

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ТЕХНОЛОГІЇ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

**ПРАКТИКУМ ПО ВИКОРИСТАННЮ КОМПОНЕНТИ MAINMENU
І ОБРОБКИ ПОДІЙ МИШКИ В C++ BUILDER ПРИ ВІЗУАЛЬНОМУ
ПРОГРАМУВАННІ ПРИКЛАДНОЇ ПРОГРАМИ ДО МНEMОСХЕМИ
ТЕХНОЛОГІЧНОГО ПРОЦЕСУ З ХІМІЧНОГО ВИРОБНИЦТВА**

*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
2018

Технології розробки програмного забезпечення. Практикум по використанню компоненти *MainMenu* і обробки подій мишки в *C++ Builder* при візуальному програмуванні прикладної програми до мнемосхеми технологічного процесу з хімічного виробництва. [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: 2,04 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 135 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 9 від 24.05.2018 р.) за поданням Вченої ради Інженерно-хімічного факультету (протокол № 4 від 24.04.2018 р.)

Електронне мережне навчальне видання

ТЕХНОЛОГІЇ РОЗРОБКИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПРАКТИКУМ ПО ВИКОРИСТАННЮ КОМПОНЕНТИ MAINMENU І ОБРОБКИ ПОДІЙ МИШКИ В C++ BUILDER ПРИ ВІЗУАЛЬНОМУ ПРОГРАМУВАННІ ПРИКЛАДНОЇ ПРОГРАМИ ДО МНЕМОСХЕМИ ТЕХНОЛОГІЧНОГО ПРОЦЕСУ З ХІМІЧНОГО ВИРОБНИЦТВА

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. Практикум по використанню компоненти *MainMenu* і обробки подій мишки в *C++ Builder* при візуальному програмуванні прикладної програми до мнемосхеми технологічного процесу з хімічного виробництва» створено для проведення практичних занять з розробки *C++* програми до модульної контрольної роботи кредитного модуля «Візуальне програмування прикладних програм» навчальної дисципліни «Технології розробки програмного забезпечення». Відповідно до робочої програми навчальної дисципліни у навчальному посібнику розглядаються основні компоненти *C++ Builder* та методики і техніка їх використання для розробки і візуального програмування прикладних програм на комп'ютері в інтегрованому програмувальному середовищі.

© КПІ ім. Ігоря Сікорського, 2018

Зміст

Вступ	4
1. Техніка використання і програмування компоненти <i>Form</i> в <i>C++ Builder</i>	11
2. Алгоритми програмування і обробки рисунків у вікні компоненти <i>Form</i>	17
2.1 Техніка програмування обробки компонент <i>VLC</i> встановлених на полі вікна компоненти <i>Form</i>	20
2.2 Приклад програмування обробки подій до компонент у вікні <i>Form</i>	29
3. Правила використання компоненти <i>MainMenu</i> для програмування меню команд прикладної <i>C++</i> програми	38
3.1 Приклад розробки та програмування обробки подій до меню команд на основі компоненти <i>MainMenu</i>	42
4. Навчальний приклад до розробки і програмування прикладної <i>C++</i> програми для модульної контрольної роботи кредитного модуля	46
4.1 Постановка завдання до навчального прикладу з виконання модульної контрольної роботи	48
4.2 Приклад з розробки матеріалів до етапу виконання завдань модульної контрольної роботи	52
4.2.1 Навчальний приклад. Схема і опис технологічного процесу виробництва хлорметанів	53
4.2.2 Навчальний приклад до розробки і оформлення матеріалів для меню «Апарати»	65
5. Навчальний приклад до правил використання компоненти <i>MainMenu</i> для створення меню команд у прикладній <i>C++</i> програмі	73
5.1 Навчальний приклад з програмування функцій для виконання команд у меню «Технологія»	88
5.2 Навчальний приклад з програмування функцій для виконання команд у меню «Апарати»	92
5.3 Навчальний приклад з програмування функцій для виконання команд у меню «Інформація»	100
5.4 Навчальний приклад з програмування функцій для виконання команд у меню «Вихід»	103
6. Файли <i>C++</i> програми до навчального прикладу з модульної контрольної роботи	104
6.1 Структура головного файлу проекту до навчального прикладу з модульної контрольної роботи	104
6.2 Структура файлів програмних модулів віконних форм	108
6.3 Компіляція файлів проекту і компонування виконавчого коду <i>C++</i> програми до навчального прикладу модульної контрольної роботи	124
7. Література	131
8. Додатки: Д1, Д2, Д3, Д4	132

