МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ТЕХНОЛОГІЇ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ЗАВДАННЯ І АЛГОРИТМИ КОМАНД ДЛЯ ПРАКТИКУМУ З ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ ПРИКЛАДНИХ ПРОГРАМ В C++ BUILDER

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для студентів, які навчаються
за спеціальністю 151 «Автоматизація та комп'ютерно-інтегровані технології»,
спеціалізацією «Автоматизація хіміко-технологічних процесів і виробництв»

Київ
КПІ ім. Ігоря Сікорського
2017

Технології розробки програмного забезпечення. Завдання і алгоритми команд для практикуму з візуального програмування прикладних програм в С++ Builder. [Електронний ресурс] : навчальний посібник для студентів спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології», спеціалізації «Автоматизація хіміко-технологічних процесів і виробництв» / КПІ ім. Ігоря Сікорського; уклад.: В. М. Ковалевський. – Електронні текстові данні (1 файл: 2,62 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2017. – 185 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 6 від 22 . 02 . 2018 р.)

за поданням Вченої ради Інженерно-хімічного факультету (протокол № 10 від 18 . 12 . 2017 р.)

Електронне мережне навчальне видання

ТЕХНОЛОГІЇ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ЗАВДАННЯ І АЛГОРИТМИ КОМАНД ДЛЯ ПРАКТИКУМУ З ВІЗУАЛЬНОГО ПРОГРАМУВАННЯ ПРИКЛАДНИХ ПРОГРАМ В C++ BUILDER

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

Відповідальний редактор: *Жученко А. І.,* завідувач кафедри «Автоматизація хімічних виробництв», доктор технічних наук, професор

Рецензент: *Шилович Т. Б.,* к.т.н., доцент кафедри «Хімічного, полімерного та силікатного машинобудування» Інженерно-хімічного факультету НТУУ «КПІ» ім. Ігоря Сікорського

Навчальний посібник «Технології розробки програмного забезпечення. Завдання і алгоритми команд для практикуму з візуального програмування прикладних програм в *C++ Builder*» створено для виконання лабораторних робіт до навчальної дисципліни "Технології розробки програмного забезпечення" і кредитного модуля «Візуальне програмування прикладних програм». Відповідно до матеріалів з тем навчальної дисципліни у навчальний посібник записано 12 завдань з описами покрокових алгоритмів команд для виконання візуального програмування прикладних програм на комп'ютеру в інтегрованому програмувальному середовищі *C++ Builder* .

Зміст

стор.

Вступ …….. 4

Завдання № 1 та алгоритми команд до теми.
Структура вікон і компонент у *C++ Builder* та техніка
їх використання для побудови прикладної програми …… 7
Завдання № 2 та алгоритми команд до теми.
Вказівки препроцесору до умовної та багато-файлової
компіляції файлів прикладної *C++* програми при створенні
консольного виконуючого коду для *MS DOS* ……..…….. 21
Завдання № 3 та алгоритми команд до теми.
Команди редактора *C++ Builder* і структура файлів
створюваних до проекту прикладної програми ……..…… 40
Завдання № 4 та алгоритми команд до теми.
Техніка використання і контролю обробки
структурних даних в *C ++* програмі …… 64
Завдання № 5 та алгоритми команд до теми.
Імітаційне моделювання алгоритму з відображення
інформації на дисплеях мікропроцесорного приладу *ITM*-11 …….. 80
Завдання № 6 та алгоритми команд до теми.
Алгоритми використання функцій для обробки команд
мишки до подій у прикладній *C++* програмі …..…… 91
Завдання № 7 та алгоритми команд до теми.
Компоненти *C++ Builder* та інструменти і функції для побудови
графічних елементів до зображень у вікні прикладної програми …….. 110
Завдання № 8 та алгоритми команд до теми.
Техніка створювання графічної анімації в *C++* програмі
на основі використання компоненти *Animate* ……..... 122
Завдання № 9 та алгоритми команд на тему.
Дослідження алгоритму з динамічної графіки
на формі вікна прикладної *C++* програми …… 140
Завдання № 10 та алгоритми команд до теми.
Алгоритм роботи з фрагментами зображення
розташованого у компоненті *Image* …….. 150
Завдання № 11 та алгоритми команд до теми.
Правила зчитування *ScanCode* та *ASCII* кодів клавіш клавіатури
комп'ютера та їх використання у прикладній *C++* програмі ……..161
Завдання № 12 та алгоритми команд до теми.
Компоненти *C++ Builder* для візуалізації структури
ієрархічних даних на носіях інформації …… 172

Література ……..... 185

В С Т У П

Навчальний посібник «*Технології розробки програмного забезпечення. Завдання і алгоритми команд для практикуму з візуального програмування прикладних програм в C++ Builder*» складено для набуття студентами практичних навичок з програмування прикладних програм у процесі виконання лабораторних робіт до навчальної дисципліни “Технології розробки програмного забезпечення” і кредитного модуля «Візуальне програмування прикладних програм». Навчальний посібник написано для виконання лабораторних робіт з візуального програмування і передбачає створення прикладних *C* та *C++* програм у інтегрованому програмувальному середовищі *C++ Builder.* [1] Теми завдань та алгоритми у практикуму до лабораторних робіт орієнтовані на вивчення техніки і методики з розробки і створення прикладних програм на мовах програмування *C* та *C++*. Лабораторні роботи кредитного модуля «Візуальне програмування прикладних програм» навчальної дисципліни “Технології розробки програмного забезпечення” виконуються відповідно до графіку занять, який наведено у таблиці В-1, де вказано кількість навчальних годин, теми та зміст заняття.

Графік проведення занять з лабораторних робіт

Таблиця В-1.

№ п/п заняття	**Кількість годин заняття**	**Зміст і тема заняття**
1	**2**	Інструктаж студентів по таких питаннях: ➢ техніка безпеки при роботі з файлами з виконанних лабораторних робіт; ➢ особливості структури інтегрованого програмувального середовища *C++ Builder*; ➢ методика і правила використання *C++ Builder* для розробки текстів до листингів та виконавчого коду прикладної програми; ➢ вимоги з підготовки студентів до заняття та оформлення результатів для захисту виконаних завдань по лабораторним роботам.
2	**2**	Виконання на комп’ютеру вказівок і команд згідно алгоритмів до завдання № 1 на тему «*Структура вікон і компонент у C++ Builder та техніка їх використання для побудови прикладної програми*».

3	**2**	Виконання на комп'ютеру вказівок і команд згідно алгоритмів до завдання № 2 на тему «*Вказівки препроцесору до умовної та багато-файлової компіляції файлів прикладної C++ програми при створенні консольного виконуючого коду для MS DOS*».
4	**2**	Виконання на комп'ютеру вказівок і команд згідно алгоритмів до завдання № 3 на тему «*Команди редактора C++ Builder і структура файлів створюваних до проекту прикладної програми*».
5	**0,66**	Захист створеної прикладної програми до завдання № 1 та результатів з індивідуального корегування студентом роботи програми і алгоритмів функцій з обробки подій до компонент *Image*1 та *Button*1.
5	**0,66**	Захист створеної прикладної програми до завдання № 2 та результатів з індивідуального корегування студентом роботи програми при виконанні умовної та багато-файлової компіляції файлів.
5	**0,66**	Захист створеної прикладної програми до завдання № 3 та результатів з індивідуального дослідження студентом створюваних файлів до проекту прикладної програми.
6	**2**	Виконання на комп'ютеру вказівок і команд згідно алгоритмів до завдання № 4 на тему «*Техніка використання і контролю обробки структурних даних в C++ програмі*».
7	**2**	Виконання на комп'ютеру вказівок і команд згідно алгоритмів до завдання № 5 на тему «*Імітаційне моделювання алгоритму з відображення інформації на дисплеях мікропроцесорного приладу ІТМ-11*».
8	**2**	Виконання на комп'ютеру вказівок і команд згідно алгоритмів до завдання № 6 на тему «*Алгоритми використання функцій для обробки команд мишки до подій у прикладній С++ програмі».*
9	**0,66**	Захист створеної прикладної програми до завдання № 4 та результатів з індивідуального аналізу студентом по контролю з обробки даних структурного типу в *C++* програмі.
9	**0,66**	Захист створеної прикладної програми до завдання № 5 та результатів з індивідуального моделювання студентом алгоритму по відображенню інформації на дисплеях мікропроцесорного приладу *ІТМ*-11.
9	**0,66**	Захист створеної прикладної програми до завдання № 6 та результатів індивідуального корегування студентом алгоритму з використання функцій при обробках команд мишки до подій в прикладній *C++* програмі.
10	**2**	Виконання на комп'ютеру вказівок і команд згідно алгоритмів до завдання № 7 на тему «*Компоненти C++ Builder та інструменти і функції для побудови графічних елементів до зображень у вікні прикладної програми».*
11	**2**	Виконання на комп'ютеру вказівок і команд згідно алгоритмів до завдання № 8 на тему «*Техніка створювання графічної анімації в C++ програмі на основі використання компоненти Animate».*

12	**2**	Виконання на комп'ютеру вказівок і команд згідно алгоритмів до завдання № 9 на тему «*Дослідження алгоритму з динамічної графіки на формі вікна прикладної C++ програми».*
13	**0,66**	Захист створеної прикладної програми до завдання № 7 та результатів індивідуального корегування студентом алгоритму з використання інструментів і функцій для побудови графічних зображень на формі вікна прикладної *C++* програми.
13	**0,66**	Захист створеної прикладної програми до завдання № 8 та результатів з індивідуального створювання студентом графічної анімації в *C++* програмі на основі використання компоненти *Animate.*
13	**0,66**	Захист створеної прикладної програми до завдання № 9 та результатів індивідуального корегування студентом алгоритму з обробки динамічної графіки на формі вікна прикладної *C++* програми».
14	**2**	Виконання на комп'ютеру вказівок і команд згідно алгоритмів до завдання № 10 на тему «*Алгоритм роботи з фрагментами зображення розташованого у компоненті Image*».
15	**2**	Виконання на комп'ютеру вказівок і команд згідно алгоритмів до завдання № 11 на тему «*Правіла зчитування ScanCode та ASCII кодів клавіш клавіатури та їх використання в прикладній C++ програмі*».
16	**2**	Виконання на комп'ютеру вказівок і команд згідно алгоритмів до завдання № 12 на тему «*Компоненти C++ Builder для візуалізації структури ієрархічних даних на носіях інформації*».
17	**0,66**	Захист створеної прикладної програми до завдання № 10 та результатів з індивідуального корегування студентом алгоритму з обробки фрагменту зображення закріпленого у компоненту *Image*.
17	**0,66**	Захист створеної прикладної програми до завдання № 11 та результатів з індивідуального зчитування студентом *ScanCode* та *ASCII* кодів клавіш клавіатури та їх використання в прикладній *C++* програмі.
17	**0,66**	Захист створеної прикладної програми до завдання № 12 та результатів з індивідуального використання студентом компоненти *C++ Builder* для візуалізації структури ієрархічних даних на носіях інформації.
18	**2**	Захисти студентами залишків не захищених лабораторних робіт та оформлення звітів до практикуму з розробки і візуального програмування прикладних *C++* програм.

Кожний студент до початку заняття з лабораторної роботи повинен з даного навчального посібника надрукувати протокол до теми відповідного завдання та описів вказівок і алгоритмів зі створення програми у *C++ Builder*.

Завдання № 1 та алгоритми до теми

Структура вікон і компонент у C++ Bulder та техніка їх використання для побудови прикладної програми

Ціль створення прикладної програми. Інтегроване програмувальне середовище *C++ Builder* має відповідну структуру вікон та меню команд і тому завдання до лабораторної роботи складається з вивчення:

- 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*).

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

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

При запуску в роботу *C++ Builder* [1],[4] на екрані дисплея комп'ютера відкриваються чотири взаємно зв'язані вікна, зображення яких показано на рис. 1-1. ***Вікно 1*** - основне керуюче вікно, що має меню команд і бібліотеки візуальних компонент (*VCL*) для використання на формах вікон програми. ***Вікно 2*** - форма (поле з сіточкою) для розміщення візуальних і невізуальних компонент потрібних для використання у *C++* програмі. ***Вікно 3*** - вікно має назву «редактор кодів вихідних текстів програми» і до якого примикає вікно структури класів та функцій, застосовуваних у створеної програмі. ***Вікно 4*** – вікно інспектора об'єктів для налагодження властивостей (*Properties*) компонент і визначення подій (*Events*), що будуть відбуватися при роботі *C++* програми; ***Вікно 5*** – вікно з деревом форм та їх компонентами і яке дозволяє бачити загальну структуру компонент, з яких складається кожна форма прикладної програми.

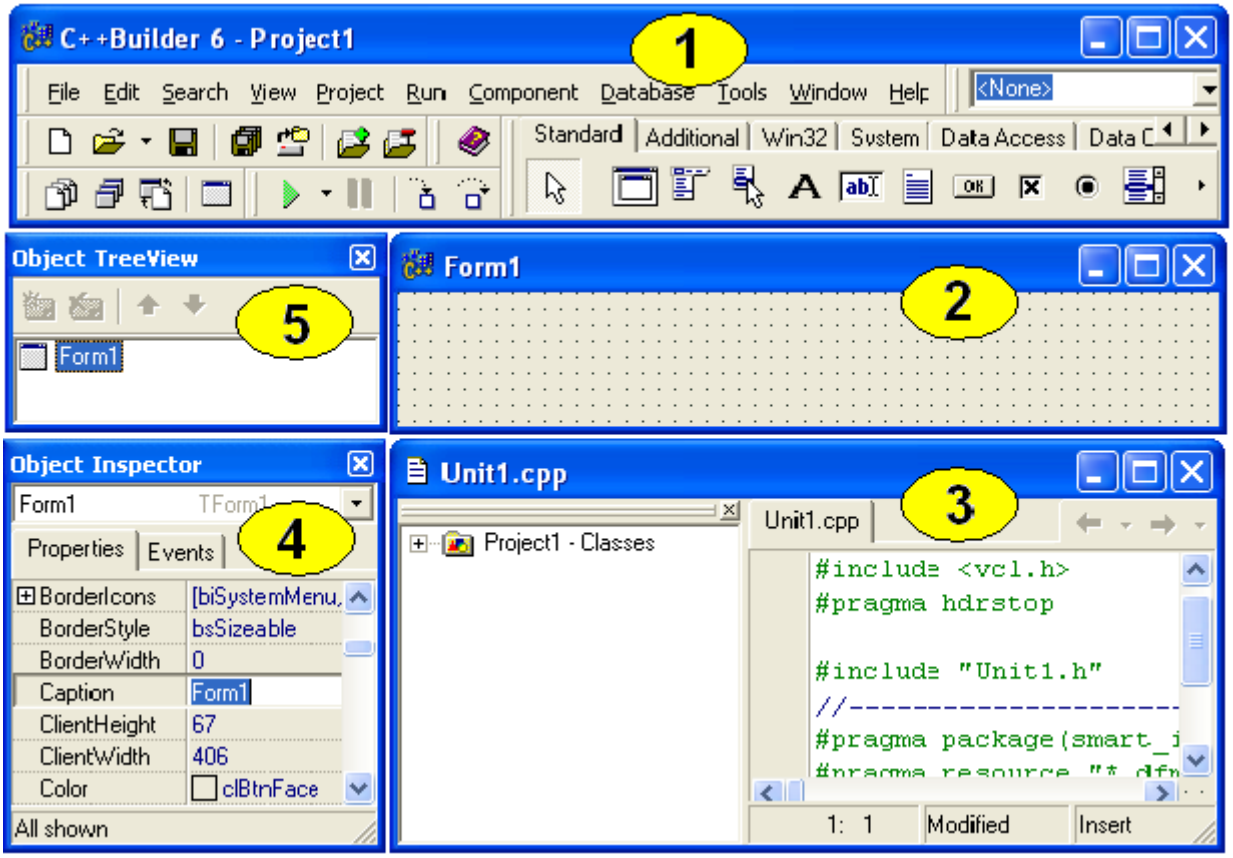


Рис. 1-1. Вікна інтегрованого програмувального середовища C++ Builder:

1 - основне керуюче вікно; 2 - вікно форми для розміщення компонент з бібліотеки *VCL*;
3 - вікно редактора коду для створення вихідних текстів *C* або *C++* до прикладної програми;
4 – вікно “Інспектор об'єкту ”; 5 - вікно дерева компонент, встановлених на форму програми.

Закриття основного керуючого вікна (1) викликає закриття одночасно і інших підлеглих та залежних вікон інтегрованого середовища *C++ Builder*. Текст програми записується і показується у вікні “Редактор коду” (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 (назва вікна) до заголовка форми програми. Команди в *C++ Builder* можна виконувати через основне меню команд або кнопками інструментів, список і опис яких наведено у таблиці 1-1.

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

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Постановка завдання до програмування програми

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

- ***Перша подія*** – поява напису у вікні текстового повідомлення "Перша програма створена для *Windows* у інтегрованому середовищі *C++ Builder*", якщо користувач виконує натискання мишкою кнопки "СТАРТ";
- ***Друга подія*** – видалення рисунка з поля вікна програми, якщо користувач виконує переміщення мишки по напису текстового повідомлення або встановлює мишку на напис;
- ***Третя подія*** – відновлення рисунка на полі вікна програми, якщо користувач мишку встановить на полі вікна або переміщувати по полю вікна.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

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

Для захисту результатів програмування *C* ++ програми до завдання № 1 після запуску у роботу виконуючого файлу необхідно виконати таке:

- нарисувати блок-схему алгоритму до роботи *C* ++ програми з завдання № 1;
- нарисувати блок-схему алгоритму по виконанню операторів функції, з обробки у програмі першої події;
- нарисувати блок-схему алгоритму по виконанню операторів функції, з обробки у програмі другої події;
- нарисувати блок-схему алгоритму по виконанню операторів функції, з обробки у програмі третьої події.

⬇⬇⬇⬇⬇⬇ Алгоритми для програмування ⬇⬇⬇⬇⬇⬇

Порядок дій і команд для виконання візуального програмування програми

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 завдання № 1", щоб на заголовку вікна форми замість назви *Form*1 з'явився напис "Лабораторна робота завдання № 1";
- Збережіть новий проект із порожньою формою з написом "Лабораторна робота завдання № 1" в заголовку вікна і така дія користувача активізує новий проект файлів в *C++ Builder* , яке запам'ятає шлях для швидкого збереження змін у проекті створюваної програми;
- Збереження файлів проекту виконайте на вказаний диск і папки з файлами *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 *Unit*1 на файл *U_Lab*1*.cpp*, а пропозицію до назви проекту *Project*1*.bpr* замініть на *P_Lab*1*.bpr*.

Крок 2. Встановлюємо на форму вікна компоненту *Panel*1:

- 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 маркерну рамку компоненти *Panel*1 (рис. 1-4).

Крок 3. Задаємо маркерною рамкою по горизонталі і вертикалі необхідний розмір компоненти *Panel*1 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 полі компоненти *Panel*1(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 *Panel*1, щоб він не накладався на виведений текст повідомлення:

- На компоненту *Panel*1 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 *Panel*1 і цей напис видалите клавішею *Delete* або *Backspace* з клавіатури**.**

Крок 6. Установимо на форму вікна кнопку *Button*1:

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

➢ На полі форми із сіткою зробіть щиглика мишкою, щоб встановилось зображення кнопки з назвою *Button*1 і охопленою маркерною рамкою.

Крок 7. На кнопці *Button*1 змінюємо назву на "СТАРТ":

➢ Клацніть по кнопці *Button*1 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 *Button*1 і змінить на назву "СТАРТ".

