меню ***File*** виберіть команду ***Save Project As*** та при появі запиту на збереження змінюємо назву файлу ***Unit*1** на файл ***U_Animate-AVI.cpp***, а пропонуємо назву проекту ***Project*1.bpr** змінюємо на ***P_ Animate-AVI.bpr***.

Крок 2. Далі потрібно скомпонувати форму вікна прикладної програми з елементами, показаними на рис. 8-2.

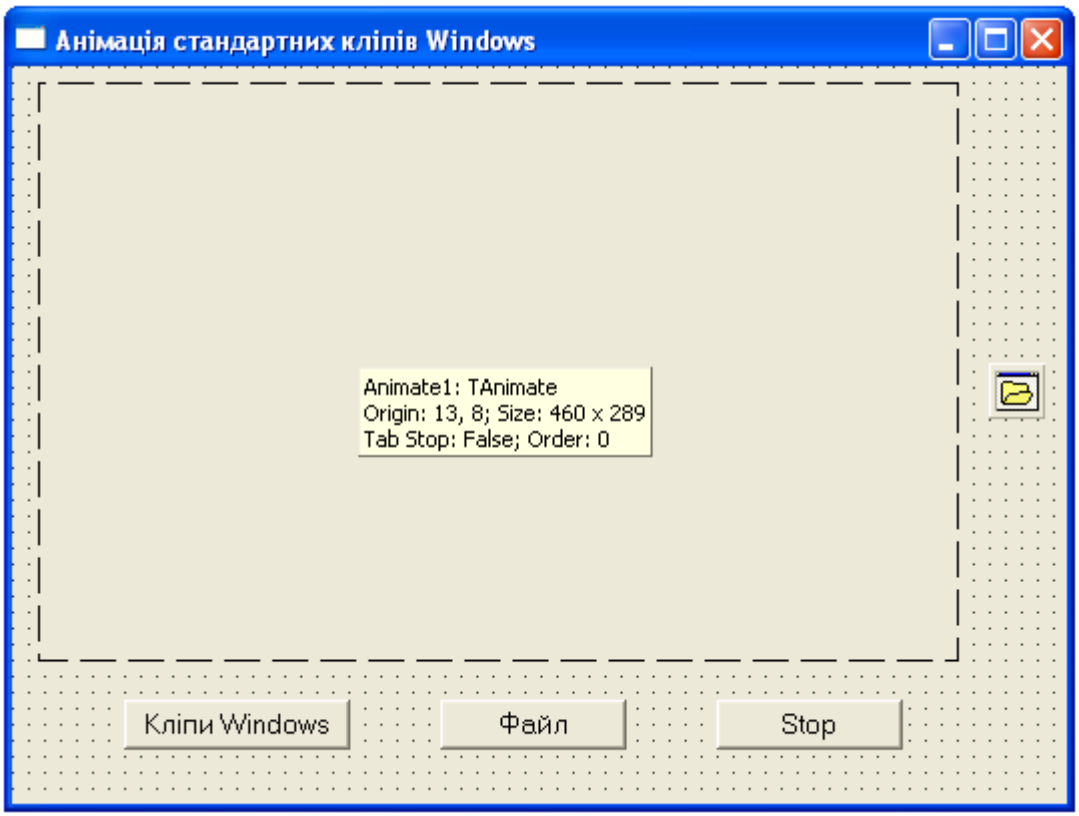


Рис. 8-2 . Зображення форми *C++* програми для виконання задачі Б.