ВСТУП

Навчальний посібник «*Технології розробки програмного забезпечення. Практикум по використанню компоненти MainMenu і обробки подій мишки в C++ Builder при візуальному програмуванні прикладної програми до мнемосхеми технологічного процесу з хімічного виробництва*» складено для набуття студентами практичних навичок з розробки та програмування прикладних програм у процесі проведення практичних занять до кредитного модуля «Візуальне програмування прикладних програм» навчальної дисципліни «Технології розробки програмного забезпечення». У навчальному посібнику розглядаються основні компоненти *C++ Builder* за допомогою яких потрібно студентам виконати модульну контрольну роботу (МКР) до кредитного модуля зі створення прикладної *C++* програми згідно індивідуального завдання.

Знання та практичні навички з кредитного модуля навчальної дисципліни закріплюються за допомогою виконання студентами індивідуальних завдань до модульної контрольної роботи на основі створення в *C++ Builder*^[1] прикладної програми з меню команд, побудованого на основі компоненти *MainMenu*. Практичні заняття до кредитного модуля «Візуальне програмування прикладних програм» навчальної дисципліни «Технології розробки програмного забезпечення - 2» передбачають можливість бакалаврам за рахунок виконання практичних завдань отримувати наступні знання:

- з техніки перетворення у графічному редакторі фотографії з зображенням копії технологічної схеми процесу відповідного хімічного виробництва у рисунок мнемосхеми технологічного процесу і у комп'ютерне креслення технологічної схеми процесу для розробки схеми автоматизації процесів до завданого хімічного виробництва;
- зі створення у графічному редакторі рисунка з конструкції технологічного апарату № 1 для використання у прикладній *C++* програмі до МКР з кредитного модуля навчальної дисципліни;

- зі створення у графічному редакторі рисунка з конструкції технологічного апарату № 2 для використання у прикладній C++ програмі до МКР з кредитного модуля навчальної дисципліни;
- з правил і методики використання в C++ *Builder* компоненти *MainMenu* для створення і програмування меню команд до прикладної C++ програми у інтегрованому програмувальному середовищі;
- з методики і техніки та алгоритмів до програмування в C++ *Builder* обробки подій мишки дії прикладної C++ програми, створюваної в інтегрованому програмувальному середовищі.

З метою контролю рівня засвоєння матеріалу та сприйняття його студентами на протязі навчального семестру знання і уміння бакалаврів з кредитного модуля “Візуальне програмування прикладних програм” навчальної дисципліни “Технології розробки програмного забезпечення –2” контролюються по результатах виконання модульної контрольної роботи № 2 і які оцінюються відповідно до положення з рейтингової системи оцінки знань студента.

Модульна контрольна робота № 2 кредитного модуля “Візуальне програмування прикладних програм” виконується студентами на тему: «Обробка подій мишки у меню команд прикладної C++ програми до мнемосхеми технологічного процесу (*назва хімічного виробництва*)» і повинна мати наступні матеріали:

- виконуючий файл прикладної C++ програми з основним меню команд і залежними підменю команд, створеними на основі компоненти *MainMenu*;
- файли проекту *Project_MKP-2.bpr* до прикладної C++ програми «Обробка подій мишки у меню команд прикладної C++ програми до мнемосхеми технологічного процесу (*назва хімічного виробництва*)»;
- пояснювальну записку до модульної контрольної роботи № 2 з матеріалами і алгоритмами з обробки подій мишки при обранні

відповідних команд у меню команд прикладної C++ програми та на зображених технологічних апаратах мнемосхеми технологічного процесу з хімічного виробництва.

Для модульної контрольної роботи № 2 кредитного модуля “Візуальне програмування прикладних програм” до різних процесів з хімічних виробництв кожному студенту у навчальній групі видається індивідуальне завдання на тему: **«Обробка подій мишки у меню команд прикладної C++ програми до мнемосхеми технологічного процесу (назва хімічного виробництва)»**.