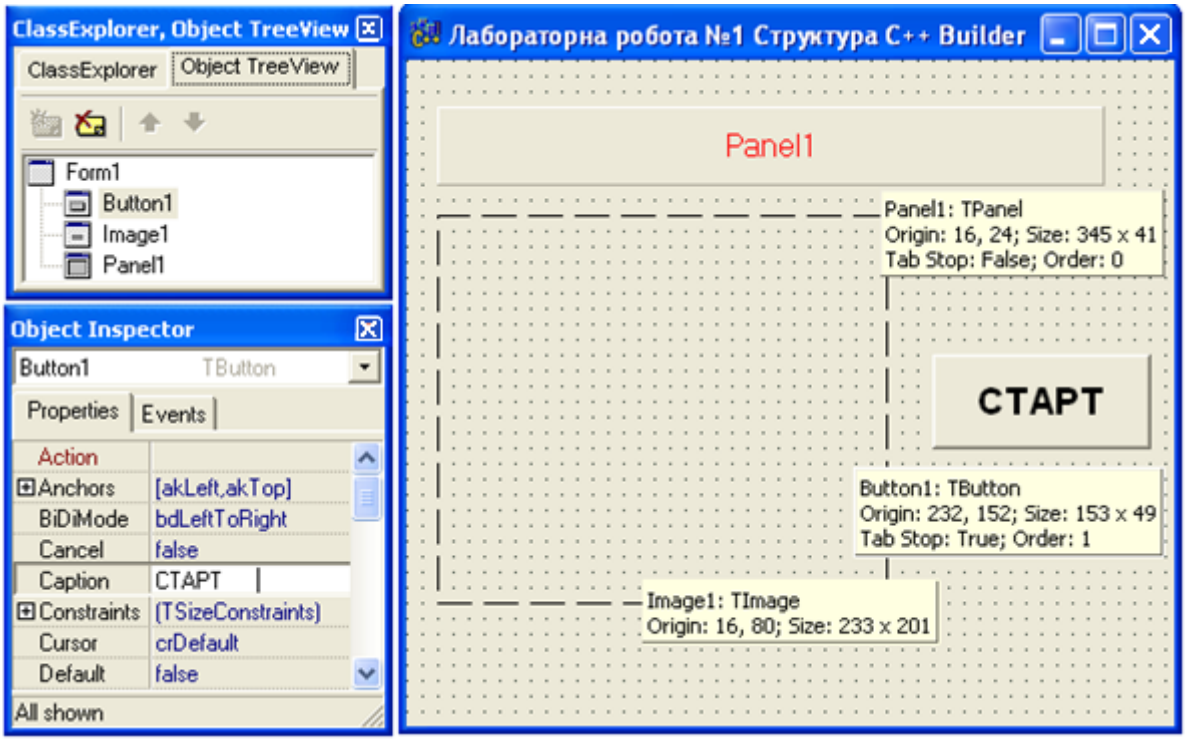


Рис. 1-3. Рис. 1-4. Розміщення на формі компонент та їхні розміри.

Крок 8. Визначаємо для кнопки "СТАРТ" першу подію - вивід напису з текстом «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 *Button*1*Click*;
  - активізується вікно редактора кодів, де буде показано 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 = " Створено першу програму для Windows у
   інтегрованому середовищі C++ Builder ";
   }
//---------------------------------------------------------------------------
```

Крок 10. Командами *Run/Run* чи на клавіатурі комп'ютера клавішею *F*9 виконуємо команду на 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* = , тоді 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 напис "Перша програма створена для *Windows* у інтегрованому середовищі *C++ Builder* " для цього необхідно:

- У вікні інспектора об'єктів зверху у списку компонент на формі 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 клавішею *F*9 на клавіатурі виконуємо компіляцію вихідних файлів і запускаємо на виконання *C++* програму для перевірки роботи.

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 у роботу:

```
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
TImage *Pict = new TImage(Form1);
Pict->AutoSize = true;

//---------- завантаження вихідного рисунка у  Image при відкритті вікна
//------рисунок Lab1_On.bmp помістити у 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 клавішею *F*9 на клавіатурі виконуємо компіляцію вихідних файлів і запускаємо на виконання *C++* програму для перевірки роботи операторів доданих 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* ":

- На формі виділить 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 написом:

```
//---------------------------------------------------------------------------
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. Якщо з'явилося повідомлення, що не оголошена змінна *On_Off*, тоді додаємо для змінної таке оголошення *int On_Off* = 0 на зовнішньому рівні програми (дивись крок № 25).

Крок 23. Для вікна форми *Form*1 задаємо подію – «відновлення рисунка на формі», якщо покажчик мишки буде розташовано на полі форми вікна програми:

- На сітці форми *Form*1 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 форми 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. Вказівки препроцесору і оголошення у файлі *U_Lab1.cpp* перевіряємо і вони 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. До завдання №1 перевіряємо роботу програми з лабораторної роботи шляхом виконання компіляції вихідних файлів і для цього виконуємо команди *Run/Run* або на клавіатурі натискається клавіша *F9*.

Контрольні запитання до завдання № 1

1. Пояснить загальні правила активізації (ініціалізації) основних файлів проекту, створюваної програми у *C++ Builder*.
2. На блок схемі алгоритму, яка команда виконується для обробки у програмі події «поява на формі вікна текстового повідомлення».
3. Пояснить алгоритм та початкові тексти *C++* програми з обробки у програмі події «вилучення рисунка з форми вікна програми».
4. Пояснить алгоритм та початкові тексти *C++* програми з обробки у програмі події «відновлення рисунка на формі вікна програми».
5. Покажіть команди, якими виконується налаштування властивостей компонент у *C++ Builder* для обробки подій мишки.

Завдання № 2 та алгоритми до теми

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

Ціль створення прикладної програми. Передбачається виконання таких навчальних завдань:

- 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.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Методика одержання коду C++ програми для виконання в *MS DOS*

У *Windows C++* програми працюють с *MS DOS* у спеціальному вікні і програма може виконуватися у віконному або в режимі повного екрану.[1] 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 [2]

#if умова

фрагмент коду програми

#endif

У цьому запису умова є цілочисловим заданим виразом. Якщо вираз умови повертає не нуль(істинну), тоді фрагмент коду, записаний між ***#if*** та ***#endif*** компілюється. Якщо умова помилкова тоді повертається нуль і текст коду пропускається препроцесором тобто не компілюється. Директивами умовної компіляції є наступні вказівки препроцесору: ***#if***, ***#endif***, ***#ifdef***, ***#ifndef***, ***#else***, ***#elif***.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Постановка завдання до програмування у лабораторній роботі

При виконанні завдання № 2 необхідно виконати таки два завдання:

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

У першій частині буде завдання А1 – необхідно компілювати повний вихідний файл, у якому містяться об’яви та визначення усіх нестандартних

функцій. Для цього в папці *LAB2* створюємо папку *LAB2_A1* і в неї зберігаємо усі файли проекту *P_Lab2_A1*.

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

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

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

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

Після запуску в роботу *C++* програм до завдань 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.

⬇⬇⬇⬇⬇⬇ Алгоритми для програмування ⬇⬇⬇⬇⬇⬇

Методика виконання завдання А1.

Порядок дій і команд при виконанні програмування завдання А1 для однофайлової компіляції файлу C++ 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 *Lab*2 і усередині створіть таки дві папки: *Lab*2_A1 та *Lab*2_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 *Project*1, а в заголовку вікна редактора кодів буде показуватися назва *Unit*1.*cpp* та на полі редактора кодів буде уставлено (показано) 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 змінюємо назву файлу *Unit*1 на файл *U_Lab*2_A1.*cpp*, а назву проекту *Project*1.*bpr* змінюємо на *P_Lab*2_A1.b*pr*.

Крок 2. Набираємо текст прикладної програми для завдання 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) можлива 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 для визначення суми 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 значення інтегралів \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 .

⇓⇓⇓⇓⇓⇓ Алгоритми для програмування ⇓⇓⇓⇓⇓⇓

Методика виконання завдання А2.

Порядок дій і команд при виконанні програмування завдання 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 вказаних дій у вікні редактора коду відкриється додаткова чиста сторінка з закладкою *File*1.*cpp*;
- На чисту сторінку вставляйте з буфера обміну опис функцій і текст та 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("\n Для ординат F(x) можлива 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* і 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 виконання її операторів 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 у папку *Lab*2_A2**:**

```
    //---------------------------------------------------------------------
    /* Функція з визначення суми 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_Lab2_*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* та в папку *Lab2_*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;            /* 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;
        }
//-----------------------------------------------------------------
```

Крок 6. Створюємо основний програмний модуль для багато-файлової компіляції методом вказівок препроцесору і зберігаємо у файл. З файлу *U_Lab*2_A1.*cpp* копіюйте в буфер обміну оператори функції *int main*( ) і вставте також за допомогою буфера обміну на додаткову сторінку у вікні редактора коду та збережіть у файл з назвою *U_Lab*2_A2.*cpp* та в папку *Lab*2_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 у 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:

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

⇓⇓⇓⇓⇓⇓ Алгоритми для програмування ⇓⇓⇓⇓⇓⇓

Методика виконання завдання 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 ***Lab*2_B** на диску ***D***:\ у папці ***Lab*2**;
- Скопіюйте файли Input_abE.cpp, Avt_step.cpp, Sum_ord.cpp із папки *Lab*2_A2 в папку *Lab*2_B.

Крок 2. Створюємо текст файлу *F_x*_B.*cpp* у папці *Lab*2_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 С++ 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 *Project2*, а в заголовку вікна редактора коду буде назва Unit1.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 замінюємо 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 файлу -> 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 кнопкою *F*9 на клавіатурі для файлу *U_Lab2_*B.*cpp* та перевірте правильність роботи *C++* програми у вікні при таких значеннях: ***a*** = 10; ***b*** = 45; ***E*** = 0,2.

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

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

Крок 9. Відновлюємо у файлу *F_x_B.cpp* наступні коментарі тобто не задаємо значення *METOD*

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

та перевіряємо правильність роботи *C++* програми:

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

Контрольні запитання до завдання № 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.

Завдання № 3 та алгоритми до теми

Команди редактора C++ Builder і структура файлів створюваних 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 у 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.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Особливості формування структури файлів до проекту 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.[1],[4]

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 таке інше, 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 *C++ Builder* сама викликає компоненти у 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, яки повинні використовуватись прикладною програмою. Розміри виконуючих модулів істотно скорочуються (приблизно у 10 разів менше) за

рахунок того, що велика частина програмних кодів міститься в цих пакетах. Інтегроване програмувальне середовище *C++ Builder* при виконанні команди *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 у режимі *Packages*(пакети) для вже створеного проекту *project.exe* потрібно виконати команди *Project / Build*.

У пакету часу виконання є бібліотеки *DLL,* які потрібні для роботи прикладної програми у середовищі *Windows* і їх можна з'ясувати за допомогою спеціальної програми *tdump.exe*, що входить до складу інтегрованого середовища *C++ Builder* і зберігається у каталозі *...\bin*. Ця програма працює у режимі *DOS* і тому необхідно перейти в *DOS* та увійти у 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 у зазначений текстовий файл *dump.txt*. У цьому файлі можна буде бачити список пакетів і бібліотек *DLL* , яки 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 у *Windows*. При проектуванні прикладної програми для *Windows* створюється проект (комплект файлів) і у таблиці № 3-1 показано список файлів, що створює інтегроване середовище *C++ Builder*.

Набір файлів у проекту прикладної C++ програми

Таблиця № 3-1.

Назва файлу	Призначення файлу
Головний файл проекту (*.cpp*)	*C++ Builder* створює файл *.cpp* для головної функції *WinMain*, ініціалізує програму і запускає у 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 вигляді чи у вигляді 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 робочого столу проекту (*.dsk*)	У цьому текстовому файлі *C++ Builder* зберігає інформацію про останній сеанс роботи з проектом, про відкриті вікна, їхніх розмірах та положенні. Завдяки цьому файлу у 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 виконуючий (*.exe*)	Файл програми, що виконується.
Об'єктний файл модуля (*.obj*)	Файл модуля (*.cpp*) після обробки вказівок препроцесору і компіляції, який редактором зв'язків компонується у остаточний файл, що буде виконуватися.
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 роботи у *C++ Builder*

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.

Редактор кодів *C++ Builder* є повноцінним програмним редактором. Цей редактор можна налаштовувати на різний стиль роботи, що вам більше 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 наклонна (*//*).

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 обрати 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 у іншій. Якщо якийсь з файлів більше не потрібний, то можна закрити цю сторінку з файлом, обравши 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, чи будуть символи, що вводяться, або вставлятися у 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 змінних і глобальних функцій, що містяться у файлі модуля чи форми у 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 інши 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 переміщується в межах іншого вікна. Вбудовування вікон дозволяє використовувати всю площу екрану дисплея. 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. Найбільш зручно вмонтувати вікно *ClassExplorer* у вікно "Інспектор Об'єктів" (*Object Inspector*). При цьому утворюються 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 вбудованих вікон можуть уміститися у заголовку вікна *Object Inspector* і тоді 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 у 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 не на багато зачасу 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 у 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 для змінної, до котрої буде присвоювання значення. Аналогічним чином можна одержати підказки по аргументах функцій і процедур.

❖ *Параметри функцій, процедур, методів*

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

❖ *Шаблони коду*

У помічнику *Code Insight* записано безліч шаблонів стандартних структур мови *C++*. Причому сам програміст зможе додавати чи видаляти ці шаблони. Виклик шаблону виконується натисканням клавіш "*Ctrl + J*". Зі

списку, що випадає, можна вибрати потрібний шаблон. Наприклад, якщо обраний шаблон циклу *for* , тоді у текст коду додається запис:

for(; ;) { }

і програмісту залишається лише тільки заповнити заголовок циклу і написати у дужках 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*, тоді при переміщенні курсору мишки 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, тоді 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 проекту *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, змінної, 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, тоді 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* і тоді можна зробити щиглик правою клавішею мишки і обрати у контекстному меню *Open File at Cursor* і відразу же у вікно редактора коду буде завантажений відповідний файл для перегляду оголошень функцій, констант, макросів і інших об'єктів.

❖ *Закладки для 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 інший, тоді у цьому випадку спочатку випливає контекстне меню і командою *New Edit Window* можна відкрити додаткове вікно редагування того ж файлу.

Тоді можна у одному вікні перейти до однієї закладки, а в іншім вікні до іншої закладки і одночасно працювати з обома фрагментами коду.

⇓⇓⇓⇓⇓⇓ Інформація для користувача ⇓⇓⇓⇓⇓⇓

Постановка завдань до програмування у лабораторній роботі

Необхідно для *Windows* створити прикладні програми відповідно до таких наступних завдань:

Завдання А. Прикладна *C++* програма повинна формувати зображення до залежності функції *F(x)* і для завданих значень аргументу ***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 уведеннях інших коефіцієнтів до функції *F*(*x*) і щигликах на кнопці "ПУСК" програма виконує нові розрахунки значення *F*(*x*). *Завдання **А** програмується і виконується по двох варіантах проектування*:

- Для першого варіанта виконуємо завдання **А1**, у якому створюється проект файлів у вигляді пакета часу виконання для одержання виконуючого файлу прикладної *C++* програми;
- По другому варіанту виконуємо завдання **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.

Завдання Б. Необхідно створити для *Windows* прикладну *C++* програму у вікні, якої буде кнопка з назвою "Розрахувати" і після щиглика мишкою по кнопці 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.

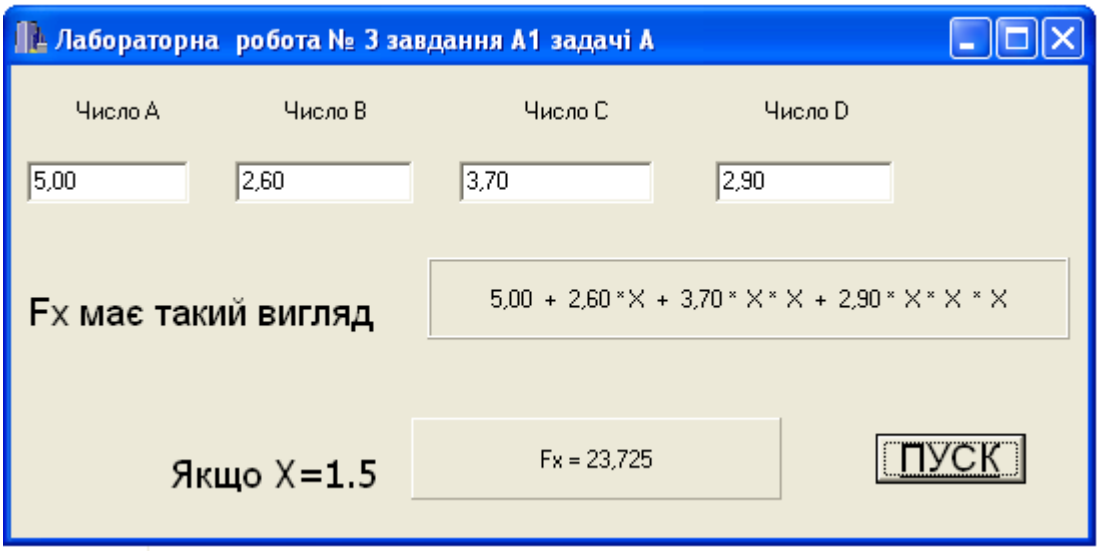


Рис. 3-1. Результат роботи *C++* програми при виконанні завдань А1 та А2.

↓↓↓↓↓↓ Інформація для користувача ↓↓↓↓↓↓

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

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

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

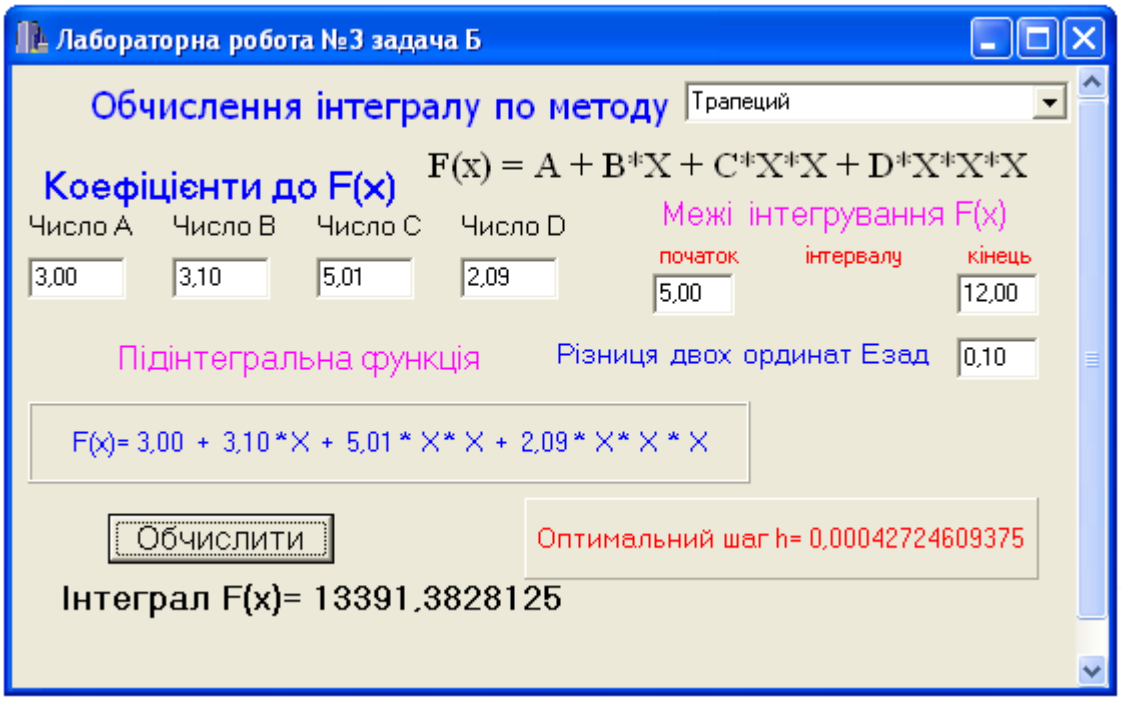


Рис. 3-2. Результат роботи *C++* програми при виконанні завдання Б.

⇓⇓⇓⇓⇓⇓ Алгоритми для програмування ⇓⇓⇓⇓⇓⇓

Методика виконання завдання А1.

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

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

- Виконайте у 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 завдання А1", щоби у заголовку вікна форми замінилась 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:

- Збережіть 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_Lab*3_A1.bpr**.**

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 для записів підказок - який коефіцієнт варто вводити;
- Під компонентами *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.

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

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

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**".

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".

Рис. 3-4. Вигляд форми для завдання А1 перед виконанням компіляції файлів.

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 повинно 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 ” у шаблон функції *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 це показано на рис. 3-4.

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

⇓⇓⇓⇓⇓⇓ Алгоритми для програмування ⇓⇓⇓⇓⇓⇓

Порядок дій і команд при виконанні програмування завдання А2 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"* та 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 *A*1, 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;
- Виконуємо визначення списку динамічних бібліотек, яки використовує виконуючий файл *P_ Lab*3_A_2*.exe* і для цього закриємо вікна *C++ Builder;*
- Входимо у каталог *Lab3*_A_2, де зберігається файл *P_ Lab*3_A_2*.exe*;
- Набераємо у командному рядку tdump P_ Lab3_A_2.exe > dump.txt

і натискаємо *Enter*. У список файлів буде доданий текстовий файл *dump.txt* у якому можна побачити дані виконуючого файлу з пакету у часі виконання:

- ❖ Переглянемо текст файлу *dump.txt* та встановимо курсор у розділ *Imports*, де буде показано список застосованих динамічних бібліотек.

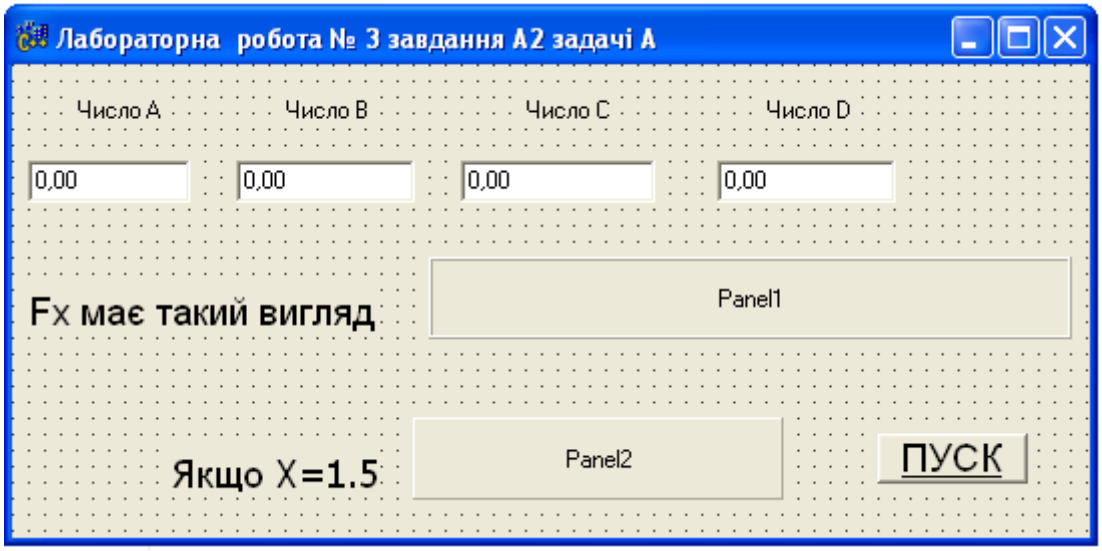


Рис. 3-5. Вигляд форми з завдання А2 до початку компіляції файлів.

⇓⇓⇓⇓⇓⇓ Алгоритми для програмування ⇓⇓⇓⇓⇓⇓

Методика виконання завдання Б.

Послідовність команд при виконанні програмування завдання Б по створенню проекту програми з розрахунком функції F(x)

Крок 1. Установить на форму компоненту *Label*1 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 назв 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 обчислення 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 *E*зад" та задайте *Font*;

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

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

Крок 10. Установить 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.

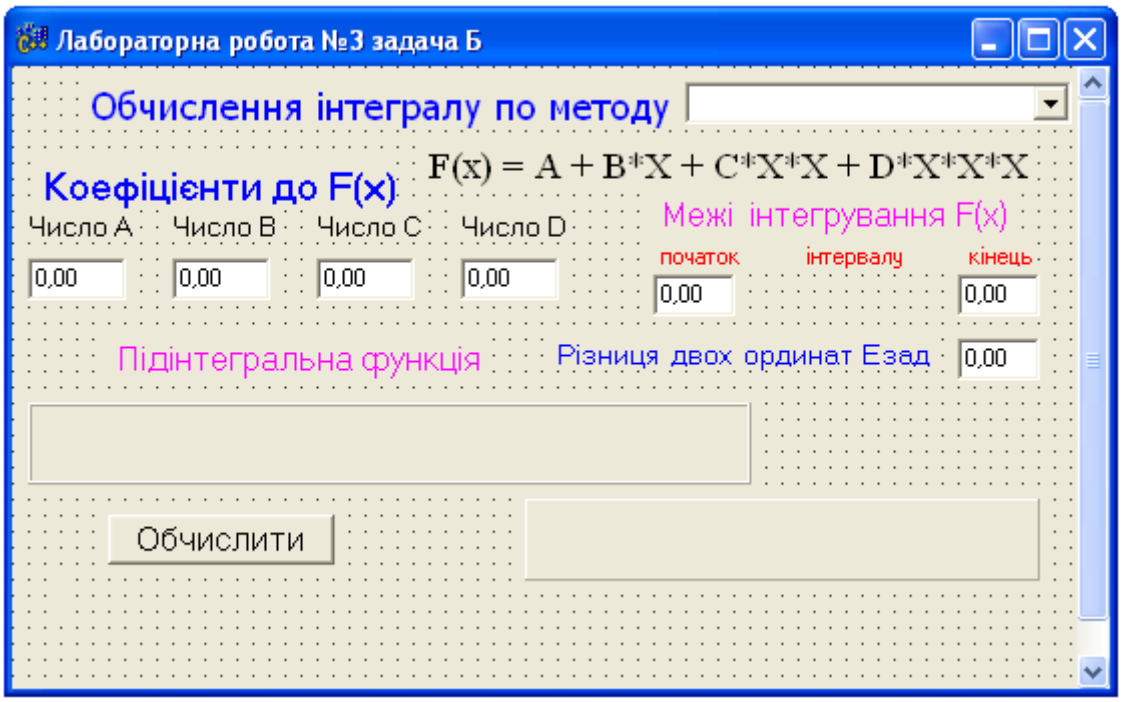


Рис. 3-6. Розташування компонент на формі *C++* програми до завдання Б.

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":

- Розташуйте на формі напис "Лабораторна робота № 3 завдання Б " як це показано на рис. 3-6.

Крок 14. У шаблон файлу *U_ Lab*3_Б.*cpp* потрібно додати оператори, які необхідні для обчислення значення інтеграла у 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 функції 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 інтегралу 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 можна побачити на рис. 3-2.

Контрольні запитання до завдання № 3

1. Які пакети (*packages*) до файлів може створити *C++ Builder*.
2. Пояснить, як у *C++ Builder* утворюються файли реалізації модуля форми (*.cpp*) та об'єктний файл модуля (*.obj*).
3. Які файли з розширенням (*.h*) застосовуються у *C++* програмах до завдань А та Б з лабораторної роботи № 3.
4. У *C++ Builder* які елементи показуються у рядку стану редактора коду.
5. Пояснить алгоритм обробки події для кнопки *Button*1 у *C++* програмі .

Завдання № 4 та алгоритми до теми

Техніка використання і контролю обробки структурних даних в C++ програмі

Ціль створення прикладної програми. В *C++* програмах обробляється і використовується багато інформації яка має відповідну структуру даних типу (*struct*) і тому основне завдання у даній лабораторній роботі визначено тім, що створювана у *C++ Builder* програма повинна мати структуру даних і сприяти вивченню наступних завдань:

- Техніки і методики використання в прикладній *C++* програмі даних типу «структура»;
- Правил та команд для контролю обробки даних структурного типу (*struct*) за допомогою застосування програмного відладчіка *C++ Builder* ;
- Алгоритму з пошуку помилок програмування у 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, які використовує прикладна *C++* програма.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Структури даних типу *struct* в C++ програмах

Структури - це створюваний програмістом складений тип даних, що будується з використанням інших типів даних. Структури являють собою об'єднаний загальним ім'ям набір даних різних типів. Саме цим відрізняються структури даних і в них можуть зберігатися дані різних типів, наприклад, у масивах зберігаються дані одного типу.[2] Значення даних до змінних та іншої

інформації у структурах називаються елементами або полями структури. Формат оголошення структури даних у *C++* програмі має такий вигляд