Крок 3. Після закінчення розроби по рис. 8-2 форми програми потрібно у редакторі текстів сформувати наступний лістинг команд для обробок подій при виконанні анімації AVI-файлу.

```
//--------------------- U_Animate-AVI.cpp ----------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "UAnimate.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
int i;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
    : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void __fastcall TForm1::BWindClick(TObject *Sender)
{
 Animate1->Visible = true;
 i = 1;
```

```
 Animate1->CommonAVI = aviFindFolder;
 Animate1->Active = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::BStopClick(TObject *Sender)
{
 Animate1->Stop();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::BFileClick(TObject *Sender)
{
 if (OpenDialog1->Execute())
  {
   i = 9;
   Animate1->FileName = OpenDialog1->FileName;
   Animate1->Visible = true;
   Animate1->Active = true;
  }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Animate1Stop(TObject *Sender)
{
 i++;
 switch (i)
 {
  case 2: Animate1->CommonAVI = aviFindFile;
          break;
  case 3: Animate1->CommonAVI = aviFindComputer;
          break;
  case 4: Animate1->CommonAVI = aviCopyFiles;
          break;
  case 5: Animate1->CommonAVI = aviCopyFile;
          break;
  case 6: Animate1->CommonAVI = aviRecycleFile;
          break;
  case 7: Animate1->CommonAVI = aviEmptyRecycle;
          break;
  case 8: Animate1->CommonAVI = aviDeleteFile;
 }
  if (i < 9) Animate1->Active = true;
  else Animate1->Visible = false;
}
//---------------------------------------------------------------------------
```

```
//---------------------- U_Animate-AVI.h -----------------------------------
//---------------------------------------------------------------------------
#ifndef UAnimateH
#define UAnimateH
```

```
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <Dialogs.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:      // IDE-managed Components
    TAnimate *Animate1;
    TOpenDialog *OpenDialog1;
    TButton *BWind;
    TButton *BFile;
    TButton *BStop;
    void __fastcall BWindClick(TObject *Sender);
    void __fastcall BStopClick(TObject *Sender);
    void __fastcall BFileClick(TObject *Sender);
    void __fastcall Animate1Stop(TObject *Sender);
private:    // User declarations
public:             // User declarations
    __fastcall TForm1(TComponent* Owner);
};

//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
//---------------------------------------------------------------------------
```

P_Animate-AVI.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
USERES("PAnimate.res");
USEFORM("UAnimate.cpp", Form1);
//---------------------------------------------------------------------------

WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
    try
    {
            Application->Initialize();
            Application->CreateForm(__classid(TForm1), &Form1);
            Application->Run();
    }
    catch (Exception &exception)
```

```
{
        Application->ShowException(&exception);
    }
    return 0;
}

//---------------------------------------------------------------------------
```

<u>Контрольні запитання до лабораторної роботи № 8</u>

1. Пояснить по листигну ***C***++ програми призначення ***RadioButton1Click*** при виконанні операторів задачи А;
2. Пояснить по листигну ***C***++ програми призначення ***Button2Click*** при виконанні операторів задачи А;
3. Пояснить складові у запису операторів перемикача ***switch (i)*** при виконанні задачи Б.
4. Пояснить складові у файлі ***U_Animate-AVI.cp*** при виконанні задачи Б.
5. Пояснить складові у файлі ***U_Animate-AVI.h*** при виконанні задачи Б.

Лабораторна робота № 9

Дослідження алгоритму з динамічної графіки на формі вікна прикладної С++ програми

<u>Ціль виконання лабораторної роботи</u>. Інтегроване програмувальне середовище ***C***++ ***Builder*** має можливості для розробки прикладних ***C***++ програм з динаміною графікою на основі використання компоненти «***Timer***» і тому задача даної роботи складається у вивченні:

- техніки і методики використання компоненти "***Timer*** " при побудованні прикладної ***C***++ програми з динамічню графікою на формі вікна програми;
- алгоритма з функціонування динамічної графіки на формі вікна програми.

Загальні зведення до динамічної графіки на формі вікна

Динамічна графіка на формі вікна програми будується на принципах мультиплікації окремих кадрів з зображенням. Для динамічної графіки необхідно будувати набір кадрів і кожних з них трішечки повинен відрізнятися деяким елементом від попереднього кадру в послідовності. Динамічне графічне зображення виникає за рахунок перегляду кадрів у завданій послідовності при відповідному інтервалі часу між змінами кадрів. Зменшення інтервалу часу між кадрами збільшує динаміку графічного зображення, а збільшення інтервалу часу – навпаки уповільнює динаміку спостереження графічних кадрів.

В ***C++ Builder*** для розробки прикладних програм з керуванням інтервалів часу використовується компонента ***Timer*** . Таймер має дві властивості , яки дозволяють керувати роботою компоненти ***Timer*** :

- перша властивість ***Interval*** визначає в мілісекундах інтервал зміни часу між кадрами динамічної графіки або іншими подіями в прикладній програмі. Наприклад, значення 500 для властивості ***Interval*** фізично відповідає 0,5 сек.;

- друга властивість ***Enabled*** доступність до роботи компоненти.

Властивість ***Interval*** задає період спрацьовування таймеру і може налаштовуватись при проектуванні форми або встановлюватись програмно при роботі програми. Якщо при запуску програми властивість ***Interval*** завдано, тоді активною буде у ***C++*** програмі обробка події ***OnTimer*** кожен раз після закінчення інтервалу часу, завданого у властивості ***Interval*** . Якщо задати для ***Interval*** = **0** або для ***Enabled*** = ***false,*** тоді таймер стане вимкнутим. Для включення відліку часу потрібно задати ***Enabled*** = ***true***. Оператори у ***C++*** програмі для керування властивостями компоненти таймер записуються у таких форматах:

```
Timer->Interval = 5000;
Timer->Enabled = true; .
```

Кнопка ***SpeedButton*** з фіксацією положення (натиснута / не натиснута) може мати напис або зображення іконки. У властивості ***Caption*** задається напис на кнопку та можливо додати зображення іконки у властивості ***Glyph*** .

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Постановка задачі по програмуванню у лабораторній роботі

При виконанні лабораторної роботи потрібно створити ***C***++ програму для роботи у ***Windows***, в який повинна бути можливість для спостереження за динамічною графікою зображення та досліджувати у покроковому режимі при допомозі відладчіка ***C***++ ***Builder*** виконання алгоритму з динамічної графіки на формі вікна.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Завдання для аналізу і обробки результатів роботи запрограмованої C++ програми

Після запуску в роботу ***C***++ програми лабораторної роботи № 9 для захисту результатів програмування необхідно виконати таке:

- визначити перелік функцій, яки забезпечують динаміку для графічного зображення на формі ***C***++ програми;
- скласти специфікацію на визначені функції та їх призначення у ***C***++ програмі з динамічним графічним зображенням на формі;
- розробити блок-схеми алгоритмів до функцій, записаних до специфікації;
- розробити блок-схему алгоритму з функціонування ***C***++ програми з динамічним графічним зображенням у вікні лабораторної роботи № 9.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Методика виконання задачі .

Порядок дій і команд при виконанні програмування задачі **:**

Крок 1. Активізуємо проект файлів для нової ***C***++ програми та виконайте команду для зберігання їх на диску в задану папку (каталог):

- Виконайте команду ***File/New Application***, щоб з'явилася нова чиста форма ***Form*****1**;

- Перейдіть у вікно інспектора об'єкту і у властивості ***Caption*** задайте назву лабораторної роботи **" Динамічна графіка “**, щоб у заголовку вікна форми ***C++*** програми замість назви ***Form*1** з'явився напис "**Динамічна графіка** ";
- Збережить новий проект файлів разом з порожньою формою і змінами в заголовку вікна. Це автоматично активізує новий проект файлів і ***C++*** Builder запам'ятає шлях для швидкого збереження змін у файлах, розробляємої програми. Для збереження файлів проекту відкрийте на диску таки папки ***D:\LA_NN\LAB-Dinamica***, де ***NN*** - номер навчальної групи;
- Далі у меню ***File*** виберіть команду ***Save Project As*** та при появі запиту на збереження змінюємо назву файлу ***Unit*1** на файл ***U_dinamica.cpp***, а пропонуєму назву проекту ***Project*1.*bpr*** змінюємо на ***P_dinamica.bpr.***

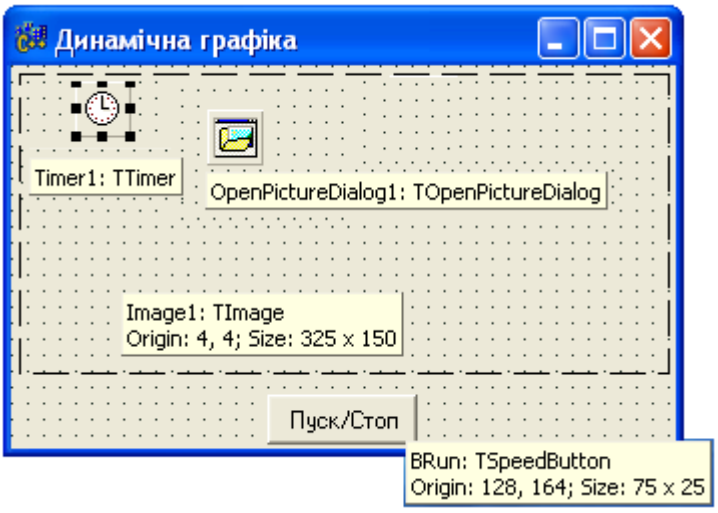


Рис. 9-1. Форма програми для роботи з динамічною графікою.

Крок 2. Далі потрібно скомпонувати форму для прикладної програми з елементами, показаними на рис. 9-1.

Крок 3. Після закінчення розроби форми програми по рис. 9-1 потрібно у вікні редактора текстів сформувати наступний лістинг команд для обробок подій при роботі з динамічною графікою в ***Image***:

```
//------------------------------ U_dinamica.cpp ------------------------------------

//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop

#include "U_dinamica.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
short int num = 0;
short int H=20;                  // шаг
short int Xpos = 2 * H;          // координати тіла
short int Ypos = 120;            // "земля"
short int Hmen = 30;             // висота тіла
short int Rhead = 10;            // радіус голови
short int Rhead2 = Rhead / 2;    // радіус ладоньок
short int revers = 1;            // напрямок руху
short int L = H * 1.41;          // довжина ноги

//---------------------------------------------------------------------------
void __fastcall TForm1::Draw()
{
short int Yhead;                 // координата голови знизу
switch (num)
{
 case 0:
   Yhead = Ypos-H-Hmen;
   Image1->Canvas->MoveTo(Xpos-H,Ypos);
   Image1->Canvas->LineTo(Xpos,Ypos-H);                    // нога
   Image1->Canvas->LineTo(Xpos+H,Ypos);                    // друга нога
   Image1->Canvas->MoveTo(Xpos,Ypos-H);
   Image1->Canvas->LineTo(Xpos,Yhead);                     // тіло
   Image1->Canvas->MoveTo(Xpos+revers*H,Yhead-H);
   Image1->Canvas->LineTo(Xpos,Yhead+4);                   // рука
   Image1->Canvas->Ellipse(Xpos+revers*H-Rhead2,Yhead-H-
Rhead2,Xpos+revers*H+Rhead2,Yhead-H+Rhead2);
   Image1->Canvas->LineTo(Xpos+revers*H,Yhead+H);          // друга рука
   Image1->Canvas->Ellipse(Xpos+revers*H-Rhead2,Yhead+H-
Rhead2,Xpos+revers*H+Rhead2,Yhead+H+Rhead2);
   Image1->Canvas->Ellipse(Xpos-Rhead,Yhead,Xpos+Rhead,Yhead-2*Rhead);
```

```
  Image1->Canvas->Rectangle(Xpos-Rhead,Yhead-2*Rhead-1,
              Xpos+Rhead,Yhead-2*Rhead-4);                    // шляпа
  break;
 case 1:
  Yhead = Ypos-L-Hmen;
  Image1->Canvas->MoveTo(Xpos,Ypos);
  Image1->Canvas->LineTo(Xpos,Yhead);
  Image1->Canvas->MoveTo(Xpos,Yhead+4);
  Image1->Canvas->LineTo(Xpos+revers*L,Yhead+4);
  Image1->Canvas->Ellipse(Xpos+revers*L-Rhead2,Yhead+4-
Rhead2,Xpos+revers*L+Rhead2,Yhead+4+Rhead2);
  Image1->Canvas->Ellipse(Xpos-Rhead,Yhead,Xpos+Rhead,Yhead-2*Rhead);
  Image1->Canvas->Rectangle(Xpos-H / 2,Yhead-2*Rhead-1,
                    Xpos+H / 2,Yhead-2*Rhead-4);
}
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
 Draw();
 if ((Xpos >= Image1->Picture->Width-H)||(Xpos <= H))
  revers = -revers;
 Xpos = Xpos + revers * H;
 num = 1 - num;
 Draw();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::BRunClick(TObject *Sender)
{
 Timer1->Enabled = ! Timer1->Enabled;
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
//можна зняти коментар для зміни фону
//  if (OpenPictureDialog1->Execute())
//    Image1->Picture->LoadFromFile(OpenPictureDialog1->FileName);
  Image1->Canvas->Brush->Color = 0;
  Image1->Canvas->Rectangle(90,0,200,100);
  Image1->Canvas->Brush->Color = clWhite;
  Image1->Canvas->MoveTo(0,Ypos+3);
  Image1->Canvas->Pen->Width = 4;
  Image1->Canvas->LineTo(Image1->ClientWidth,Ypos+3); // çåìëÿ
  Image1->Canvas->Pen->Width = 1;
  Image1->Canvas->Pen->Mode = pmNotXor;
 Draw();
}
```