Індивідуальні завдання студентам з розробки і програмування прикладної C++ програми до МКР № 2 мають такі формулювання:

- Необхідно для роботи у *Windows* розробити з використанням компоненти *MainMenu* прикладну C++ програму з основним меню команд та підменю команд, які повинні мати структуру команд відповідно до зображення на рис. В-1;
- Після запуску виконуючого файлу прикладної C++ програми на екрані дисплея комп'ютера повинно з'явитися основне вікно програми з зображенням заставки на якій мають бути виконані написи відповідно до рис. В-2;
- Якщо у меню “Технологія” мишкою обирається команда “Мнемосхема”, тоді у цьому випадку заставка C++ програми змінюється на зображення рисунка мнемосхеми до завданого відповідного технологічного процесу, яке далі стає фоном основного вікна прикладної програми та знизу вікна з'являються написи підказок до назв технологічних апаратів, на зображенні яких встановлюється або переміщується курсор мишки;
- При виконанні команд:
 - ❖ Технологія/ Опис процесу;
 - ❖ Технологія/ Продукція

інформація у прикладній C++ програмі до цих команд повинна показуватися на дисплею комп'ютера в окремому вікні *Windows*;

➤ Якщо у меню “Технологія” виконується команда “Опис процесу”, то в цьому випадку в окремому вікні *Windows* повинен з’явитися текст опису процесів до технологічних апаратів, зображених на мнемосхемі, рисунок якої вставлено у основне вікно прикладної *C++* програми. Показ тексту з опису технологічних процесів до зображення мнемосхеми у вікні *Windows* виконується за допомогою компоненти *RichEdit*;