```
struct <ярлик структури> {
    тип даних змінна1;
    тип даних змінна2;
    ……..;
}<ім'я структури даних>;
```

Ярлик структури дозволяє створювати і повідомляти у програмі нові змінні структурного типу з зазначеним ярликом. У оголошенні структури даних обов'язковим елементом повинен бути заданий "ярлик структури " або "ім'я_ структури даних ". 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;
```

У програмуванні на мові *C++* структури даних наділені додатковими можливостями:

– перша – це можна як елемент структури вказувати визначення функції

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

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

Дозволяється застосовувати специфікатори *public* (відкритий) та *private* (закритий). Закриті елементи структури можуть бути доступні тільки для функцій, як 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 різними функціями при роботі програми зі структурою, а функція *F( )* і дані *a* та *b* відповідно закриті і ними може користуватися тільки функція *F( ).* Структура може мати елементи з побітовими полями, тобто елемент може займати завдану кількість біт

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

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Постановка завдання до програмування програми

Необхідно створити за допомогою *Consol Wizard* прикладну програму з виконанням таких дій:

- У структуру даних задаються (вводяться з клавіатури) значення координат для трьох позначень (крапок) на площині;
- По значенням до координат позначень точки, записаних в структуру даних, визначаються масштабні коефіцієнти і з урахуванням їх значень будується поле площини для розміщення позначень (крапок) на екрані дисплея комп'ютера;
- По значенням до координат позначень (крапок), записаних у структуру даних, розміщуються позначення до крапок на площину зеленого кольору з урахуванням визначених відповідних масштабних коефіцієнтів;
- За допомогою програмного відладчіка *C++ Builder* виконується по шагах перевірка правильності виконання обробки структури даних операторами в функціях програми;
- За допомогою програмного відладчіка *C++ Builder* виконується контроль значень розрахованих масштабних коефіцієнтів при розміщенні позначень (крапок) на площину зеленого кольору.[1]

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

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

Після запуску у роботу *C++* програми для захисту результатів програмування до даної лабораторної роботи необхідно виконати таке:

– нарисувати блок-схему алгоритму по виконанню операторів у *C++* програмі лабораторної роботи № 4;

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

⬇⬇⬇⬇⬇⬇ Алгоритми для програмування ⬇⬇⬇⬇⬇⬇

Порядок дій і команд для виконання програмування програми

Крок 1. На диску *D*:\ створіть папку *Lab_4*.