```
//---------------------------------------------------------------------------
//--------------- U_dinamica.h-------------------------------------
//---------------------------------------------------------------------------
#ifndef UMult1H
#define UMult1H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Buttons.hpp>
#include <Dialogs.hpp>
#include <ExtCtrls.hpp>
#include <ExtDlgs.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published: // IDE-managed Components
        TImage *Image1;
        TSpeedButton *BRun;
        TTimer *Timer1;
        TOpenPictureDialog *OpenPictureDialog1;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall BRunClick(TObject *Sender);
        void __fastcall Timer1Timer(TObject *Sender);
private:        // User declarations
public:                 // User declarations
        void __fastcall Draw();
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
//---------------------------------------------------------------
//------------ P_dinamica.cpp ---------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
USERES("P_dinamica.res");
USEFORM("U_dinamica.cpp", Form1);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                Application->Initialize();
                Application->CreateForm(__classid(TForm1), &Form1);
```

```
Application->Run();
        }
        catch (Exception &exception)
        {
                Application->ShowException(&exception);
        }
        return 0;
}

//---------------------------------------------------------------------------
```

<u>Контрольні запитання до лабораторної роботи № 9</u>

1. Пояснить по листигну ***C***++ програми призначення компоненти ***Timer*1** при виконанні операторів у програмі з лабораторної роботи № 9;
2. Пояснить по листигну ***C***++ програми призначення на формі компоненти ***TSpeedButton*** при виконанні програми з лабораторної роботи № 9;
3. Пояснить складові операторів у запису перемикача ***switch (num)*** при виконанні програмного модуля форми ***U_dinamica.cpp***.
4. Пояснить по листінгу програми оператори, яки вирісовують графічні елементи динамічної фігури на формі програми.
5. Пояснить оператори за допомогою яких можна змінити розміри динамічної графічної фігури у лабораторній роботі № 9.

Лабораторна робота № 10

Алгоритм роботи з фрагментами зображення розташованого у компоненті «Image»

<u>Ціль виконання лабораторної роботи</u>. Інтегроване програмувальне середовище ***C***++ Builder має можливості для розробки прикладних ***C***++ програм з обробкою фрагментів растрового зображення на основі використання компоненти «***Image***» і тому задача даної роботи складається у вивченні:

- техніки роботи з фрагментами зображення з файлу ***.bmp*** на основі програмної обробки компоненти ***Image;***

- алгоритма по формуванню фрагментів зображення з файлу ***.bmp*** ;
- алгоритма з обробки подій мишки при роботі з фрагментами зображення з файлу ***.bmp*** .

Загальні зведення до роботи з фрагментами зображення на формі вікна