➤

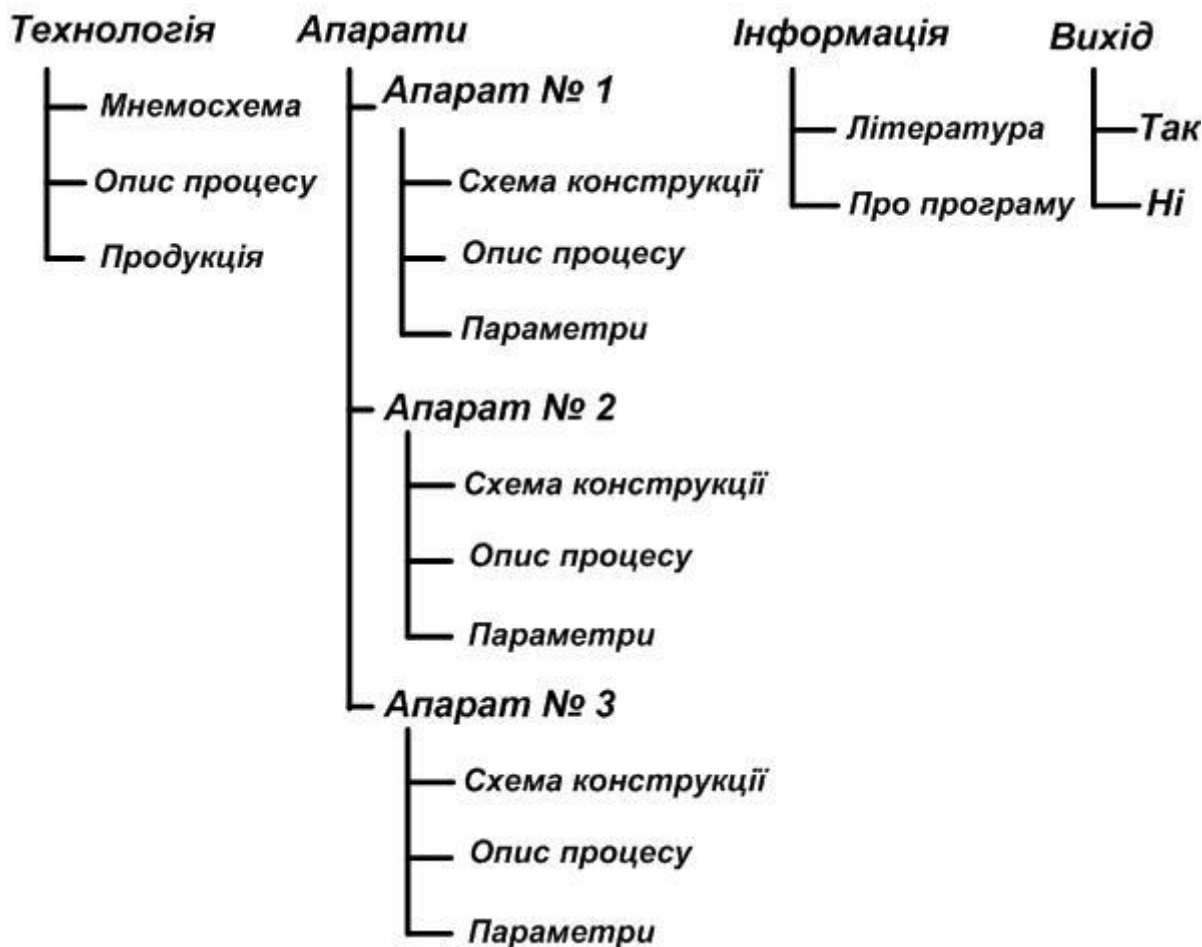


Рис. В-1. Меню команд прикладної *C++* програми до модульної контрольної роботи № 2.

➤ Якщо у меню “Технологія” виконується команда “Продукція”, тоді в цьому випадку в окремому вікні *Windows* повинен з’явитися текст опису властивостей і призначення продукції з хімічного виробництва, яке завдано до МКР № 2. Для показу тексту з опису продукції хімічного виробництва використовується компонента *RichEdit*;

➤ При виконанні команд:

❖ Апарати/Апарат № / Конструкція;

❖ Апарати/Апарат № / Опис процесу;

❖ Апарати/Апарат № / Параметри

інформація у прикладній C++ програмі до цих команд повинна показуватись на дисплею комп'ютера в окремому вікні *Windows*;

**Модульна контрольна робота кредитного модуля
“Візуальне програмування прикладних програм”
навчальної дисципліни “Технології розробки програмного забезпечення – 2”**

**на тему: «Обробка подій мишки у меню команд прикладної
C++ програми до мнемосхеми технологічного
процесу (назва хімічного виробництва)»**

Розробник C++ програми студент(ка)

П.І.Б.

ІХФ гр. ЛА-№ XX

Київ

КПІ ім. Ігоря Сікорського

20_____р.

Рис. В-2. Написи для заставки прикладної C++ програми модульної контрольної роботи.

- Коли у меню “Апарати” виконується команда “Апарат № / Конструкція” тоді у окремому вікні *Windows* повинно з’явитися зображення відповідного рисунка до конструкції технологічного апарату, назва якого вказана у меню обраної мишкою команди.
- Для показу зображення відповідного рисунка конструкції технологічного апарату з мнемосхеми технологічного процесу хімічного виробництва використовується компонента *Image*.
- Коли у меню “Апарати” виконується команда “Апарат № / Опис процесу” тоді у вікні *Windows* потрібно показати текст опису особливостей технологічного процесу до схеми конструкції завданого апарату з хімічного виробництва. Для показу тексту з опису процесу до схеми апарату у вікні *Windows* використовується компонента *RichEdit* ;

- Коли у меню “Апарати” виконується команда “Апарат № / Параметри” тоді у вікні *Windows* показуються графіки зміни у часі технологічних параметрів процесу до обраного технологічного апарату. Для побудови даних графіків у *C++ Builder* передбачено використання компоненти *Chart*;
- Результати до розробок алгоритмів і програмування прикладної *C++* програми для МКР № 2, описуються і оформлюються у вигляді пояснювальної записки, яка для захисту модульної контрольної роботи подається викладачеві на 8-му тижні навчального семестру і повинна мати наступний зміст:

	<u>Стор.</u>
Зміст	
1. Завдання до модульної контрольної роботи № 2	2
2. Меню команд та інформація до програми «Обробка подій мишки у меню команд прикладної <i>C++</i> програми до мнемосхеми технологічного процесу(назва хімічного виробництва)»	3
2.1 Структура команд та інформація до меню команд <i>C++</i> програми (показується рисунок меню команд та відповідна інформація, яка повинна виводитися у вікні до обраних мишкою команд у меню команд)	3
2.2 Опис алгоритму до обробки подій мишки при встановленнях і зміщеннях курсору на зображенні мнемосхеми технологічного процесу(рисунок блок-схеми алгоритму на аркушу формату А3)	5
3. Структура віконних <i>Forms</i> , які використовуються у <i>C++</i> програмі та алгоритми їх відкриття і закриття (рисунок блок-схеми алгоритму на аркушу формату А3).....	8
4. Лістинги програмних модулів прикладної <i>C++</i> програми до МКР - № 2	12
5. Література	20
6. Додаток:	
6.1 Виконуючий файл і файли проекту на диску CD-R до прикладної програми «Обробка подій мишки у меню команд прикладної <i>C++</i> програми до мнемосхеми технологічного процесу (назва хімічного виробництва)»	21

Інтегроване програмувальне середовище *C++ Builder* має для програмування прикладних програм багато бібліотечних компонент, які утворюють бібліотеку візуальних компонент (*Visual Component Library – VCL*).

Даний навчальний посібник написано для практичного виконання студентами модульної контрольної роботи до кредитного модуля навчальної дисципліни «Технології розробки програмного забезпечення», тобто потрібно створити прикладну **C++** програму з меню команд, яке показано на рис. В-1. Тому далі розглянемо які компоненти з **C++ Builder** потрібні для розробки і створення прикладної **C++** програми з відповідним меню команд згідно рис. В-1. Для розробки і програмування **C++** програми до МКР-2 потрібно використовувати такі компоненти:

- *Form* (форма) є вікном *Windows* з полем покритому крапками сіточки і на якому встановлюються та закріплюються усі необхідні компоненти для створення відповідних зображень і обробки подій при обранні мишкою команди у меню команд прикладної **C++** програми;
- *MainMenu* ця компонента є конструктором для створення відповідної структури команд у основному меню команд та і у залежних команд – підменю. За допомогою *MainMenu* в прикладній **C++** програмі до МКР-2 потрібно створити структуру меню команд відповідно до рис. В-1;
- *Image* ця компонента потрібна для створення і програмування подій при виконанні у **C++** програмі наступних команд: Технологія/Мнемосхема, Апарати/Апарат № 1/Схема конструкції, Апарати/Апарат № 2/Схема конструкції;
- *RichEdit* цю компоненту потрібно використовувати на формах вікон для показу у програмі текстів з описами до наступних коман: Технологія/Опис процесів, Апарати/Апарат № 1/Опис процесу, Апарати/Апарат № 2/Опис процесу, Інформація/Література, Інформація/Про програму;
- *Chart* цю компоненту необхідно встановлювати на фому вікна *Windows* для програмування подій при виконанні у програмі таких

команд: Апарати/Апарат № 1/Параметри, Апарати/Апарат № 2 /Параметри;

- *Label* цю компоненту необхідно встановлювати на фому вікна *Windows*, щоби показувати написи підказок до назв технологічних апаратів на рисунку мнемосхеми технологічних процесів з хімічного виробництва.

Бібліотечні компоненти *C++ Builder* є шаблонами інструментів за допомогою яких можливо створювати інтерфейси для користувачів у прикладних програмах різного призначення. Інтерфейс у прикладній *C++* програми до модульної контрольної роботи № 2 визначено структурою меню команд, яка показана на зображенні рис. В-1. Тексти до описів процесів у технологічних апаратах з мнемосхеми потрібно набирати у програмі *Word*. Рисунки схем конструкції апаратів та рисунок мнемосхеми технологічних процесів до відповідного хімічного виробництва необхідно створювати у комп'ютерному графічному редакторі і такими можуть бути наступні редактори: PAINT, VISIO, GIMP, POTOSHOP та інші.