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

Крок 3. 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 ");
   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 площини для рисунка*/
  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;
}
//---------------------------------------------------------------------------
```

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Види команд *C++ Builder* для налагодження та пошуку помилок у текстах 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" функціональною клавішею *F9*. У цьому випадку виконується компіляція початкового тексту *C++* програми, її компонування та створюється виконуючий файл (*.exe*) і він запускається на виконання. Створення (*.exe*) файлу 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 компілюються 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 у 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 змінної і на екрані з'являється числове значення цієї змінної. У такий спосіб можна подивитися значення змінних *C++* програми у даний момент. Деякі змінні не вдається спостерігати, тому що компілятор який оптимізував код, видалив їх з результуючого коду. Цей інструмент дозволяє контролювати тільки окремі змінні;

- **Вікно спостережень (***Watch List***).** У *C++ Builder* це вікно є програмним влаштованим інструментом, який забезпечує можливість бачити значення декількох змінних 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 змінної та натиснути *Ctrl+F*5. При цьому вікно спостереження автоматично відкриється і у ньому з'явиться ім'я змінної і її значення (значення змінної буде видно тільки при зупинці виконання програми і переходу у *C++ Builder*). Потім можна

підвести курсор до іншої змінної, знову натиснути *Ctr+F*5 і у вікні спостережень з'явиться новий рядок. Більш того можна виділити курсором якйсь вираз, натиснути *Ctr+F*5 і у вікні спостереження побачити значення цього виразу. Іноді змінні не вдається спостерігати, тому що компілятор з оптимізацією видалив їх з результуючого коду і помістив відповідні значення у системні регістри. Це прискорює виконання обчислень у програмі, заощаджує пам'ять, але перешкоджає спостереженню змінних у процесі налагодження програмного модуля. У таких випадках можна оголосити відповідну змінну з ключовим словом *volatile*. Наприклад, *volatile int x*1 .

Специфікатор *volatile* показує компілятору, що дану змінну не можна зберігати у регістрі мікропроцесора. 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 програми ці варіанти потрібно зняти, щоб не знижувати ефективність роботи прикладної програми*. Якщо перейти у вікно спостережень і там клацнути правою клавішею мишки та у меню обрати 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. У полі редагування *Expression* можна записати ім'я будь якої змінної чи будь який 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, тоді виділить його у вікні і зробіть на ньому подвійний щиглик. Відкриється вікно *Watch Properties* із завантаженим у нього виразом 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 у полі *Expression* вираз *++A*. Якщо індикатор *Allow Side Effects* виключений (він виключений за замовчуванням), тоді у вікні спостережень поруч з виразом *++A* покажеться текст:"*Side effects are not allowed*" (побічні ефекти заборонені). Радіокнопки у 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, щоби відображати деяку цілу змінну один раз у десятковому вигляді, а іншого разу в шістьнадцатирічному вигляді. Список, що випадає, у полі редагування *Expression* дозволяє обірати вирази з тих, котрі використовувалися раніше і при необхідності його відредагувати. Наприклад, якщо потрібно вивести значення

*Form1->Label*1*->Caption, Form*2*->Label*2*->Caption, Form*3*->Label*3*->Caption*

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

❖ **<u>Вікно оцінки і модифікації</u> (***Evaluate/Modify***).** В *C++ Builder* це вікно є програмним влаштованим інструментом, який забезпечує можливість в процесі налагодження не тільки спостерігати, але і змінювати значення змінних. Можна зробити це вікно видимим 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 змінної чи виразу. Після цього потрібно клацнути

кнопку *Evaluate* і в полі *Result* з'явиться поточне значення. Якщо в полі *Expression* задане ім'я змінної, то стає доступною кнопка *Modify* що дозволяє змінити значення, тобто можна втрутитися у процес виконання програми шляхом завдання нового значення в полі *New value* і потім необхідно натиснути кнопку *Modify*. У результаті значення змінної в програмі зміниться, що можна бачити у полі *Result*. Також зміниться значення і у вікні спостережень *Watch List* якщо в нього перейти.

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

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

Команди для проходу фрагменту програми по кроках. Таблиця 4-1.

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 знаходиться за викликом функції.
Show Execution Point (Показати крапку виконання)	-	Розміщується курсор на оператор, що буде виконуватися наступним.

❖ **Крапки переривання (*breakpoint*).** В *C++ Builder* ці крапки є програмним влаштованим і могутнім інструментом для налагодження програми за рахунок розміщення крапок переривань у операторах і функціях програми. Щоб ввести просту (безумовну) крапку переривання, досить у вікні "Редактор коду" клацнути мишкою на смужці зліва від коду необхідного рядка програми. Рядок програми офарблюється у 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(зняти позначення), досить клацнути мишкою на червоній крапці зліва від коду, відзначеного крапкою переривання. *Крапки переривання можна встановлювати тільки на виконуваних операторах*. Якщо, наприклад, установити переривання на рядку оголошення змінної, то в момент запуску програми у 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 мікропроцесора.

❖ Внутрішня програма ***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 у момент завдання крапки переривання. Поле *Condition*(умова) дозволяє ввести де які необхідні умови. Переривання буде відбуватися тільки у тому випадку, якщо значення умови стане дорівнювати *true*. Наприклад, можна вказати умову зупинки в такому вигляді ***i* = = 10**.

Контрольні запитання до завдання № 4

1.Пояснить на прикладах у яких випадках потрібно у програмі використовувати дані типу "структура".

2.Запишіть можливі варіанти оголошення структури даних у програмі.

3.Пояснить правила запису і зчитування даних змінної у полях структури коли завдано ім'я структури або покажчик на структуру.

4. Яка структура даних використовуються у *C++* програмі лабораторної роботи(введення значень та використання значень змінних).

5. Покажіть у програмі лабораторної роботи варіанти використання крапок (*breakpoint*).

Завдання № 5 та алгоритми до теми

Імітаційне моделювання алгоритму з відображення інформації на дисплеях мікропроцесорного приладу ІТМ-11

Ціль створення прикладної програми. Завдання на дану тему передбачає виконання імітаційного моделювання роботи мікропроцесорного приладу і для цього потрібно:

- Розробити прикладну *C++* програму яка буде імітувати показання числових значень даних контрольованого параметру на дисплеях у зображенні панелі мікропроцесорного приладу *ІТМ*-11;
- Вивчення алгоритму по створенню комп'ютерної імітаційної моделі з роботи мікропроцесорного приладу *ІТМ*-11, використовуючи стандартні компоненти інтегрованого програмувального середовища *C++ Builder*.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Відомості до панелі приладу ІТМ-11

У системах регулювання параметрів технологічних процесів для показу значень технологічного параметру використовуються прилади *ІТМ*-11(рис. 5.1) які мають також і функцію сигналізації та інтерфейсу RS-485 для з'єднання сигналами між приладами системи контролю параметрів технологічного процесу.

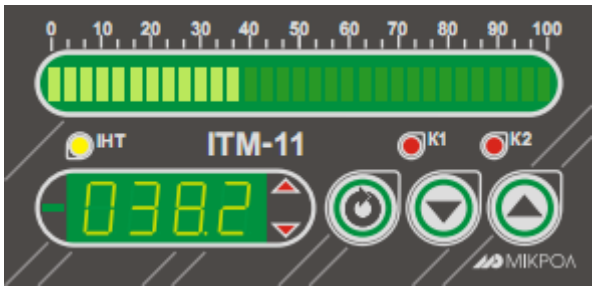


Рис 5.1. Передня панель мікропроцесорного приладу *ІТМ*-11.

Індикатор технологічний мікропроцесорний *ІТМ*-11:

- ✓ універсальні багатофункціональні та одноканальні індикатори значень технологічних параметрів;
- ✓ модифікації *ІТМ*-11 є таки: горизонтальне виконання з цифровим дисплеєм і лінійним сегментним індикатором (31 сегмент) та конструкції приладів *ІТМ* з вертикальними дисплеями.

Область застосування:

- Системи цифрової і лінійної індикації значень технологічних параметрів;

- Двопозиційне, трипозиційне і багатопозиційне регулювання температури, тиску, витрат, рівня та інших вимірюваних параметрів;
- Дистанційні пристрої зв'язку з об'єктом керування;
- Територіально розподілені і локальні системи управління;
- На місцевих щитах та пультах керування, мнемосхемах і т. п.

Функціональні можливості *ITM*-11:

- Робота з уніфікованими сигналами, термо вимірювачами опору та термопарами;
- Кожен аналоговий вхід може бути налаштований на підключення будь-якого стандартного типу вимірювача;
- Індикація параметрів в технологічних одиницях на цифрових і лінійних (0-100%) індикаторах;
- Цифрове (автоматичне і ручне) калібрування від початку до кінця шкали з діапазону вимірювання;
- Вибір методу лінійної індикації (сегмент, гістограма);
- Завдання значення для сигналізації відхилення контрольованого параметру від налаштувань мінімум і максимум до світлодіодів на передній панелі приладу;
- Можливі таки типи технологічної сигналізації: без запам'ятовування спрацьовування, із запам'ятовуванням спрацьовування і квітуванням;
- Вхідний цифровий фільтр аналогових входів;
- Витяг квадратного кореня з сигналу вимірювання витрати по перепаду тиску;
- Функція вимірювання інтегральної витрати;
- Лінеаризація вхідного сигналу (по 16 точках):
- Конфігурування дискретних вихідних команд пристроями - транзистор, реле, оптосімистор або твердотільне реле;
- Програмована логіка роботи вихідних пристроїв: більше (*MAX*), менше (*MIN*), в зоні (*MIN-MAX*), поза зоною **(***MIN-MAX***);**
- Аналоговий вихід для ретрансмісії вхідних аналогових сигналів;

- Архівація даних у енергонезалежну пам'ять;
- Збереження параметрів налаштувань при відключенні живлення;
- Захист від несанкціонованої зміни налаштованих параметрів;
- Гальванічно окремий інтерфейс *RS*-485 по протоколу *ModBus RTU* (збір інформації та конфігурації).

↓↓↓↓↓↓ Інформація для користувача ↓↓↓↓↓↓

Постановка завдання до програмування програми

Необхідно створити *C++* програму для роботи у *Windows* і у який буде: вікно з зображенням панелі приладу *ITM*-11 горизонтального розташування, з цифровим дисплеєм та лінійним сегментним індикатором.

Для створення імітаційної прикладної програми на формі необхідно використовувати [3] таки компоненти *C++ Builder* :

Image – компонента із вкладки *Additional* для завантаження рисунків і відображення їх на формі;

Timer – компонента із вкладки *System*, необхідна для створення анімованого зображення;

Label – компонента із вкладки *Standard* , необхідна для відображення тексту;

TrackBar – компонента із вкладки *Win*32, необхідна для завдання різних значень контрольованого параметру приладом *ITM*-11;

Edit – компонента із вкладки *Standard*, для забезпечення введення мінімального та максимального значення для контрольованого параметру;

Chart – компонента із вкладки *Additional* для графічного відображення змінювання значень контрольованого параметру.

↓↓↓↓↓↓ Інформація для користувача ↓↓↓↓↓↓

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

Для захисту результатів програмування після запуску у роботу *C++* програми до завдання № 5 необхідно виконати таке:

– нарисувати блок-схему алгоритму з роботи програми до завдання № 5;
– нарисувати блок-схему алгоритму по виконанню операторів функцій, з обробки подій у програмі.

⬇⬇⬇⬇⬇⬇ Алгоритми для програмування ⬇⬇⬇⬇⬇⬇

<u>Порядок дій і команд</u> для виконання програмування завдання

<u>Крок 1.</u> Створюємо новий проект *C++* та змінюємо назву форми проекту через властивість *Caption*. Замінюємо назву *Form*1 на назву «Емулятор ІТМ-11». Зберігаємо зміни у папку з картинками із назвою *ITM*-11 яку розміщуємо в папку, де знаходяться файли збереженого проекту.

<u>Крок 2.</u> Розміщуємо на формі програми таки необхідні компоненти:

Timer1 – Interval = 100; Timer2 - Interval = 100; Timer3 – Interval = 100.

Image1,Image2, Image3, Image4, Image5, Image6, Image7, Image8 (*для кожного з Image підбираємо необхідні розміри і позиції, усі зміни параметрів робимо через Object Inspector - Properties*).

Image1 – AutoSize – true; Picture – завантажуєм ITM-11.bmp; Left =0;Top=0;

Image2 - Left =62; Top=88; Height=28; Width=20; Picture – завантажуємо ITM-11\Digits\0.bmp

Image3 - Left =84; Top=88; Height=28; Width=20; Picture – завантажуємо ITM-11\Digits\0d.bmp

Image4 - Left =106; Top=88; Height=28; Width=20; Picture – завантажуємо ITM-11\Digits\0.bmp

Image5 - Stretch=true; Left=22;Top=28; Height=23; Width=251; Picture – завантажуємо ITM-11\Linear\0.bmp

Image6 - Transparent=true; Left =133; Top=106; Height=17; Width=17; Picture – завантажуємо ITM-11\at.bmp

Image7 – Transparent=true; Left =133; Top=87; Height=17; Width=17; Picture – завантажуємо ITM-11\at2.bmp

Image8 - Left =38; Top=88; Height=28; Width=20; Picture – завантажуємо ITM-11\Digits\0.bmp

TrackBar1 – Min=0, Max=100; SelStart=30; SelEnd=70;

Label1 – змінюємо параметр Font, настроюємо бажаний шрифт і розмір шрифту.

Label2 – змінюємо параметр *Caption* на напис «Контроль параметрів».

Розміщуємо на формі компоненти *Edit*1 та *Edit*2 і змінюємо параметри для них:

Edit1 – Text – min;

Edit2 – Text – max.

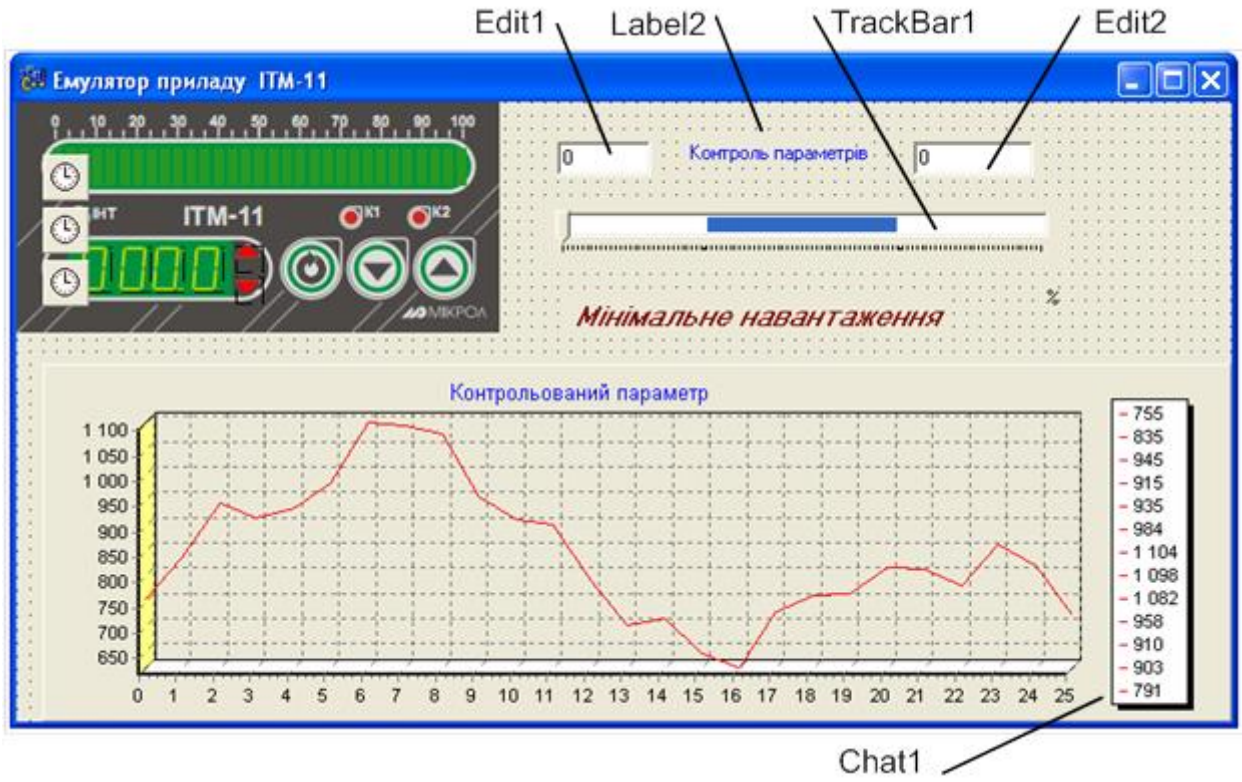


Рис. 5.2. Форма вікна програми із розміщеними на ній компонентами.

Додаємо[3] компоненту *Chart* і на формі бачимо область для побудови графіку, робимо двічі щелчок по компоненті лівою клавішою мишки та обираємо *Add*… із вкладки *Series* і з'являється напис *Series*1. Переходимо на вкладку *Titles* і створюємо напис для графіку «Контрольований параметр». Після того як усі компоненти розставлені, змінюємо властивості форми: *Form*1 – *AutoSize* =

true. Форма з усіма необхідними компонентами повинна мати вигляд, як показано на рисунку 5.2.

Крок 3. Заповнюємо лістинг програми, створюючи необхідні події і додаючи вказівки препроцесорів. Усі події створюємо або подвійним щелчком мишки по компоненті, або через *Object Inspector – Events*.

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include <math.h>
#include "Unit1.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
float k,P[100],x1,ch;
int x,c,c2,c3,c4,i,min=0,max=10,i2,j,l;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
 Form1->DoubleBuffered=true;
 Image6->Visible=false;
 Image7->Visible=false;
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer1Timer(TObject *Sender)
{
   k=(max-min)*0.01;
```

```
 x1=min+(TrackBar1->Position)*k;
 x=min+(TrackBar1->Position)*k;
 c=(x-x%100)/100;                    //перша цифра числа
 c2=((x-x%10)-(x-x%100))/10;    //друга цифра
 c3=x%10;                            //третя цифра
 ch=ceil((x1-x)*10);
 c4=ch;                              //цифра після коми
 l=(TrackBar1->Position*31)/100; //відповідає за заповнення сегментів
 if (max<10)
{Image8->Picture->LoadFromFile("ITM-11/Digits/0.bmp") ;
 Image2->Picture->LoadFromFile("ITM-11/Digits/0.bmp") ;
 Image3->Picture->LoadFromFile("ITM-11/Digits/"+IntToStr(c3)+"d.bmp") ;
 Image4->Picture->LoadFromFile("ITM-11/Digits/"+IntToStr(c4)+".bmp");
 Image5->Picture->LoadFromFile("ITM-11/Linear/"+IntToStr(l)+".bmp");
}
if (max<=99 )
{
 Image8->Picture->LoadFromFile("ITM-11/Digits/0.bmp") ;
 Image2->Picture->LoadFromFile("ITM-11/Digits/"+IntToStr(c2)+".bmp") ;
 Image3->Picture->LoadFromFile("ITM-11/Digits/"+IntToStr(c3)+"d.bmp") ;
 Image4->Picture->LoadFromFile("ITM-11/Digits/"+IntToStr(c4)+".bmp");
 Image5->Picture->LoadFromFile("ITM-11/Linear/"+IntToStr(l)+".bmp");
 }
  if (max<1000 )
{
 Image8->Picture->LoadFromFile("ITM-11/Digits/"+IntToStr(c)+".bmp") ;
 Image2->Picture->LoadFromFile("ITM-11/Digits/"+IntToStr(c2)+".bmp") ;
 Image3->Picture->LoadFromFile("ITM-11/Digits/"+IntToStr(c3)+"d.bmp") ;
```

```
 Image4->Picture->LoadFromFile("ITM-11/Digits/"+IntToStr(c4)+".bmp");
 Image5->Picture->LoadFromFile("ITM-11/Linear/"+IntToStr(l)+".bmp");
}
if (TrackBar1->Position<=30)
{ Timer2->Enabled=true;Timer3->Enabled=false; Image7->Visible=false;
 Label1->Caption="  Мінімальне навантаження ";
 Label1->Font->Color=clGreen;}
if (TrackBar1->Position>=70)
{Timer3->Enabled=true;Timer2->Enabled=false; Image6->Visible=false;
 Label1->Caption="  Максимальне навантаження ";
 Label1->Font->Color=clRed;}
if (TrackBar1->Position>30 && TrackBar1->Position<70)
{i=0;
 Label1->Caption="  Робочий режим ";
 Image6->Visible=false;
 Image7->Visible=false;
 Timer2->Enabled=false;
 Timer3->Enabled=false;
 Label1->Font->Color=clBlue;}
 //*** Рисування графіка функції***//
  P[100]= x+ch/10;
  for (i2=0;i2<100;i2++)
 {P[i2]=P[i2+1];}
  Series1->Clear();
  for (j=0;j<100;j++)
  {
    Series1->AddY(P[j]);
  }
```

```
 }
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer2Timer(TObject *Sender)
{
  i++;
  if (i==1)
  {Image6->Visible=true;}
  if(i==2)
  {Image6->Visible=false;i=0;}
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Timer3Timer(TObject *Sender)
{
  i++;
  if (i==1)
  {Image7->Visible=true;}
  if(i==2)
  {Image7->Visible=false;i=0;}
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Change(TObject *Sender)
{
 min=StrToInt(Edit1->Text);
}
//---------------------------------------------------------------------------

void __fastcall TForm1::Edit2Change(TObject *Sender)
{
```

```
 max=StrToInt(Edit2->Text);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit2Click(TObject *Sender)
{
 Edit2->Text=IntToStr(max);
}
//---------------------------------------------------------------------------
void __fastcall TForm1::Edit1Click(TObject *Sender)
{
Edit1->Text=IntToStr(min);
}
//---------------------------------------------------------------------------
```

Готова робоча *C++* програма до завдання № 5 повинна мати вікно, як показано на рисунку 5.3.

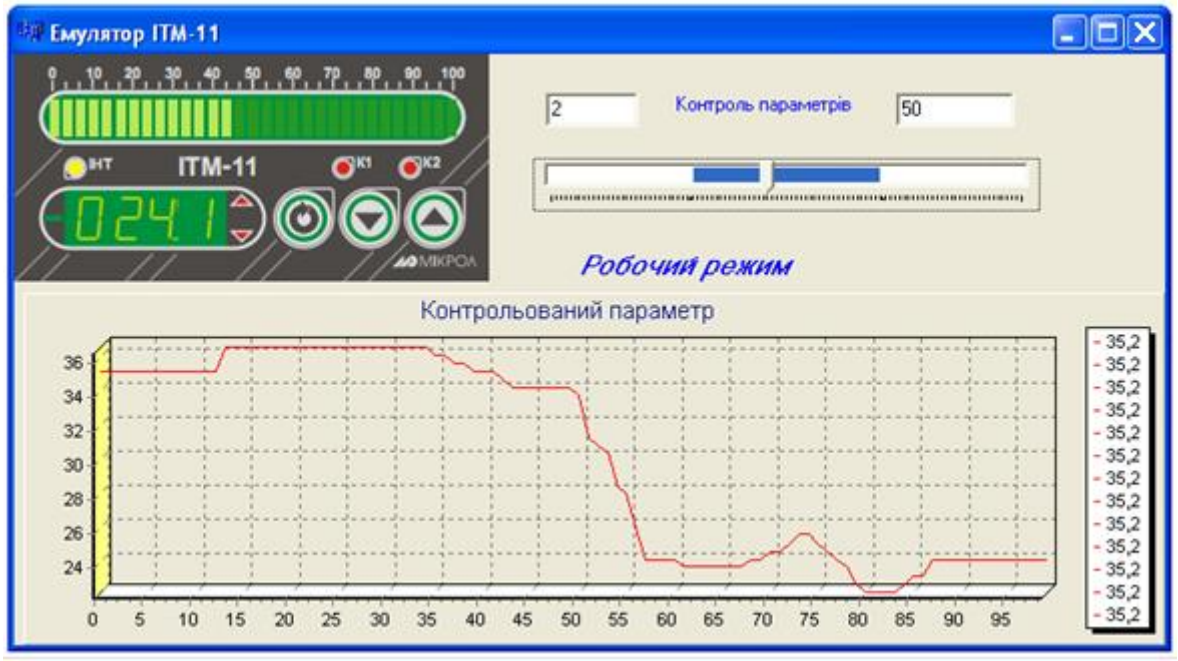


Рис. 5.3. Вікно програми з демонстрації роботи дисплеїв приладу *ІТМ*-11.

Контрольні запитання до завдання № 5

1. Пояснить загальні правила активізації (ініціалізації) основних файлів проекту, створюваної програми у *C++ Builder*.

2. Які команди на блок-схемі алгоритму виконуються для обробки події з появи графіка на полі вікна.

3. Пояснить алгоритм та початкові тексти *C++* програми по обробці події з показу числових значень на полі цифрового дисплея.

4. Пояснить алгоритм та початкові тексти *C++* програми по обробці події до лінійного сегментного індикатора.

5. Покажіть команди *C++ Builder*, якими можна виконувати налаштування компонент для обробки подій мишки.

Завдання № 6 та алгоритми до теми

Алгоритми використання функцій для обробки команд мишки до подій у прикладній C++ програмі

Ціль створення прикладної програми. Інтегроване програмувальне середовище *C++ Builder* має відповідний набір функцій для обробки команд маніпулятора «МИШКА» до подій які виникають в прикладних програмах і тому завдання до даної лабораторної 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 у проекті файлів прикладної *C++* програми;
- Команд по управлінню замінами зображень курсору мишки при її зміщеннях по полю вікна прикладної *C++* програми.

⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇

Загальні зведення о функціях для обробки команд мишки і подій у 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 у прикладних програмах інтегроване середовище *C++ Builder* має набір функцій для обробки визначених користувачем команд до подій при роботі створеної прикладної програми. Обробка команд мишки і подій виконується за допомогою бібліотечних функцій,[3] які наведені у табл. 6.1

Функції для обробки команд мишки та їх призначення.

Таблиця 6.1

Функція з обробки команд мишки	Опис призначення функції при обробках команд до подій мишки
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 натиснута кнопка зі стрілкою;
- Користувач натиснув клавішу пробіл або коли кнопка була 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' підчоркнуто , тоді натискання

користувачем клавіш *Alt* + П викликає виконання події *OnClick* в цьому індикаторі;

- Програма прикладна встановила в *true* властивість *Checked* радіокнопки *RadioButton*;
- Програма виконала заміну у індикатора властивості *Checked***;**
- Виконано метод *Click* для елемента у 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* у процесі проектування, визначає, яка з розміщених компонент буде у фокусі в перший момент при виконанні програми. У 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, у який розміщувались керуючи елементи на формі. У першого елемента на формі значення *TabOrder*, дорівнює **0**, другому відповідно **1** і так далі. Значення *TabOrder*, нуль, визначається, ще при перший появі форми на екрані і у фокусі буде перша компонента, яка була встановлена на форму (якщо для форми не встановлено властивість *ActiveControl*).

Всім цім функціям при обробках подій, які пов’язані з маніпуляціями мишки передається параметр *Sender* типу *Tobject*. Цей параметр є вказником на компоненту до якої виникла подія. Параметр *Sender* можна використовувати для позначення причини події. Наприклад, записати такий оператор

if (Sender = = Image1)

……;

Окрім параметру *Sende**r*** у функції з обробки подій *OnMouseDown* та *OnMouseUp* передаються параметри, які 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* у 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 з обробки 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 *Shift* типу *TshiftState* визначає, які додаткові клавіші на клавіатурі були натиснути у момент події – натискання кнопки мишки.

В усіх подіях, пов’язаних з командами мишки також передаються для курсору мишки координати *X* та *Y*. Дані параметри визначають координати курсору у клієнтській площині компоненти. Це дає можливість визначати різну реакцію програми у залежності від розміщення 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 може створювати зображення у вигляді бітових матріц, піктограмок та зображень до

курсорів і зберігати одночасно у файл та у файл ресурсів програми. Цим відрізняється редактор *Image Editor* від других більших графічних редакторів.