Як відомо, розміщення зображення на формі вікна з файлів ***.bmp*** забезпечує компонента ***Image***. Поєднання команд з обробки подій мишки над компонентою ***Image*** дозволяє в ***C***++ програмі виконувати дії по створенню фрагментів до зображення, завантаженого з файлу ***.bmp*** , а також різні переміщення цих фрагментів зображення по формі вікна прикладної ***C***++ програми.

Постановка задачі по програмуванню у лабораторній роботі

При виконанні лабораторної роботи потрібно створити ***C***++ програму для роботи у ***Windows***, в який на формі вікна повинно виконуватись:

- завантаження на форму вікна програми зображення з файлу ***.bmp*** ;
- формування окремих фрагментів по зображенню, яке було розміщено у компоненту ***Image***;
- вільне переміщення сформованих фрагментів зображень з компоненти ***Image*** по формі вікна програми за рахунок обробки подій до маніпулятора «мишка».

Завдання для аналізу і обробки результатів роботи запрограмованої C++ програми

Після запуску в роботу ***C***++ програми лабораторної роботи № 10 для захисту результатів програмування необхідно виконати таке:

- визначити перелік функцій, яки забезпечують створення фрагментів зображень відповідних до змісту зображення з компоненти ***Image*****1**;
- скласти специфікацію на визначені функції та їх призначення у ***C***++ програмі з динамічним графічним зображенням на формі;

- розробити блок-схеми алгоритмів до функцій, записаних до специфікації;
- розробити блок-схему алгоритму з функціонування ***C***++ програми по створенню фрагментів зображень по зображенню з компоненти ***Image*1** на формі вікна лабораторної роботи № 10.

⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇⬇

Методика виконання задачі .

Порядок дій і команд при виконанні програмування задачі **:**

Крок 1. Активізуємо проект файлів для нової програми ***C***++ та виконуємо команду для зберігання їх на диску в задану папку (каталог):

- Виконайте команду ***File/New Application***, щоб з'явилася нова чиста форма ***Form*1**;
- Перейдіть у вікно інспектора об'єкту і у властивості ***Caption*** задайте назву лабораторної роботи **" Робота с фрагментами *Image* “**, щоби на заголовку вікна форми ***C***++ програми замість назви ***Form*1** з'явився напис **" Робота с фрагментами *Image* "**;
- Збережить новий проект файлів разом з порожньою формою і змінами в заголовку вікна. Це автоматично активізує новий проект файлів і ***C***++ Builder запам'ятає шлях для швидкого збереження змін у файлах, розробляємої програми. Для збереження файлів проекту відкрийте на диску таки папки ***D:\LA_NN\LAB-Fragment***, де ***NN*** - номер навчальної групи;
- Далі у меню ***File*** виберіть команду ***Save Project As*** та при появі запиту на збереження змінюємо назву файлу ***Unit*1** на файл ***U_fragment.cpp***, а пропонуєму назву проекту ***Project*1.*bpr*** змінюємо на ***P_fragment.bpr***.

Крок 2. Далі потрібно створити форму прикладної програми з елементами, показаними на рис. 10-1.

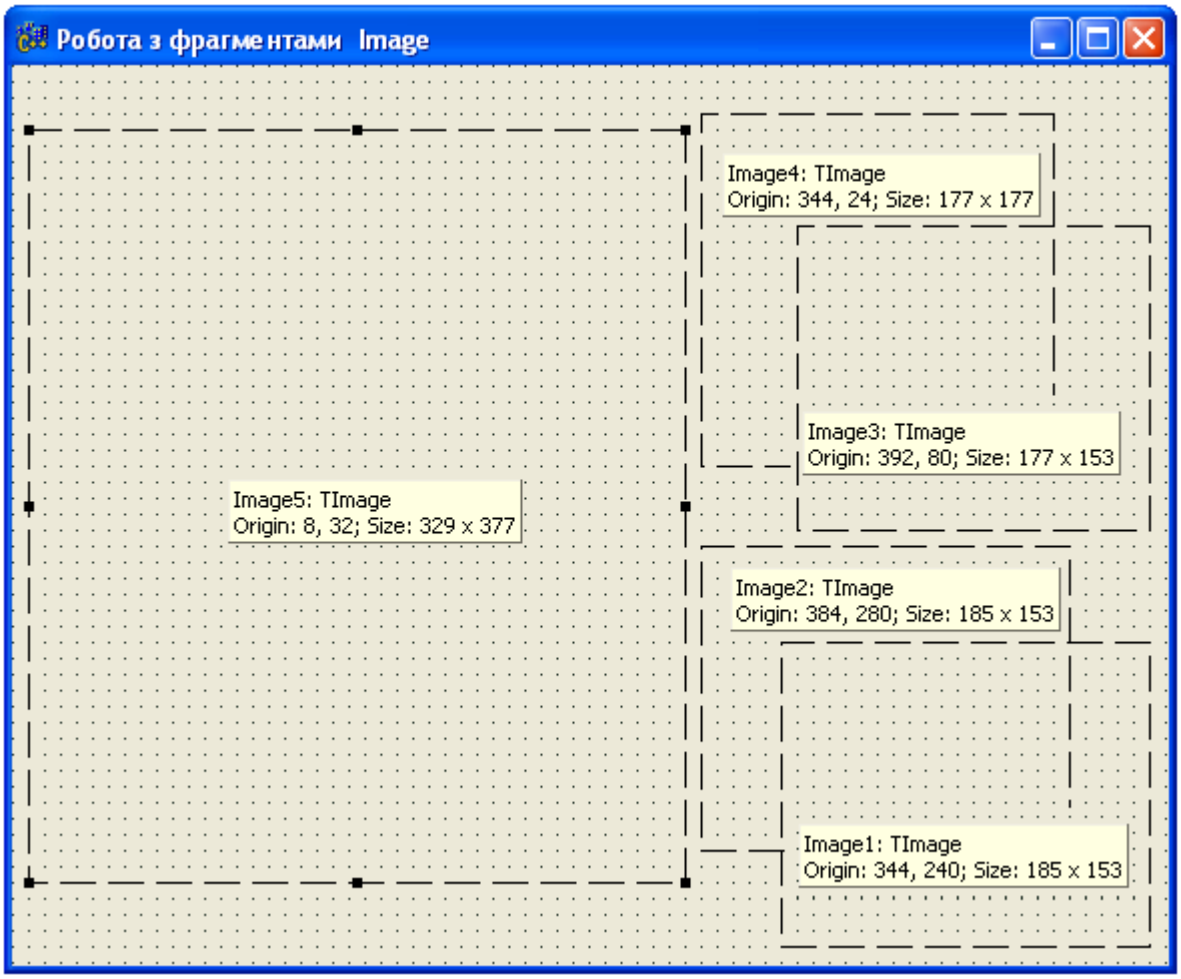


Рис. 10-1. Форма програми для роботи з фрагментами компоненти *Image*.

Крок 3. Після закінчення розроби форми програми по рис. 10-1 потрібно у вікні “Редактор коду” сформувати наступний лістинг команд для обробок подій мишки при роботі з фрагментами з компоненти ***Image***.