1. ТЕХНІКА ВИКОРИСТАННЯ І ПРОГРАМУВАННЯ КОМПОНЕНТИ *FORM* В *C++ BUILDER*

Візуальне програмування в *C++ Builder* прикладної програми для роботи у *Windows* передбачає створення набору файлів, який буде мати назву проекту програми і кількість файлів залежить від кількості компонент *Form* потрібних для програмування роботи *C++* програми^[1]. Проект файлів до прикладної *C++* програми завжди має один головний файл з ім'ям *Project.bpr* та відповідні до модулів форм файли з назвами *Unit.cpp*, кількість яких залежить від кількості компонент *Form*, необхідних для роботи у *Windows* прикладної програми. Файли *Unit.cpp* та *Project.bpr* до проекту програми утворюються автоматично інтегрованим програмувальним середовищем *C++ Builder* і для програміста пропонується їх обов'язкове збереження на диск. Зміст операторів головного файлу проекту до прикладної програми *C++ Builder* може показати, якщо у меню

команд виконати команду *Project | View Source* тоді у вікні редактора коду програмувального інтегрованого середовища буде показано наступний текст

```
//-----  
# include <vlc.h >  
# pragma hdrstop  
//-----  
USERES( "Project1.res" );  
USERFORM( "Unit1.cpp", Form1 );  
//-----  
WINAPI WinMain( HINSTANCE, HINSTANCE, LPSTR, int )  
{  
    try  
    { Application->Initialize( );  
      Application->CreateForm( _classid( TForm1 ), &Form1 );  
      Application->Run( );  
    }  
    catch(Exception & exception )  
    {  
        Application->Show Exception( & exception );  
    }  
    return 0;  
}  
//-----
```

В строках, вище наведеного тексту головного файлу проекту, передбачено вказівку препроцесору `# include <vlc.h >` про підключення текстів з описами об'єктів з *vlc* (візуальної бібліотеки компонент) при компіляції файлів проекту прикладної **C++** програми. Також у строку `USERFORM("Unit1.cpp", Form1);` записано, що буде використовуватися одна компонента *Form1* та для неї існує один файл модуля форми *Unit1.cpp*. Керувати роботою прикладної **C++** програми буде головна функція *WinMain()* і про це записано у строку

```
WINAPI WinMain( HINSTANCE, HINSTANCE, LPSTR, int );
```

Оператор `Application->Initialize()`; з головної функції показує, що буде виконуватися ініціалізація об'єктів до компонент запрограмованої прикладної програми. Виконуючий оператор головної функції

Application->CreateForm(_classid(TForm1), &Form1); показує що на початку роботи прикладної C++ програми утворюються об'єкти до компоненти *Form1* і далі починається виконання програми за допомогою оператора Application->Run(); .