Робота у 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

У прикладній *C++* програмі, яка розроблена у *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
{ //--- виконуются довгі операії
   ……
catch (…..)
```

```
{
   Screen->Cursor = crDefault; //--відновлення курсору
    throw
}
   Screen->Cursor = crDefault;
```

При успішному або аварійному закінченні довгих операцій у програмі зображення курсору у 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 підключається 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;
```

У такому випадку необхідно в функцію з обробки подій на формі *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 вказника курсору. Точка вказника курсору – це та точка зображення нестандартного курсору координати якої (*X* та *Y*) передаються функції з обробки подій мишки.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

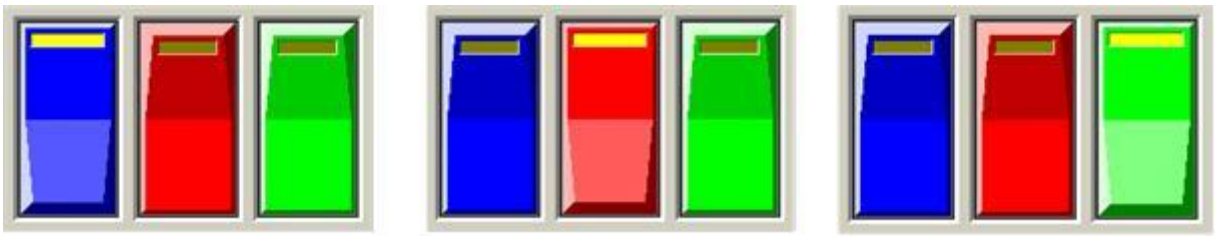
Постановка завдання до програмування програми

Необхідно для *Windows* створити 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 лівої клавіши мишки над зображенням нестандартної першої кнопки (лівої) зображення цих 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 лівою клавіши мишки над зображенням нестандартної

другої кнопки (середнєю) зображення цих кнопок повинно змінюватись на зображення (*Button_2_On*) відповідно до рис. 6-2;

❖ При натисканні лівою клавіши мишки над зображенням нестандартної третьої кнопки (правої) зображення цих кнопок повинно змінюватись на зображення (*Button_3_On*) відповідно до рис. 6-2;

❖ Після натискання клавіші мишки над однією з нестандартних кнопок форми *C++* програма повинна одночасно з переключенням зображень стану нестандартних кнопок також у полі *Image*1 переключати зображення приладів, які показані на рис. 6-3 та 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;

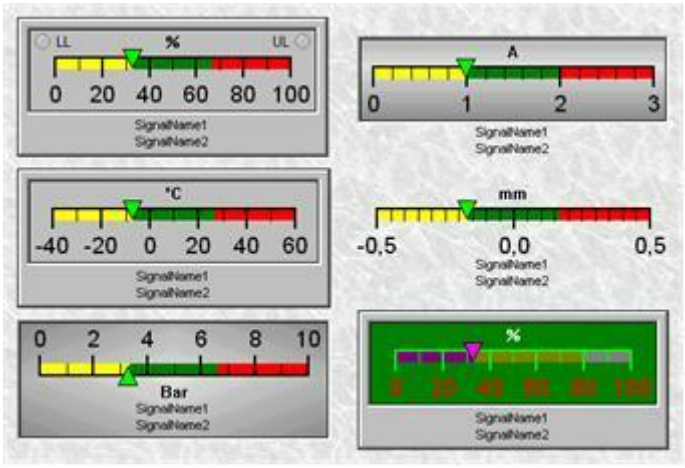


Рис.6-3.

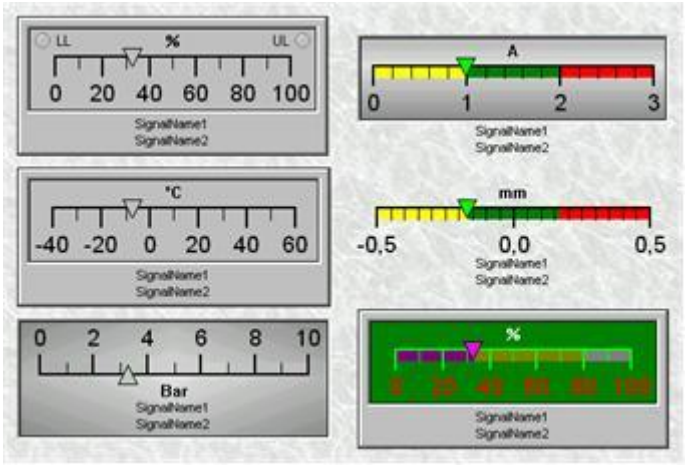


Рис. 6-4.

❖ При розташуванні курсору мишки над нестандартними кнопками зображення стандартного курсору «стрілка» повинно замінюватися на інший курсор у вигляді «рука»;

❖ Необхідно в редакторі зображень *Image Editor* [1] створити нестандартний курсор і підключити у файл ресурсів прикладної *C++* програми для обробки команд мишки до подій у програмі;

❖ 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.

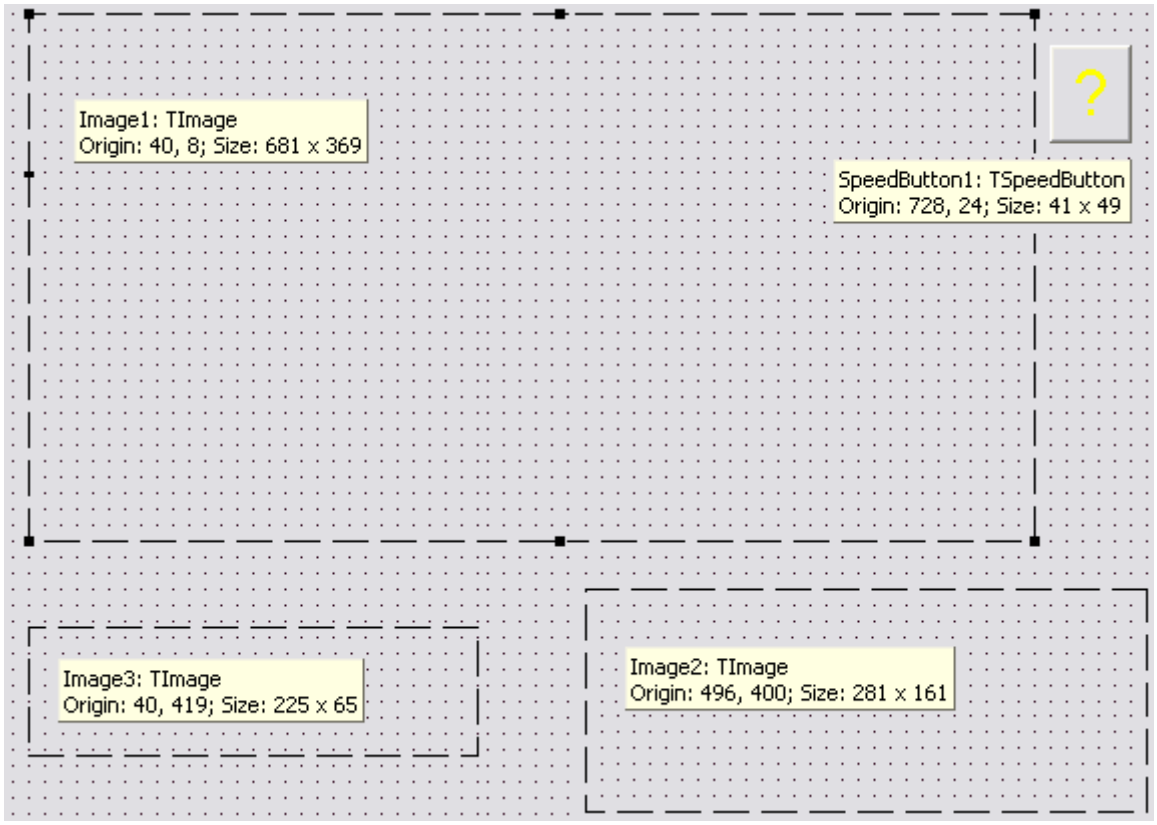


Рис. 6-5. Розміри до компонент *Image*1, *Image*2 та *Image*3 на формі вікна.

↓↓↓↓↓↓ Інформація для користувача ↓↓↓↓↓↓

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

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

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

⬇⬇⬇⬇⬇⬇ Алгоритми для програмування ⬇⬇⬇⬇⬇⬇

Порядок дій і команд для візуального програмування програми

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 у 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.

Крок 2. Виберіть в бібліотеці компонент *VCL* сторінку *Additional* і з неї розташуйте на форму компоненти *Image1, Image2* та *Image3* та 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 *Image1* потім на *Image2* 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 *Image1* і відкрийте шаблон для події *OnMouseDown* та заповніть відповідні команди з модуля *U_work_6.cpp* .

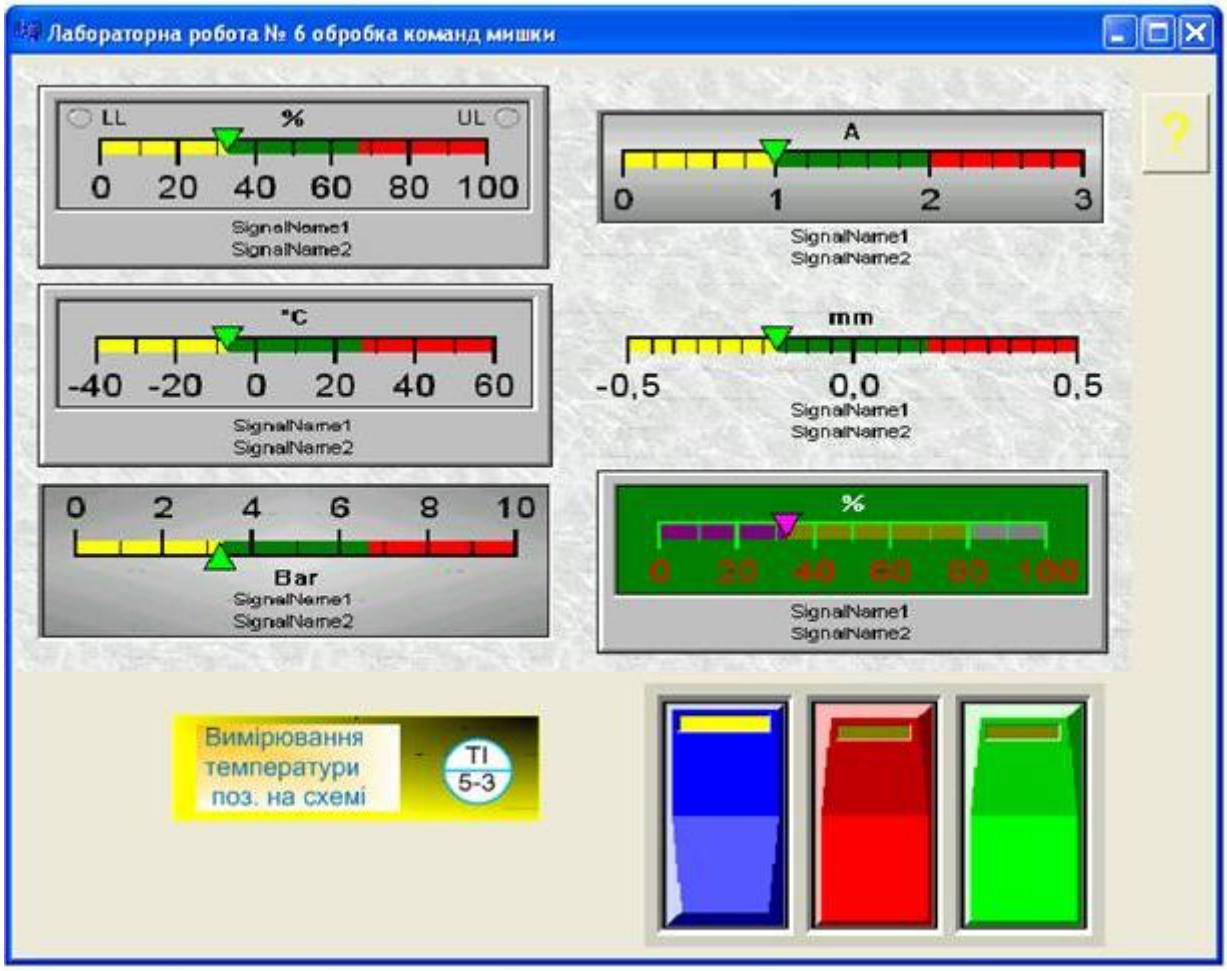


Рис. 6-6.

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* необхідно створіти нестандартний тимчасовий курсор і його зареєструвати у файл ресурсів створюваної прикладної *C++* програми. Для цього виконайте:

- Виберіть у меню таки команди *Tools/ ImageEditor*;
- Виберіть у меню наступні команди *File/ Open* і знайдіть у проекті файлів для створюваної прикладної *C++* програми файл ресурсів проекту *(.res).*

У результаті відкриється вікно зі структурою написів та в меню буде додано *Resource*;

- Виберіть таки команди *Resource/ New/ Cursor* , щоби додати нову групу *Cursor* і нове ім’я *Cursor*1.
- Над ім’ям *Cursor*1 зробіть щиглика правою клавішею мишки і виберіть команду *Rename* та задайте нове ім’я нестандартного курсору.
- Над 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 для створюваної *C++* програми.

Крок 9. У файл *U_work_*6*.cpp* збережіть для модуля прикладної програми таки три віда текстів: 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;
//--- вказники 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);  //--- для 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 миготіння 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 миготіння 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 миготіння 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 миготіння 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 миготіння при 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 миготіння 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 виконанні лабораторної роботи № 6 вікно *C++* програми повинно мати зображення, як це показано на рис. 6-6.

Контрольні запитання до завдання № 6

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

2. Покажіть оператори у *C++* програмі яки забезпечують вивід повідомлень у компоненту *Image*3.

3. Пояснить структуру операторів, записаних у перемикач *switch*( ) при обробці команд мишки.

4. Поясніть заголовки функцій, які обробляють команди мишки над зображеннями приладів на рис. 6-3 та рис. 6-4.

5.Поясніть оператори, які забезпечують заміну курсора мишки на інший вид при виконанні команд мишки над зображеннями приладів та нестандартних кнопок у полі *Image*2.

Завдання № 7 та алгоритми до теми

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

Ціль створення прикладної програми. Інтегроване програмувальне середовище *C++ Builder* має відповідний набір компонент, інструментів, властивостей, методів і функцій для побудови графічних елементів до зображень на формі прикладної *C++* програми[3] і тому дане завдання передбачає вивчення таких особливостей програмування графічних елементів на формі вікна прикладної програми:

- Методики використання з *VCL* компонент для побудови графічних елементів на формі програми;
- Техніки роботи з *Canvas* – канвою форми прикладної програми;
- Методики рисування кольорових елементів до графічних зображень на формі програми;

- Правил налагодження компоненти для рисування відповідного типу лінії: суцільної лінії; штрихової ліні; пунктирної лінії та штрих-пунктирної лінії;
- Техніки рисування базових графічних фігур: дуга чи круг або еліпс; сегмент або сектор; прямокутник та інші фігури за допомого інструмента ***Pen*** (перо).

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Загальні зведення до компонент та інструментів для рисування графічних елементів на формі вікна 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 створення графічних елементів до зображень на формі прикладної програми.

Таблиця 7.1

Піктограмка	Назва компоненти	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 рисовати графічні елементи

Канва (*Canvas*) не є компонентой. 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 та графічні компоненти *Image*, *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 зображень із системою заховані для користувача.[3]

Кожна крапка канви має координати *X* та *Y* . Система координат канви починається з лівого верхнього кута області полотна. Координата *X* зростає при змінах від ліворуч до праворуч, а координата *Y* змінюється зверху до низу. Координати змінюються у пікселях. ***Піксель*** – це найменший елемент у графічному зображенні, яким можна маніпулювати. Найважливіша властивість пікселя - це колір. Для опису кольору використовується тип *Tcolor*. Малювати на канві можна різними способами:

- ***Перший варіант*** *(Pixels)* – рисування по окремим пікселям.
- ***Другий варіант*** *(Pen)* – рисування пером *Pen*.

Властивість *Pixels* у канві являє собою двовимірний масив, що визначає на канві кольори крапок:

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

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

```
//-------------------------------------------------------------
float x, y;             //---координати функції
float PX, PY;      //---координати пікселей
for( PX=0; PX <= Image1->Width;  PX ++ )
{  //--x – координата, що відповідає пікселю з координатою PX
x = Xmin + PX*(Xmax - Xmin) / Image -> Width;
```

```
y = F(x);
//--PY –координата пікселя, що відповідає координаті У
PY = Image1->Width – (y - Ymin)* Image->Height/(Ymax - Ymin);
//---установлюється чорний колір обраного пікселя
Image1->Canvas->Pixels[PX][PY] = clBlack;
}
//---------------------------------------------------------
```

У канви мається властивість *Pen* - перо. Цей об'єкт, у свою чергу має ряд властивостей. Властивість *Color* – колір, яким наноситься малюнок, властивість *Widht* - ширина лінії, що малюється. Ширина лінії задається у 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 лінії зі штрихами і пунктирами доступні тільки при Widht = 1. У противному випадку лини цих стилів малюються як суцільні. Стиль *psinsideFrame* - єдиний, який 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,У*) – 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,У)*. 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 у 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).

↓↓↓↓↓↓ Інформація для користувача ↓↓↓↓↓↓

Постановка завдання до програмування програми

Для завдання № 7 необхідно виконати таки два завдання:

Завдання А. Необхідно створити прикладну *C++* програму, яка буде демонструвати побудову 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)* .

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

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

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

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

- нарисувати блок-схему алгоритму з роботи програми до завдання А;
- нарисувати блок-схему алгоритму з роботи програми до завдання Б;
- 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*.

⬇⬇⬇⬇⬇⬇ Алгоритми для програмування ⬇⬇⬇⬇⬇⬇

Порядок дій і команд при виконанні програмування до завдання 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 змінювання 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 пікселей
 for (PX = 0; PX <= Image1->Width; PX++)
  {
//------X - координата, що відповідає пікселю з координатою PX
   X = PX * 4 * Pi / Image1->Width;
```

```
    Y =  sin(X);
  //-----PY - координата пікселя, що відповідає координаті Y
    PY = Image1->Height - (Y+1) * Image1->Height / 2;
  //-----Встановлюється колір обраного пікселя
    Image1->Canvas->Pixels[PX][PY] = clRed;
   }
  }
  //---------------------------------------------------------------------------
```

Крок 6. Для побудови за допомогою інструмента *Pen* другого графіку функції *Sin*(*X*) додайте на форму компоненту *Image*2 .

Крок 7. Необхідно сформувати два однакових поля для рисування графіків:

- Утримуючи натиснутою клавішу *Shift* замаркіруйте *Image*1, *Image*2 і виконаєте команду правою клавішею мишки 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 пікселей
Image2->Canvas->MoveTo(0,Image2->Height/2);
 for (PX = 0; PX <= Image1->Width; PX++)
  {
//-----X - координата, що відповідає пікселю з координатою PX
```

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

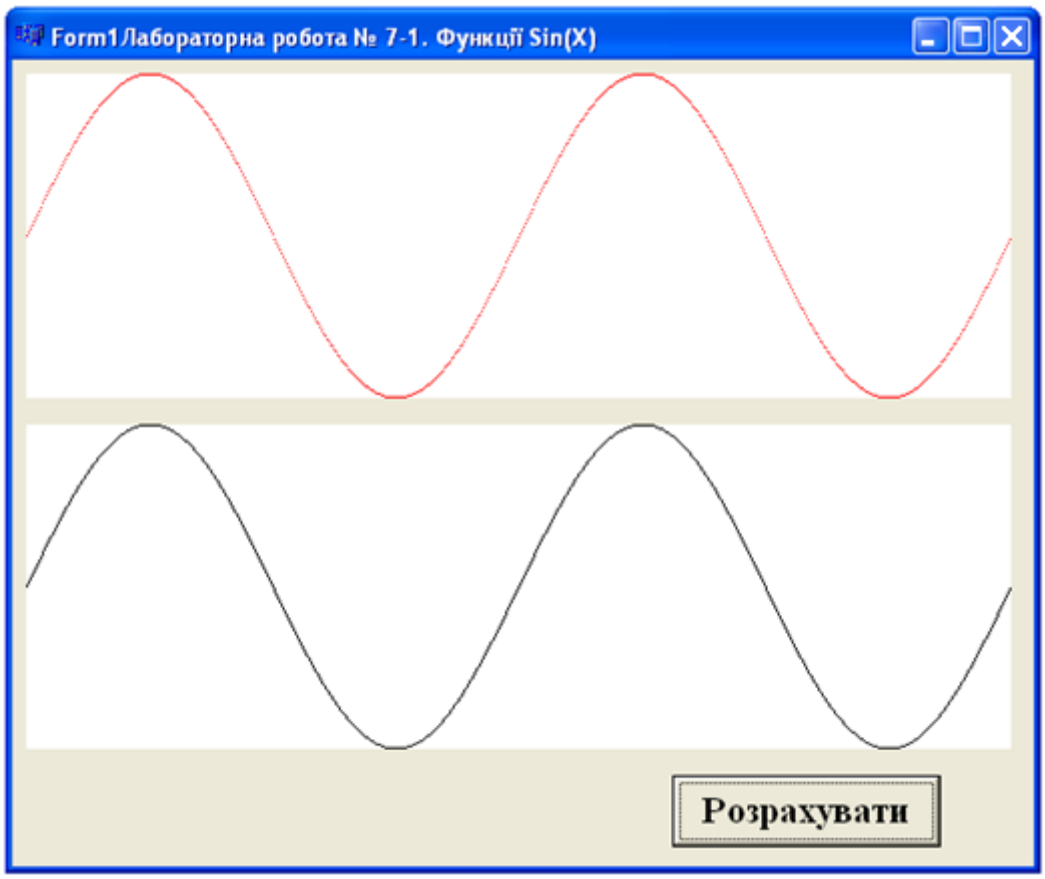


Рис. 7-1.