```
//---------------------------- U_fragment.cpp ---------------------------------
//-----------------------------------------------------------------------

#include <vcl.h>
#pragma hdrstop
#include "U_fragment.h"
//-----------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
int X0, Y0;
bool move = false;
TRect rec;
//-----------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
```

```
}
//---------------------------------------------------------------------------

void __fastcall TForm1::FormCreate(TObject *Sender)
{
TImage * Pict = new TImage(Form1);
Pict->AutoSize = true;
/* Файл для розподілу на фрагменти */
 Pict->Picture->LoadFromFile("MIK-21.BMP");
 Image5->Canvas->CopyRect(Image5->ClientRect, Pict->Canvas,
                    Rect(0,0,Pict->Width, Pict->Height));
 Image1->Canvas->CopyRect(Image1->ClientRect, Pict->Canvas,
               Rect(0,0,Pict->Width / 2,Pict->Height / 2));
 Image2->Canvas->CopyRect(Image2->ClientRect, Pict->Canvas,
       Rect(Pict->Width / 2,0,Pict->Width,Pict->Height / 2));
 Image3->Canvas->CopyRect(Image3->ClientRect, Pict->Canvas,
      Rect(0,Pict->Height / 2,Pict->Width / 2,Pict->Height));
 Image4->Canvas->CopyRect(Image4->ClientRect, Pict->Canvas,
                 Rect(Pict->Width / 2,Pict->Height / 2,
                 Pict->Width,Pict->Height ));
 delete Pict;
 Image3->ManualFloat(Rect(Form1->Left+Image3->Left,
                  Form1->Top+Image3->Top,
                  Form1->Left+Image3->Left+Image3->Width,
                  Form1->Top+Image3->Top+Image3->Height));
 Image3->ManualDock(Form1,NULL,alLeft);
 Image4->ManualFloat(Rect(Form1->Left+Image4->Left,
                  Form1->Top+Image4->Top,
                  Form1->Left+Image4->Left+Image4->Width,
                  Form1->Top+Image4->Top+Image4->Height));
 Image4->ManualDock(Form1,NULL,alLeft);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Image1MouseDown(TObject *Sender,
     TMouseButton Button, TShiftState Shift, int X, int Y)
{
  if(Button != mbLeft) return;
  X0 = X;
  Y0 = Y;
  move = true;
  ((TControl *)Sender)->BringToFront();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Image1MouseMove(TObject *Sender, TShiftState Shift,
     int X, int Y)
{
 if (move)
```

```
 {
  TImage * Im = (TImage *)Sender;
  Im->SetBounds(Im->Left + X - X0,
         Im->Top + Y - Y0, Im->Width, Im->Height);
 }
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Image1MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
 move = false;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Image2MouseDown(TObject *Sender,
      TMouseButton Button, TShiftState Shift, int X, int Y)
{
 if(Button != mbLeft) return;
 X0 = X;
 Y0 = Y;
 rec = ((TControl *)Sender)->BoundsRect;
 move = true;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Image2MouseMove(TObject *Sender, TShiftState Shift,
      int X, int Y)
{
 if(! move) return;
 Canvas->DrawFocusRect(rec);
 rec.left += X - X0;
 rec.right += X - X0;
 rec.top += Y - Y0;
 rec.bottom += Y - Y0;
 X0 = X;
 Y0 = Y;
 Canvas->DrawFocusRect(rec);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Image2MouseUp(TObject *Sender, TMouseButton Button,
      TShiftState Shift, int X, int Y)
{
 Canvas->DrawFocusRect(rec);
 if(! Shift.Contains(ssAlt))
 {
 ((TControl *)Sender)->SetBounds(
                    rec.Left + X - X0, rec.Top + Y - Y0,
                    ((TControl *)Sender)->Width,
                    ((TControl *)Sender)->Height);
 ((TControl *)Sender)->BringToFront();
```