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

Порядок дій і команд при виконанні програмування до завдання 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 *Image1* і заповнить у 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 спостерігайте результат рисування у вигляді рис. 7-2.

Рис.7-2.

Контрольні запитання до завдання № 7

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);

Завдання № 8 та алгоритми до теми

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

Ціль створення прикладної програми. Інтегроване програмувальне середовище *C++ Builder* має можливості для розробки прикладних *C++* програм з відеороликами та мультипликаціями[3] на основі компоненти *Animate* і тому дане завдання № 8 передбачає наступні вивчення:

- 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.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

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 *Win32* та 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.[3]

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 у 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* можна властивість 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 у програмі обробока 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.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Постановка завдання до програмування програми

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

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

⇓⇓⇓⇓⇓⇓ Інформація для користувача ⇓⇓⇓⇓⇓⇓

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

Після запуску у роботу *C++* програми з завдання № 8 для захисту результатів програмування необхідно виконати таке:

- нарисувати блок-схему алгоритму з роботи програми до завдання А;
- нарисувати блок-схему алгоритму з роботи програми до завдання Б;
- 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*.

⇓⇓⇓⇓⇓⇓ Алгоритми для програмування ⇓⇓⇓⇓⇓⇓

Порядок дій і команд при виконанні програмування завдання А

Крок 1. Активізуйте проект файлів для нової *C++* програми та виконайте команду для зберігання 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 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.

- Збережіть 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, розробляємой *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 запропоновану назву проекту *Project1.bpr* змінюємо на *P_ Animate-AVI.bpr*.

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

Крок 3. Після закінчення створення форми *C++* програми по рис. 8-1 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 файлу, 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 анимаціця має звук.";
  ShowMessage(msg);
  return;
  }
  Edit1->Text = OpenDialog1->FileName; //відображення назви файлу
  RadioButton1->Checked = true;//режим безперервного 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 кнопку "Наступний кадр"-----------------------------------
```

```
void __fastcall TForm1::Button2Click(TObject *Sender)
{
CFrame = CFrame + 1;
//---відображення зображення 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" неробоча
}
//---------------------------------------------------------------------------
```

```
//---------------------------- 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*.

⬇⬇⬇⬇⬇⬇ Алгоритми для програмування ⬇⬇⬇⬇⬇⬇

Порядок дій і команд при виконанні програмування завдання Б:

Крок 1. Активізуємо проект файлів для нової *C++* програми та виконуємо команду для зберігання 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 *Form1* з'явився напис «Анімація стандартних кліпів Windows» ;
- Збережіть новий проект файлів разом з порожньою формою і з змінами в заголовку вікна. Це автоматично активізує новий проект файлів і *C++ Builder* запам'ятає шлях для швидкого збереження змін у файлах, розробляємой *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 *Unit1* на файл *U_Animate-AVI.cpp*, а назву проекту *Project1.bpr* змінюємо на *P_ Animate-AVI.bpr*.

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

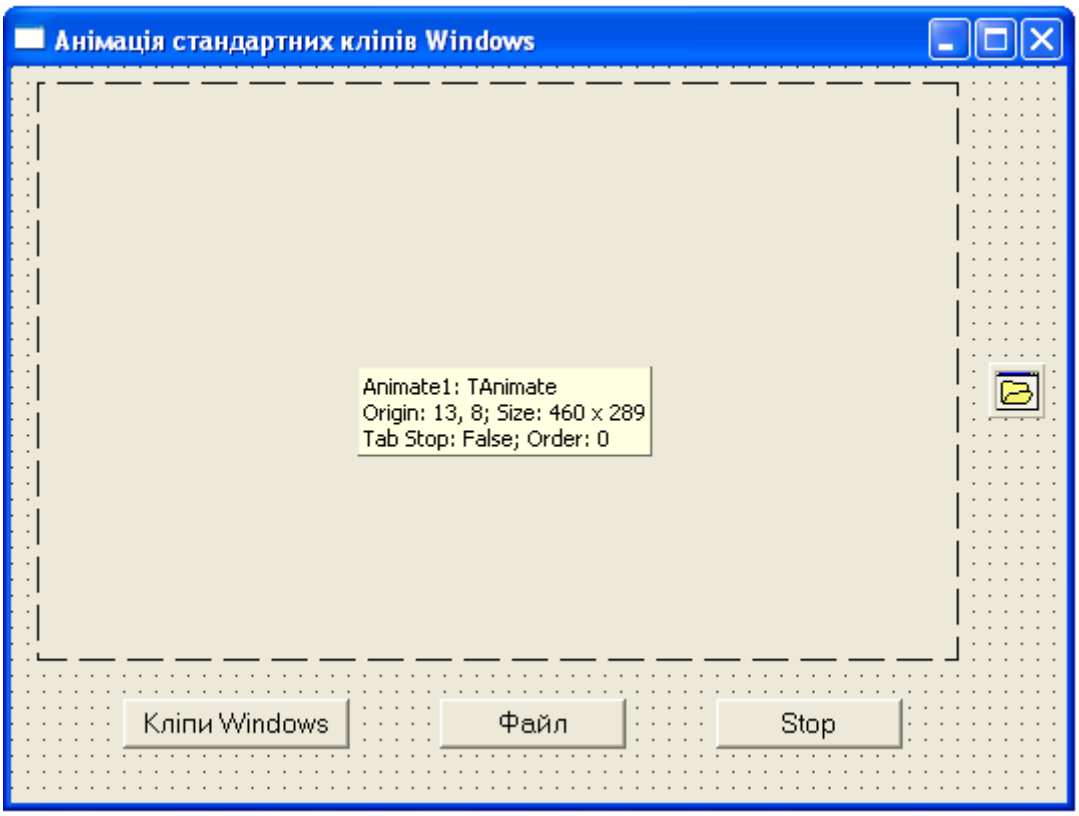


Рис. 8-2 . Зображення форми *C++* програми для виконання завдання Б.

Крок 3. Після закінчення розробки форми програми згідно рис. 8-2 потрібно у редакторі текстів сформувати наступний листинг 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;
}
//---------------------------------------------------------------------------
```

Контрольні запитання до завдання № 8

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

Завдання № 9 та алгоритми до теми

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

Ціль створення прикладної програми. Інтегроване програмувальне середовище *C++ Builder* має можливості для розробки прикладних *C++* програм

з динамічною графікою[3] на основі використання компоненти «*Timer*» і тому завдання № 9 даної роботи складається у вивченні:

- Техніки і методики використання компоненти "***Timer*** " при побудові прикладної *C++* програми з динамічною графікою на формі вікна програми;
- Алгоритму з функціонування динамічної графіки на формі вікна прикладної програми.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Загальні зведення до динамічної графіки на формі вікна програми

Динамічна графіка на формі вікна програми будується на принципах мультиплікації окремих кадрів з зображеннями. 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.[1],[3]

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 у 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 властивості *Interval* = **0** або для *Enabled* = *false*, тоді таймер стане вимкнутим. Для включення та 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* .

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Постановка завдання до програмування програми

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

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

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

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

– визначити перелік функцій, які 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 на формі у вікні лабораторної роботи № 9.

⇓⇓⇓⇓⇓⇓ Алгоритми для програмування ⇓⇓⇓⇓⇓⇓

Порядок дій і команд для візуального програмування програми

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”**;
- Збережіть новий проект файлів разом з порожньою формою і з змінами в заголовку вікна. Це автоматично активізує новий проект файлів і *C++ Builder* само запам'ятає шлях для швидкого збереження змін у файлах, розробляємой 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 для запропонованої назви проекту *Project*1.*bpr* замінюємо цю назву на таку *P_dinamica.bpr*.

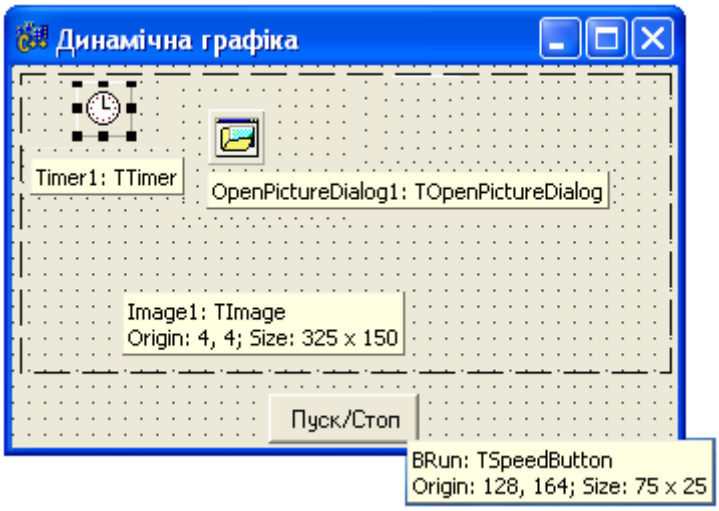


Рис. 9-1. Форма програми для роботи з динамічною графікою.

Крок 2. Далі потрібно створити форму для прикладної *C++* програми з компонентами, показаними на рис. 9-1.

Крок 3. Після закінчення розроби форми програми згідно 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;
}
//---------------------------------------------------------------------------
```

Контрольні запитання до завдання № 9

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, які рисують графічні елементи для динамічної фігури на формі *C++* програми.
5. Пояснить оператори за допомогою яких можна змінити розміри динамічної графічної фігури у лабораторній роботі № 9.

Завдання № 10 та алгоритми до теми

Алгоритм роботи з фрагментами зображення розташованого у компоненті Image

Ціль створення прикладної програми. Інтегроване програмувальне середовище *C++ Builder* має можливості для розробки прикладних *C++* програм з обробкою фрагментів растрового зображення[3] на основі використання компоненти «*Image*» і тому завдання даної лабораторної роботи складається у практичному вивченні:

- Техніки роботи з фрагментами зображення з файлу *.bmp* на основі програмної обробки компоненти *Image;*
- Алгоритму по формуванню фрагментів зображення з файлу *.bmp* ;
- Алгоритму з обробки подій мишки при роботі з фрагментами зображення з файлу *.bmp* .

⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇

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 компонентой *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.[1],[3]

⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇

Постановка завдання до програмування програми

При виконанні лабораторної роботи № 10 потрібно для *Windows* створити *C++* програму, у який на формі вікна повинно виконуватись:

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

⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇

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, які забезпечують створення фрагментів зображень відповідних до загального 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 загальному зображенню з компоненти *Image*1 на формі вікна лабораторної роботи № 10.

⇓⇓⇓⇓⇓⇓ Алгоритми для програмування ⇓⇓⇓⇓⇓⇓

Порядок дій і команд для візуального програмування програми

Крок 1. Активізуємо проект файлів для нової *C++* програми та виконаємо 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*»;
- Збережіть новий проект файлів разом з порожньою формою і заміною в заголовку вікна. Це автоматично активізує новий проект файлів і *C++ Builder* запам'ятає шлях для швидкого збереження змін у файлах, розробляємой 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 запропоновану назву проекту *Project*1.*bpr* змінюємо на назву *P_fragment.bpr*.

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

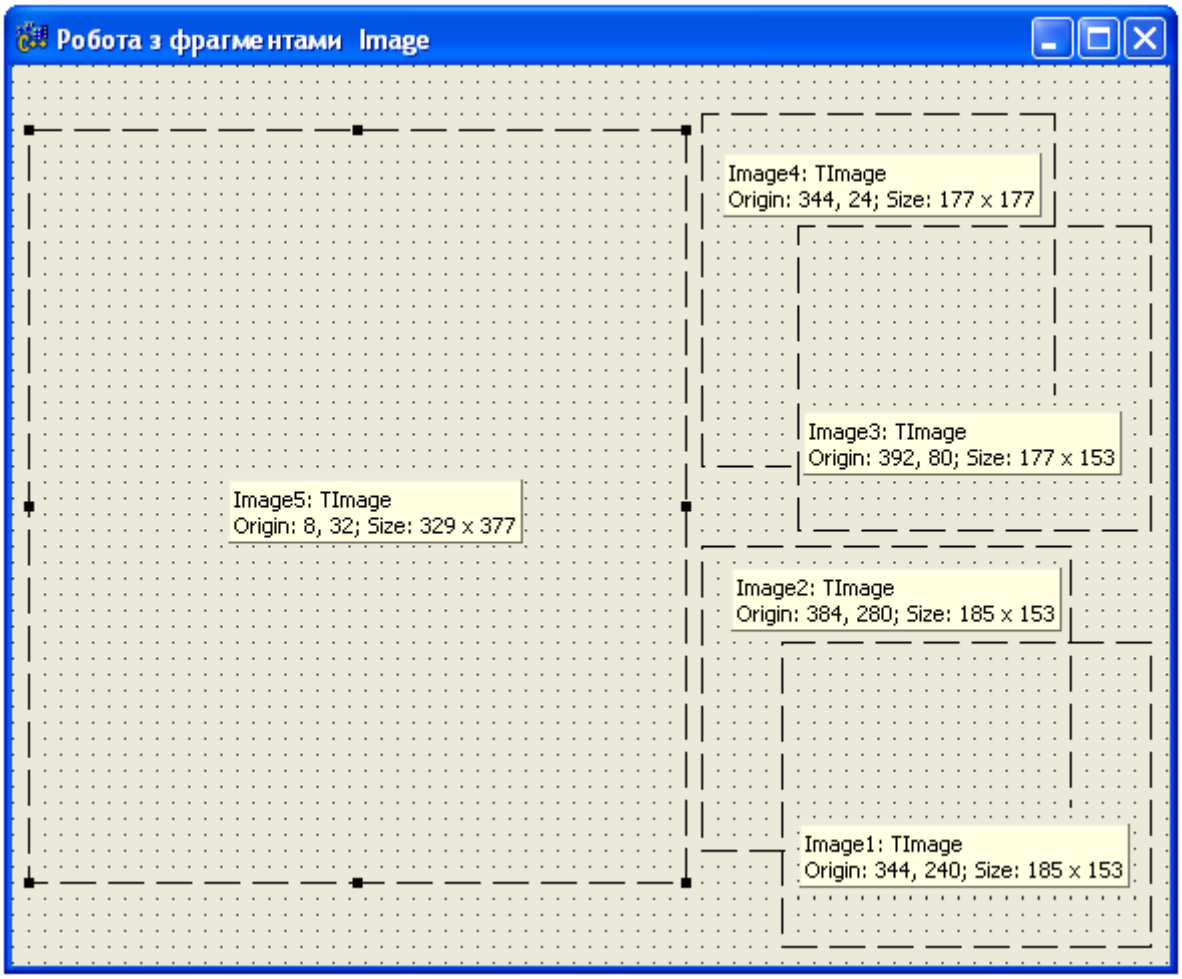


Рис. 10-1. Форма програми для роботи з фрагментами компоненти *Image*.

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 зображення 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 завдання № 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 програмі 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 у функціях, які створюють фрагменти зображень з компоненти *Image*1 у лабораторній роботі № 10.
5. Пояснить оператори за допомогою яких визначаються розміри зображень фрагментів у лабораторній роботі № 10.

Завдання № 11 та алгоритми до теми

Правила зчитування ScanCode та ASCII кодів клавіш клавіатури комп'ютера та їх використання у прикладній C++ програмі

Ціль створення прикладної програми. У процесі програмування прикладної програми передбачено виконання таких навчальних завдань:

- Визначення *ASCII* кодів до символів на кнопках клавіатури та їх використання у прикладних програмах при виконанні у операційному середовищі *MS DOS*;
- Практичне вивчення техніки використання *ASCII* кодів до символів при побудові графічних зображень на екрані дисплея комп'ютера.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Загальні зведення до *ScanCode та ASCII Code*

Кожен символ, зазначений на кнопці клавіатури, характеризується стандартним *ASCII* кодом, який можна використовувати у *C++* програмах.

Наприклад, для перевірки умови, клавіша з яким символом була натиснута, необхідно записати в умову *ASCII* коду відповідного символу. У стандартних *ASCII* таблицях також знаходяться коди для псевдографічних значків, за допомогою яких можна у текстовому режимі роботи дисплея за допомогою прикладної програми на екрані будувати різні рамки та інші фігури і позначення. На клавіатурі кожна кнопка позначена відповідним *Scan* кодом, що разом з *ASCII* кодом символу формує у буфері клавіатури *Scan-ASCII* код. При натисканні клавіші із символом на клавіатурі у буфер клавіатури записується відповідний *Scan-ASCII Code*, по якому апаратні переривання *BIOS* операційної системи компютера визначають, яку команду введено з клавіатури у *C++* програму.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Постановка завдання до програмування програми

У лабораторній роботі необхідно виконати наступні два завдання:

Завдання А. Необхідно створити за допомогою *Consol Wizard* файл *C++* програми, що виконується, у вигляді консольного додатка *MS DOS*. Дана прикладна програма повинна по діагоналі вікна будуватися три рамки за допомогою псевдографічних значків. У цих рамках потрібно забезпечити можливість відображення будь-яких символів із клавіатури та на границі рамки для останнього введеного символу на екрані потрібно показати *ASCII* код останнього введеного символу.

Завдання Б. Потрібно створити за допомогою *Consol Wizard* файл *C++* програми, що виконується, у вигляді консольного додатка *MS DOS*. При роботі даної *C++* програми на екрані повинні з'являтися запити на введення координат лівого верхнього кута рамки і координат правого нижнього кута рамки. Після введення координат кутів рамки на екрані повинні з'явитися кутові псевдографічні значки, а при натисканні будь-якої клавіші на клавіатурі(пробіл) на екрані повинні малюватися лінії рамки.

⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇

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

Після запуску у роботу *C++* програм до завдань **А** та **Б** з лабораторної роботи № 11 для захисту результатів програмування необхідно виконати таке:

- нарисувати блок-схему алгоритму до прикладної програми з проекту *P_Lab2_A* лабораторної роботи;
- нарисувати блок-схему алгоритму до прикладної програми з проекту *P_Lab2_Б* лабораторної роботи.

⬇⬇⬇⬇⬇ Алгоритми для програмування ⬇⬇⬇⬇⬇

Методика виконання завдання А.

Порядок дій і команд для виконання програмування завдання А

Крок 1. За допомогою кнопки *New* відкрийте вікно *New Items* і виберіть *Console Wizard* та для опції *Source Type* задайте режим *C++*. У результаті *C++ Builder* відкриє шаблон тексту до вихідної прикладної програми у такому вигляді:

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

Крок 2. Додайте у відповідні місця шаблону прикладної програми вказівки препроцесору та оголошення і оператори у відповідності з наступним текстом до листингу програми:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
#include <conio.h>
#include <system.hpp>
#include <stdio.h>
#define dx   24     /* Ширіна рамки вікна */
#define dy   5      /* Висота рамки вікна */
void my_box(int xul,int yul,int xlr,int ylr,int btype)
  {
static int boxcar[2][6] = { /* Графічни символи для контурів */
 { 218,196,191,179,192,217 },     /* одинарний контур     */
 { 201,205,187,186,200,188 }      /* двійний контур       */
   };
     int i,hzchar,vtchar;
   if(btype) {
      hzchar = boxcar[btype -1][1];
      vtchar = boxcar[btype -1][3];
 /* Вивід верхньої та ніжньої сторони контуру   */
 gotoxy(xul,yul);
   for(i = xul; i<= xlr; i++) putch(hzchar);
 gotoxy(xul,ylr);
   for(i = xul; i<= xlr; i++) putch(hzchar);
/* ввід правоі та лівоі сторони контуру  */
```

```
   for(i = yul; i<= ylr; i++)
     { gotoxy(xul,i); putch(vtchar);
       gotoxy(xlr,i); putch(vtchar);
     }
/* Вивід кутів контуру */
/*  верхнього лівого кута */
gotoxy(xul,yul); putch( boxcar[btype - 1][0] );
/* верхнього правого кута */
gotoxy(xlr,yul); putch( boxcar[btype - 1][2] );
/* ніжнього лівого кута */
gotoxy(xlr,ylr); putch( boxcar[btype - 1][5] );
/* ніжного правого кута */
gotoxy(xul,ylr); putch( boxcar[btype - 1][4] );
     }
  }
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
int c,ch;
   int btype ;
   int xul,yul;      /* кординати лівого кута рамки */
   int xlr,ylr;      /* кординати правого кута рамки */
   int N,n;   /* N -номер вікна, n –кількість символів у вікні */
   char S_1[7],S_2[15],S_3[15],S_4[8]; /* ліворуч Ctr+shift -> шріфт EN*/
                                       /* праворуч Ctrl+Shift -> шріфт Ru */
p1: clrscr();  btype = 2;
 gotoxy(3,22); textcolor(RED); cprintf("ESC");
```

```
gotoxy(7,22); textcolor(GREEN); CharToOem("Выход\n",S_1); cprintf(S_1);
gotoxy(1,23); textbackground(LIGHTCYAN); textcolor(RED);
     CharToOem("Лабораторна    \n",S_2); cprintf(S_2);
  gotoxy(1,24); CharToOem("робота  № 11 ->1 \n",S_3); cprintf(S_3);
 textmode(LASTMODE); gotoxy(1,1);
     N = 1; xul =1; yul = 1;  ch = (dx - 2) * (dy -1) + 2 ;
 do
  { xlr = xul + dx;  n = 0;
    ylr = yul + dy;
 textcolor(CYAN);
  my_box(xul,yul,xlr,ylr,btype);
  window(xul + 1,yul + 1,xlr - 1,ylr - 1);
    gotoxy(0,0);
   do
   { c = getch();
       if( c != 0x1b )
        {   textbackground(BLUE); textcolor(YELLOW);
            putch(c);
             n++;
        }
          else
            goto p2;
   } while( n <= ch );
 window(1,1,80,25);
     gotoxy(xlr - 17,ylr + 1);  textbackground(LIGHTGRAY);
     textcolor(MAGENTA); cprintf(" ASCII "); CharToOem("код
\n",S_4);cprintf(S_4);
     gotoxy(xlr - 17,ylr + 2);  textbackground(GREEN);
textcolor(RED); cprintf("  "); putch(c);
```

```
                    cprintf("= 0x%x  ",c);
        textmode(LASTMODE);
         N++;   btype --;
       xul = xlr + 1;     /* кординати для window(1,1,80,25) */
         yul = ylr + 1;
      } while( N <= 3);
   window(1,1,80,25);
   goto p1;
p2:         ;
        return 0;
}  /*end main*/
//---------------------------------------------------------------------------
```

⇓⇓⇓⇓⇓⇓ Алгоритми для програмування ⇓⇓⇓⇓⇓⇓

Методика виконання завдання Б.

Порядок дій і команд для виконання програмування завдання Б:

Крок 1. За допомогою кнопку *New* відкрийте вікно *New Items* і виберіть *Console Wizard* та для опції *Source Type* задайте режим *C++*. У результаті *C++ Builder* відкриє шаблон тексту до вихідної прикладної програми у такому вигляді:

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

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

Крок 2. Додайте у відповідні місця шаблона програми вказівки препроцессору, оголошення та оператори у відповідності з наступним текстом до листингу програми:

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
//---------------------------------------------------------------------------
/*=============================================*/
/*      Рисування рамки за допомогою ASCII кодів          */
/*              псевдографічних символів                  */
/*============================================= */
#include <system.hpp>
#include <stdio.h>
#include <conio.h>
 void draw_border( int startx,int starty,int endx,int endy)
 { register  int i;
   textcolor( LIGHTCYAN);
   gotoxy(startx,starty); cprintf("%c",218); /* putchar( 218 );*/
   gotoxy(startx,endy);   cprintf("%c",192); /* putchar( 192 );*/
   gotoxy(endx,starty);   cprintf("%c",191); /* putchar( 191 );*/
   gotoxy(endx,endy);     cprintf("%c",217); /* putchar( 217 );*/
    getch();
   for( i = startx + 1; i < endx ; i++ )
    {   textcolor( LIGHTRED);
      gotoxy(i,starty); cprintf("%c",196); /*putchar( 196 );*/
      gotoxy(i,endy);   cprintf("%c",196);  /*putchar( 196 );*/
      getch( );
    }
```

```
      for( i = starty + 1; i < endy ; i++ )
       {   textcolor( LIGHTMAGENTA);
         gotoxy(startx,i); cprintf("%c",179); /* putchar( 179 );*/
         gotoxy(endx,i);   cprintf("%c",179); /* putchar( 179 );*/
        getch( );
       }
   textmode(LASTMODE);
 }
//---------------------------------------------------------------------------
#pragma argsused
int main(int argc, char* argv[])
{
  int lux,luy;   /* Лівого верхнього кута рамки */
  int rdx,rdy;   /* Правого ніжнього кута рамки */
  char S_1[15],S_2[15],S_3[15],S_4[20],S_5[22],S_6[22],S_7[22];
  char S_8[30],S_9[30],S_10[30],S_11[30];
                              /* ліворуч Ctr+shift -> шріфт EN     */
                              /* праворуч Ctrl+Shift -> шріфт Ru */
m1 :  clrscr();
     gotoxy(1,23); textbackground(LIGHTCYAN); textcolor(RED);
     CharToOem("Лабораторна    \n",S_1); cprintf(S_1);
   gotoxy(1,24); CharToOem("робота № 11->2  \n",S_2); cprintf(S_2);
   gotoxy(30,1);  textbackground(GREEN); textcolor(RED);
   CharToOem("ВВЕДІТЬ\n",S_3); cprintf(S_3);
   gotoxy(1,3);   textbackground(LIGHTGRAY); textcolor(LIGHTBLUE);
   CharToOem("Кординати лівого\n",S_4); cprintf(S_4);
   gotoxy(1,4);
   CharToOem("верхнього кута рамки \n",S_5); cprintf(S_5);
```

```
    gotoxy(28,4);   textmode(LASTMODE);  textcolor(LIGHTGREEN);
                  cprintf(" X="); scanf("%d", &lux);
    gotoxy(37,4);     cprintf(" Y="); scanf("%d", &luy);
      gotoxy(40,23);  textbackground(LIGHTGRAY); textcolor(LIGHTBLUE);
      CharToOem("Кординати правого\n",S_6); cprintf(S_6);
       gotoxy(40,24);
       CharToOem("ніжнього кута рамки \n",S_7); cprintf(S_7);
    gotoxy(62,24);  textmode(LASTMODE);   textcolor(LIGHTGREEN);
                   cprintf(" X="); scanf("%d", &rdx);
    gotoxy(72,24);     cprintf(" Y="); scanf("%d", &rdy);
     if( rdx < lux)
     {  textcolor(LIGHTBLUE);
      gotoxy(40,10);
      CharToOem("У ВАС помилка при введенні кординати X \n",S_8);
cprintf(S_8);
      gotoxy(40,11);
      CharToOem("ніжнього правого кута рамки \n",S_9); cprintf(S_9);
      getch();  textmode(LASTMODE);
        goto m1;
      }
         if( rdy < luy )
         {   textcolor(LIGHTBLUE);
          gotoxy(40,12);
     CharToOem("У ВАС помилка при введенні кординати Y \n",S_10);
cprintf(S_10);
      gotoxy(40,13);
      CharToOem("ніжнього правого кута рамки \n",S_11); cprintf(S_11);
           cprintf(" ");
         getch();   textmode(LASTMODE);
```

```
            goto m1;
          }
        lux =( lux > 70 ) ? 70 : lux;
        luy =( luy > 20 ) ? 18 : luy;
        rdx =( rdx > 80 ) ? 80 : rdx;
        rdy =( rdy > 20 ) ? 20 : rdy;
      draw_border( lux,luy,rdx,rdy );
          return 0;
    } /* end main() */
//---------------------------------------------------------------------------
```

Виконання прикладної програми в *MS DOS* до завдання Б з лабораторної роботи № 11 забезпечує на екрані дисплея вікно, яке зображено на рис. 11.1.

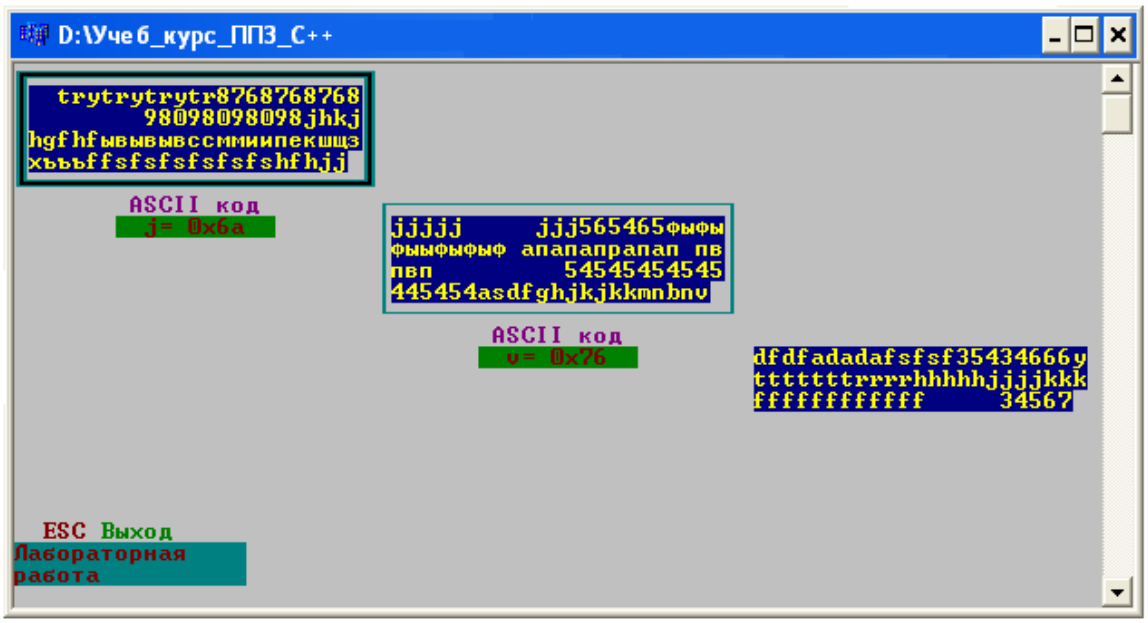


Рис. 11.1. Зображення вікна *C++* програми до завдання Б.

Контрольні запитання до завдання № 11

1. Пояснить блок-схему алгоритму з виконання прикладної програми до завдання А з лабораторної роботи № 11.
2. Пояснить, яка функція показує у вікні програми *ScanAsCII Code* після натискання клавіши на клавіатурі комп’ютера.

3. Як налаштовується *C++ Builder* для одержання коду консольної прикладної *C++* програми для виконання у середовищі *MS DOS*.
4. Поясніть алгоритм рисування рамок у вікні програми за допомогою псевдографічних символів у лабораторній роботі № 11.
5. Яка функція забезпечує у вікнах прикладної програми написи підказок кирилицею при виконанні лабораторної роботи №11.

Завдання № 12 та алгоритми до теми

Компоненти C++ Builder для візуалізації структури ієрархічних даних на носіях інформації

Ціль створення прикладної програми. Передбачається практичне виконання таких навчальних завдань:

- Вивчення методики створення прикладної *C++* програми для обробки і візуалізації ієрархічних структур даних, записаних на носіях інформації;
- Практична робота з компонентами *C++ Builder з* забезпечення перегляду даних, які записані на дисках комп'ютера, дисках CD-R, флешках та інших носіях інформації.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Загальні зведення до компонент зі структур даних

У бібліотеці VCL для відображення ієрархічних структур даних маються такі компоненти (таблиця 12.1).

Компонента *ListView* дозволяє в стилі *Windows* відображати дані у вигляді списків, таблиць, великих і дрібних піктограм.[3] Користувач зіштовхується з таким відображенням даних про файли при розкритті папок у *Windows*.

Стиль відображення інформації визначається властивістю *ViewStyle* , що може встановлюватися в процесі створення прикладної програми або програмно під час виконання прикладної програми.

Таблица № 12.1.

Компонента	Страница VCL	Опис компоненти
TreeView (вікно дерева даних)	Win32	Перегляд структури ієрархічних даних у вигляді дерева у стилі Windows95/98/2000
OutLine (вікно дерева даних)	Win3.1	Перегляд структури ієрархічних даних у вигляді дерева в стилі Windows 3
ListView (список даних в стилі Windows 95)	Win32	Відображення в стилі папок Windows списком у вигляді піктограм або стовпчиків
OpenDialog (відкрити файл)	Dialogs	Призначений для створення вікна діалогу "Відкрити файл"
FileListBox (список файлів)	Win3.1	Відображає список усіх файлів каталогу
DirectoryListBox (структура каталогів)	Win3.1	Відображає структуру каталогів диска
DriveComboBox (список дисків)	Win3.1	Список доступних дисків, що випадає
CDirectoryOutline (дерево каталогів)	Samples	Компонента для відображення структури каталогів обраного диска

Властивість *ViewStyle* може приймати такі значення: *vsicon* - великі значки; *vsSmallicon* - дрібні значки; *vsList* - список; *vsReport* - таблиця. Основна властивість компонента *ListView* задається у полі властивостей *Items*. Реорганізація дерева, зв'язана зі створенням або з видаленням многих вузлів з’єднань, може викликати неприємні мерехтіння зображення. Уникнути цього можна за допомогою методів *BeginUpdate* та *EndUpdate*. Первій метод забороняє перемальовування дерева, а другий - дозволяє.

⬇⬇⬇⬇⬇⬇ Інформація для користувача ⬇⬇⬇⬇⬇⬇

Постановка завдання до програмування прикладної програми

У лабораторній роботі № 12 необхідно створити для *Windows*:

- прикладну програму у який на вікні форми буде видна структура папок обраного диска та при відкритті подвійним щигликом на папці повинен з'являтися список файлів записаних в папку.

⇓⇓⇓⇓⇓⇓ Інформація для користувача ⇓⇓⇓⇓⇓⇓

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

Після запуску у роботу *C++* програми з лабораторної роботи № 12 для захисту результатів програмування необхідно виконати таке:

- нарисувати блок-схему алгоритму до виконання команд прикладної програми з проекту *P_Lab*12 лабораторної роботи № 12.

⇓⇓⇓⇓⇓⇓ Алгоритми для програмування ⇓⇓⇓⇓⇓⇓

Порядок дій і команд для візуального програмування програми

Крок 1. Перейдіть на форму програми і у властивості *Caption* напишіть назву роботи "Лаб. робота № 12 компонента *ListView* ", щоб цей текст з'явився в заголовку форми. Виконайте команду *File/Save Project As...* . На диску *D*:\ створіть папку *Lab*_12 для файлів проекту *P_Lab*12*.bpr* та файлу модуля форми *U_Lab*12*.cpp*.

Крок 2. Установіть на форму вікна програми таки компоненти: *DriveComboBox*1 та *DriveComboBox*2 , як це показано на зображенні вікна форми рис. 12.1.

Крок 3. На форму вікна програми установіть компоненти *DirectoryListBox*1 та *DirectoryListBox2* **,** як це показано рис. 12.2.

Крок 4. Установіть на форму компоненти *ListView*1 та *ListView2* , як це показано рис. 12.3.

Крок 5. Виконаєте попередню компіляцію тексту програми з метою перевірки роботи програми з встановленими компонентами.

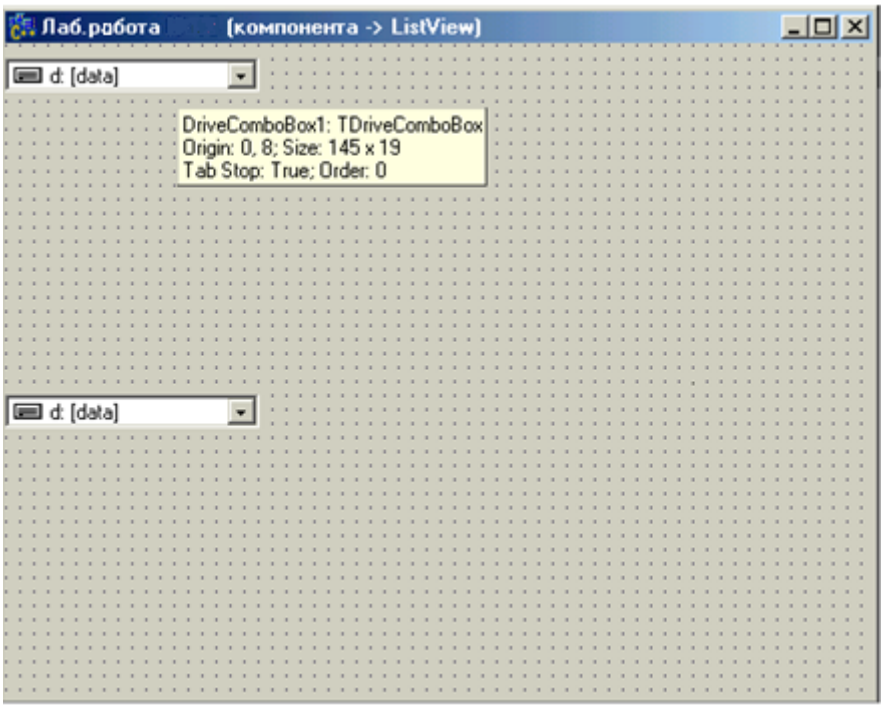


Рис. 12.1.

Крок 6. Установіть на форму компоненти *Label*1 та *Label*2 . У властивості *Caption* задайте синім кольором пояснювальний напис "Властивості значків", який розташовується згідно рис. 12.4.