```
 }
 move = false;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Image4EndDock(TObject *Sender, TObject *Target,
      int X, int Y)
{
 ((TControl *)Sender)->BringToFront();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormMouseMove(TObject *Sender, TShiftState Shift,
      int X, int Y)
{
  const int SC_DRAGMOVE = 0xF012;
  ReleaseCapture();
  Perform(WM_SYSCOMMAND, SC_DRAGMOVE, 0);
}
//---------------------------------------------------------------------------
//--------------Fragmet-image.cpp---------------------------------
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
USEFORM("U_fragment.cpp", Form1);
//---------------------------------------------------------------------------
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)
{
        try
        {
                Application->Initialize();
                Application->CreateForm(__classid(TForm1), &Form1);
                Application->Run();
        }
        catch (Exception &exception)
        {
                Application->ShowException(&exception);
        }
        catch (...)
        {
                try
                {
                        throw Exception("");
                }
                catch (Exception &exception)
                {
                        Application->ShowException(&exception);
                }
        }
```

```
        return 0;
}
//---------------------------------------------------------------------------

//------------- U_fragment.h-----------------------------------------------

//---------------------------------------------------------------------------
#ifndef UMoveH
#define UMoveH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published: // IDE-managed Components
        TImage *Image1;
        TImage *Image2;
        TImage *Image3;
        TImage *Image4;
      TImage *Image5;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall Image1MouseDown(TObject *Sender, TMouseButton
       Button, TShiftState Shift, int X, int Y);

        void __fastcall Image1MouseMove(TObject *Sender, TShiftState Shift,
       int X, int Y);

        void __fastcall Image1MouseUp(TObject *Sender, TMouseButton Button,
       TShiftState Shift, int X, int Y);
        void __fastcall Image2MouseDown(TObject *Sender, TMouseButton
       Button, TShiftState Shift, int X, int Y);

        void __fastcall Image2MouseMove(TObject *Sender, TShiftState Shift,
       int X, int Y);

        void __fastcall Image2MouseUp(TObject *Sender, TMouseButton Button,
       TShiftState Shift, int X, int Y);
        void __fastcall Image4EndDock(TObject *Sender, TObject *Target,
        int X, int Y);

        void __fastcall FormMouseMove(TObject *Sender, TShiftState Shift,
       int X, int Y);

private:        // User declarations
public:                 // User declarations
```

```
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

Контрольні запитання до лабораторної роботи № 10

1. Пояснить по листигну ***C***++ програми призначення операторів, записаних в ***TForm1::FormCreate*** у програмі з лабораторної роботи № 10;
2. Пояснить по листигну ***C***++ програми призначення операторів, записаних в ***TForm1::Image1MouseDown*** у програми з лабораторної роботи № 10;
3. Пояснить складові у запису структури ***class TForm1 : public TForm*** з файлу ***U_fragment.h*** ;
4. Пояснить по листінгу ***C***++ програми оператори, яки створюють фрагменти зображень з компоненти ***Image*1** лабораторної роботи № 10.
5. Пояснить оператори за допомогою яких визначаються розміри фрагментів до зображень у лабораторній роботі № 10.

Література

1. Культин Н.Б. Самоучитель C++ Builder [Текст] /Н.Б. Культин – СПб.: БХВ-Петербург, 2004. – 320с. Библиграф.: с. 318. 4000 экз.
2. Архангельский А.Я. Программирование в C++Builder6 [Текст] /А.Я. Архангельский -М.:ЗАО “Издательство БИНОМ”, 2002. – 1152 с. Библиграф.: с. 1149-1150. 4000 экз.
3. Архангельский А.Я. C++Builder6. Справочное пособие. Книга 1. Язык C++ [Текст] /А.Я. Архангельский – М.: Бином-Пресс, 2002. – 544 с. Библиграф.: с. 543. 4000 экз.
4. Архангельский А.Я. C++Builder6. Справочное пособие. Книга 2. Классы и компоненты [Текст] /А.Я. Архангельский – М.: Бином-Пресс, 2002. – 528 с. Библиграф.: с. 527. 4000 экз.