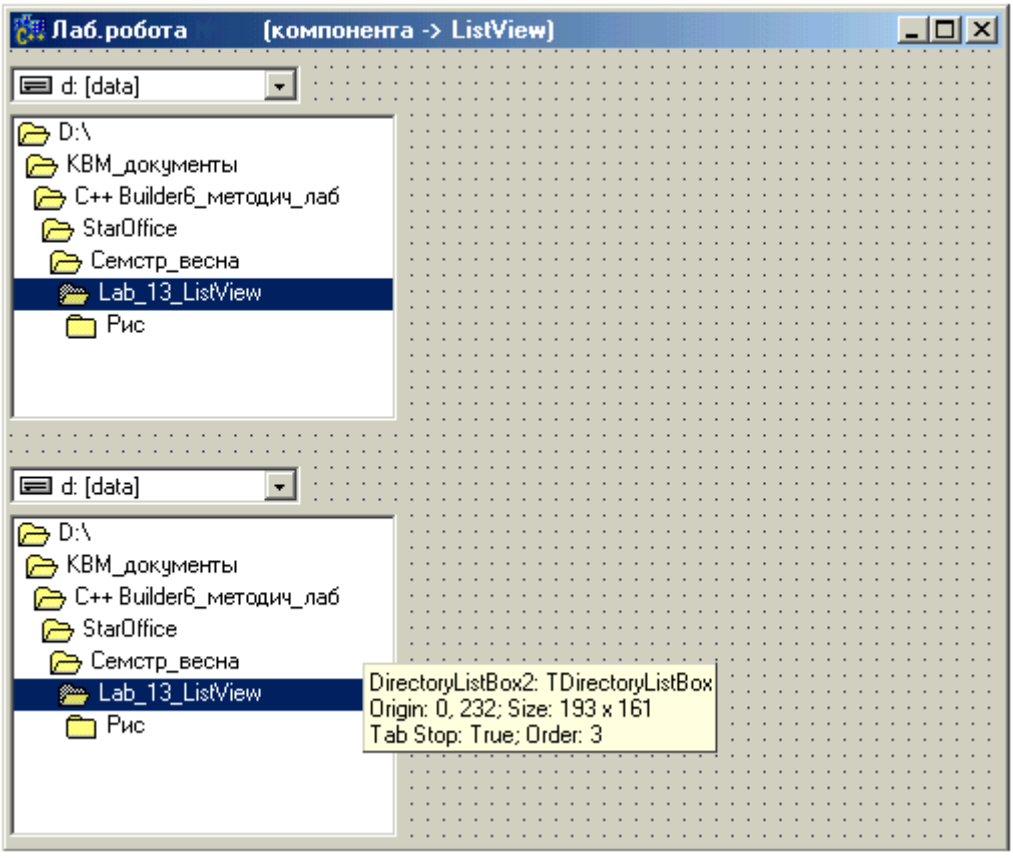


Рис.12.2.

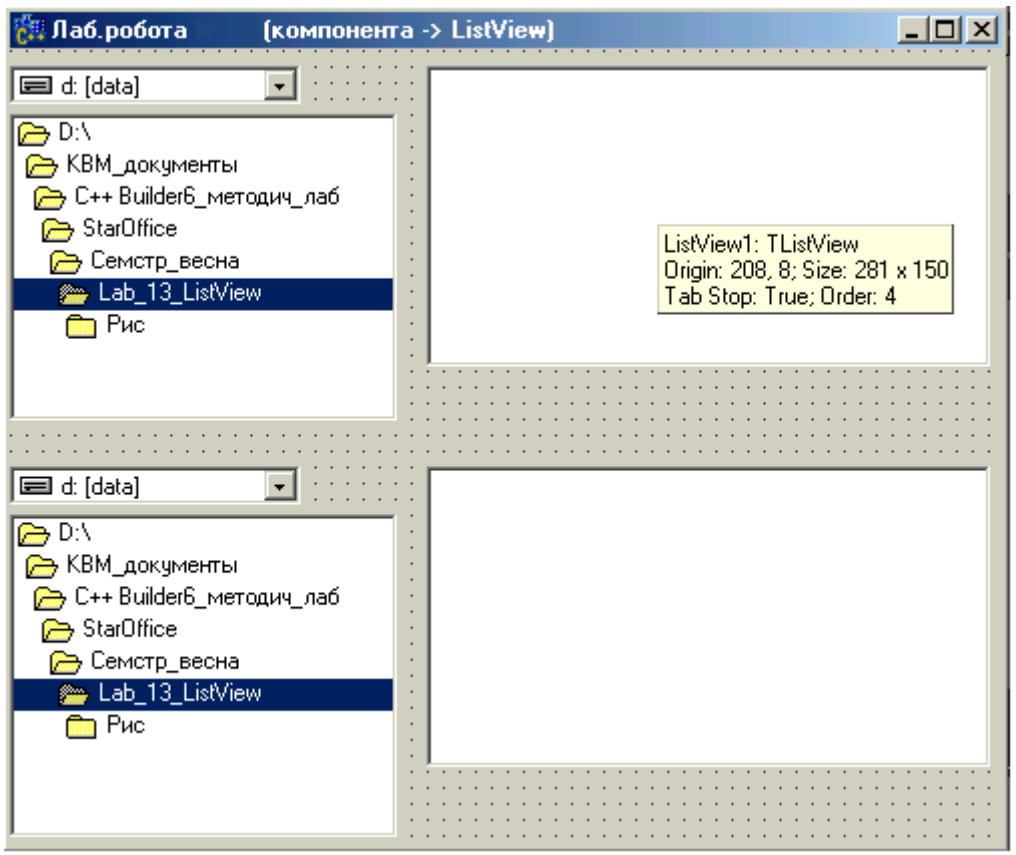


Рис.12. 3.

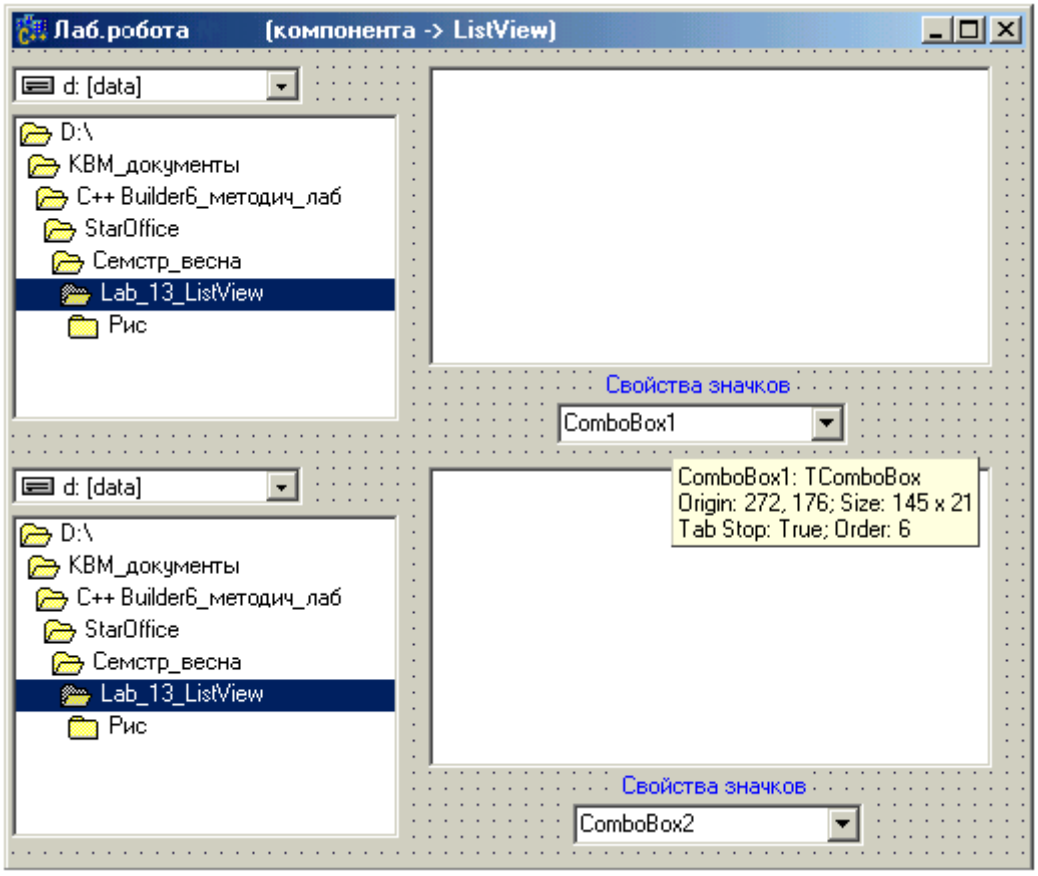


Рис. 12.4.

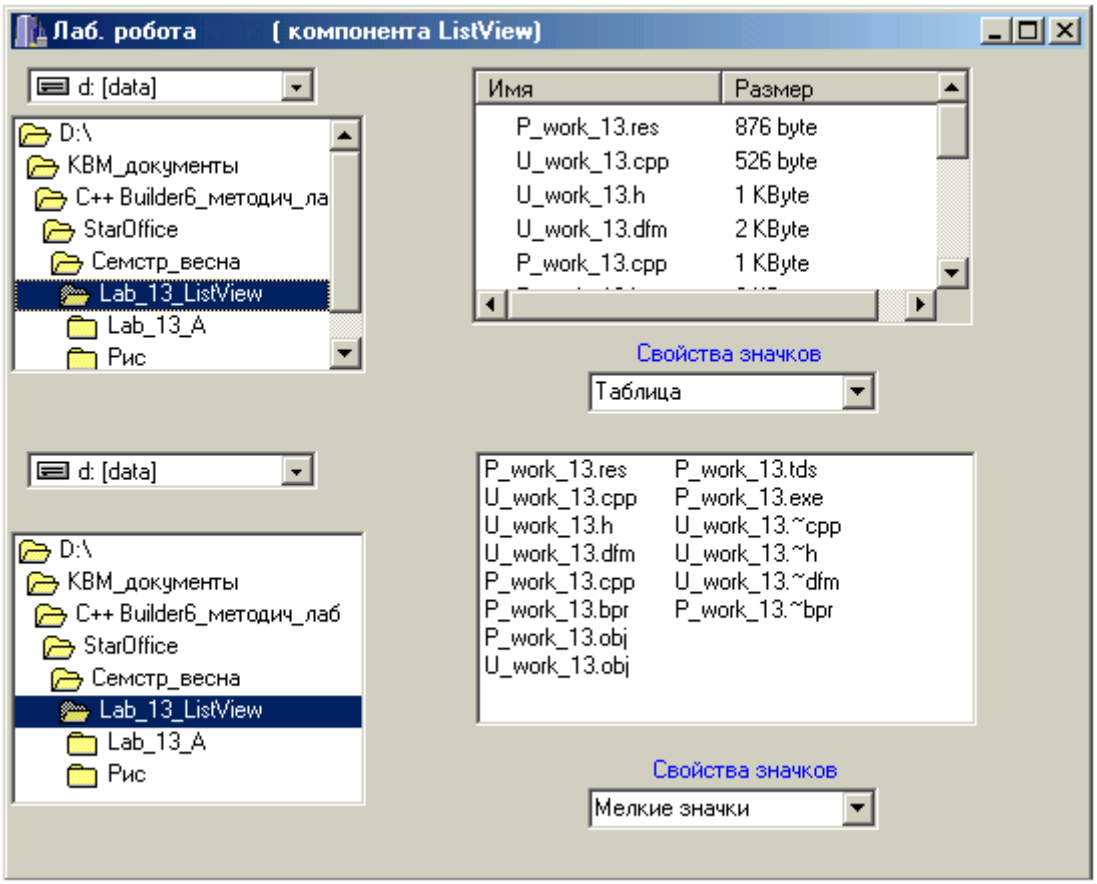


Рис. 12.5. Результат работи програми у *Windows*.

Крок 7. Установіть на форму компоненти *ComboBox*1 та *ComboBox*2 , як це показано рис. 12.4. У властивості *Items* задайте назви для списку, що випадають: «Великі значки» або «Мали значки» або «Список» чи «Таблиця».

Крок 8. Задайте подію void __fastcall TForm1::FormCreate(TObject *Sender); і заповніть оператори відповідно до лістингу файлу *U_Lab*12*.cpp* .

Крок 9. Задайте таки 2 події**:**

```
void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender);

void __fastcall TForm1::DirectoryListBox2Change(TObject *Sender);
```

та заповніть для цих подій оператори згідно листингу файлу *U_Lab*12*.cpp* .

Крок 10. Задайте наступні 2 події:

```
void __fastcall TForm1::ComboBox1Click(TObject *Sender);

void __fastcall TForm1::ComboBox2Click(TObject *Sender);
```

і заповніть оператори до подій згідно листингу файлу *U_Lab*12*.cpp* .

Крок 11. Виконаєте компіляцію файлів проекту *P_Lab12.bpr* та перевірте результат роботи прикладної програми, як показано на рис. 12.5.

Листинг файлу *U_Lab12.cpp*

```
//---------------------------------------------------------------------------
#include <vcl.h>
#pragma hdrstop
#include "U_Lab12.h"
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
void UpdateFiles_1()
{
 TSearchRec sr1;
// TSHFileInfo * fi1;
 TListItem *pItem1 = NULL;
 int IconIndex1;
 TSHFileInfo * fi1 = new TSHFileInfo;
 Form1->ListView1->Items->BeginUpdate();
 Form1->ListView1->Items->Clear();
 if (FindFirst(Form1->DirectoryListBox1->Directory + "\\*.*", faAnyFile, sr1) == 0)
```

```
  do
  {
    if (sr1.Attr == faDirectory) continue;
   pItem1 = Form1->ListView1->Items->Add();
   pItem1->Caption = sr1.Name;
   SHGetFileInfo(("*" + LowerCase(ExtractFileExt(pItem1->Caption))).c_str(),
   0, fi1, sizeof(fi1),
   SHGFI_SMALLICON | SHGFI_SYSICONINDEX | SHGFI_TYPENAME);
   pItem1->ImageIndex = fi1->iIcon;
   if (sr1.Size < 1024)
    pItem1->SubItems->Add(IntToStr(sr1.Size) + " byte");
        else if (sr1.Size < 1024 * 1024)
     pItem1->SubItems->Add(IntToStr(sr1.Size / 1024) + " KByte");
        else pItem1->SubItems->Add(IntToStr(sr1.Size / (1024 * 1024)) + " MByte");
    pItem1->SubItems->Add(fi1->szTypeName);
   }
   while (FindNext(sr1) == 0);
 FindClose(sr1);
 Form1->ListView1->Items->EndUpdate();
}
//---------------------------------------------------------------------------
void UpdateFiles_2()
{
 TSearchRec sr2;
// TSHFileInfo * fi2;
 TListItem *pItem2 = NULL;
 int IconIndex2;
 TSHFileInfo * fi2 = new TSHFileInfo;
```

```
 Form1->ListView2->Items->BeginUpdate();
 Form1->ListView2->Items->Clear();
 if (FindFirst(Form1->DirectoryListBox2->Directory + "\\*.*", faAnyFile, sr2) == 0)
  do
  {
    if (sr2.Attr == faDirectory) continue;
   pItem2 = Form1->ListView2->Items->Add();
   pItem2->Caption = sr2.Name;
   SHGetFileInfo(("*" + LowerCase(ExtractFileExt(pItem2->Caption))).c_str(),
     0, fi2, sizeof(fi2),
      SHGFI_SMALLICON | SHGFI_SYSICONINDEX | SHGFI_TYPENAME);
   pItem2->ImageIndex = fi2->iIcon;
   if (sr2.Size < 1024)
    pItem2->SubItems->Add(IntToStr(sr2.Size) + " byte");
        else if (sr2.Size < 1024 * 1024)
     pItem2->SubItems->Add(IntToStr(sr2.Size / 1024) + " KByte");
        else pItem2->SubItems->Add(IntToStr(sr2.Size / (1024 * 1024)) + " MByte");
    pItem2->SubItems->Add(fi2->szTypeName);
   }
   while (FindNext(sr2) == 0);
 FindClose(sr2);
 Form1->ListView2->Items->EndUpdate();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::FormCreate(TObject *Sender)
{
 TSHFileInfo * fi1, *fi2;
 DriveComboBox1->DirList = DirectoryListBox1;
```

```
 ListView1->SmallImages->Height = 16;
 ListView1->SmallImages->Width = 16;
 DriveComboBox2->DirList = DirectoryListBox2;
 ListView2->SmallImages->Height = 16;
 ListView2->SmallImages->Width = 16;
// SHGetFileInfo("*.*", 0, fi,
//      sizeof(fi), SHGFI_SMALLICON | SHGFI_ICON | SHGFI_SYSICONINDEX);
 ListView1->SmallImages->Handle =
 SHGetFileInfo("*.*", 0, fi1,
     sizeof(fi1), SHGFI_SMALLICON | SHGFI_ICON | SHGFI_SYSICONINDEX);
 ListView1->LargeImages->Handle = SHGetFileInfo("*.*", 0, fi1,
     sizeof(fi1), SHGFI_LARGEICON | SHGFI_ICON | SHGFI_SYSICONINDEX);
 ComboBox1->ItemIndex = 0;
 ListView1->SmallImages->Handle =
 SHGetFileInfo("*.*", 0, fi2,
     sizeof(fi2), SHGFI_SMALLICON | SHGFI_ICON | SHGFI_SYSICONINDEX);
 ListView2->LargeImages->Handle = SHGetFileInfo("*.*", 0, fi2,
     sizeof(fi2), SHGFI_LARGEICON | SHGFI_ICON | SHGFI_SYSICONINDEX);
 ComboBox2->ItemIndex = 0;
 UpdateFiles_1();
 UpdateFiles_2();
}
//---------------------------------------------------------------------------
void __fastcall TForm1::DirectoryListBox1Change(TObject *Sender)
{
 UpdateFiles_1();
}
//---------------------------------------------------------------------------
```

```
void __fastcall TForm1::ComboBox1Click(TObject *Sender)
{
 switch (ComboBox1->ItemIndex)
 {
  case 0: ListView1->ViewStyle = vsIcon;
          break;
  case 1: ListView1->ViewStyle = vsSmallIcon;
          break;
  case 2: ListView1->ViewStyle = vsList;
          break;
  case 3: ListView1->ViewStyle = vsReport;
 }
}
//---------------------------------------------------------------------------

void __fastcall TForm1::ComboBox2Click(TObject *Sender)
{
  switch (ComboBox2->ItemIndex)
 {
  case 0: ListView2->ViewStyle = vsIcon;
          break;
  case 1: ListView2->ViewStyle = vsSmallIcon;
          break;
  case 2: ListView2->ViewStyle = vsList;
          break;
  case 3: ListView2->ViewStyle = vsReport;
 }
}
```

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

void __fastcall TForm1::DirectoryListBox2Change(TObject *Sender)
{
  UpdateFiles_2();
}
//---------------------------------------------------------------------------
```

Листинг файлу ***U_Lab12.h***

```
//---------------------------------------------------------------------------
#ifndef U_Lab12H
#define U_Lab12H
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ComCtrls.hpp>
#include <FileCtrl.hpp>
#include "ShellAPI.h"
#include "shlwapi.h"
#include <ImgList.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published:        // IDE-managed Components
      TListView *ListView1;
      TDirectoryListBox *DirectoryListBox1;
```

```
        TDriveComboBox *DriveComboBox1;
        TComboBox *ComboBox1;
        TImageList *ImageList1;
        TImageList *ImageList2;
        TLabel *Label1;
        TDriveComboBox *DriveComboBox2;
        TDirectoryListBox *DirectoryListBox2;
        TListView *ListView2;
        TLabel *Label2;
        TComboBox *ComboBox2;
        void __fastcall FormCreate(TObject *Sender);
        void __fastcall DirectoryListBox1Change(TObject *Sender);
        void __fastcall ComboBox1Click(TObject *Sender);
        void __fastcall ComboBox2Click(TObject *Sender);
        void __fastcall DirectoryListBox2Change(TObject *Sender);
private:                // User declarations
public:                 // User declarations
        __fastcall TForm1(TComponent* Owner);
};

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

<u>Контрольні запитання до завдання № 12</u>

1. Покажіть, як налаштувати у програмі переміщення файлу або папки.
2. Покажіть необхідні налаштування у програмі для копіювання файлу.
3. Додайте у програму індикатори для візуального контролю процесу переміщення об’єкта.

4. Додайте у програму індикатори для візуального контролю процесу копіювання файлу.

5.Покажіть, яким чином можна у програмі змінювати налаштування розмірів значків.

Література

1. Архангельский А. Я. Программирование в C++Builder6 [Текст] /А. Я. Архангельский - М. : ЗАО “Издательство БИНОМ”, 2002. – 1152 с. Библиграф. : с. 1149 - 1150. 4000 экз.
2. Архангельский А. Я. C++Builder 6. Справочное пособие. Книга 1. Язык C++ [Текст] /А. Я. Архангельский – М.: Бином-Пресс, 2002. – 544 с. Библиграф. : с. 543. 4000 экз.
3. 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.
4. Культин Н.Б. Самоучитель C++ Builder [Текст] /Н.Б. Культин – СПб.: БХВ-Петербург, 2004. – 320 с. Библиграф. : с. 318. 4000 экз.