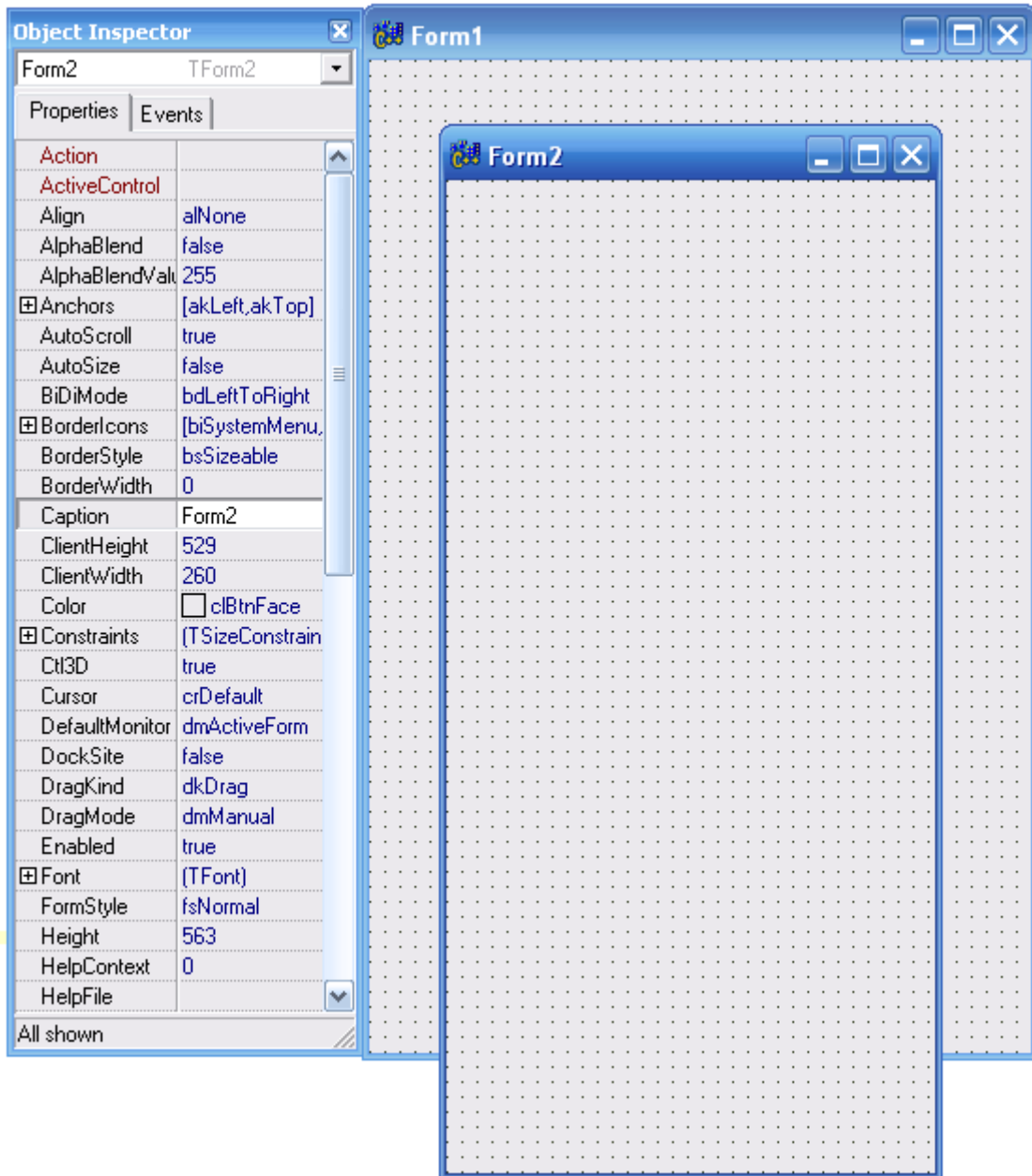


Рис. 1-1 Зображення компонент *Form1* та *Form2* та список властивостей (Properties).

Компонента *Form1* завжди автоматично утворюється при запуску у роботу інтегрованого програмувального середовища C++ *Builder* і зображення такої форми для програмування вікна *Windows* прикладної програми показано на рис. 1-1. При необхідності у прикладну програму можна додати додаткові компоненти *Form* за допомогою обрання на панелі швидких кнопок інструментів

кнопку *New Form*, а також у меню команд *C++ Builder* виконати команди *File | New* і після якої відкриється вікно з заголовком *New Item*, де необхідно обрати закладку *Forms*. Наприклад, на рис. 1-1 кнопкою *New Form* додано компоненту форми *Form2*. Вікно стартової форми *Form1* потрібно використовувати в прикладній програмі до МКР-2 при обранні у меню команд (рис. В-1) таких команд «Технологія / Мнемосхема», а при обранні наступних команд «Технологія / Опис процесу» використовувати у програмі вікно форми *Form2*.

На рис. 1-1 у вікні *Object Inspector* на закладці *Properties* показується список властивостей до активного вікна *Form2*. Деякі основні властивості вікон *Form* наведені у таблицях табл. 1-1, табл. 1-2 та табл. 1-3.

Властивість *BorderStyle* визначає загальний вигляд форми (вікна при роботі програми) і операції які дозволяються для дій користувача з вікном. У таблиці 1-1 наведені значення на які можна налаштовувати властивість *BorderStyle*.

Таблиця № 1-1.

Значення властивості	Властивість, яка буде встановлена для вікна форми
bsSizeable	Звичайний вигляд вікна <i>Windows</i> з смужкою заголовка та можливістю для користувача змінювати розміри вікна за допомогою кнопок в смужці заголовку або за допомогою мишки, потягнувши за будь-якій край вікна. Це значення задається за замовчуванням.
bsDialog	Незмінне за розмірами вікно. Типове вікно для діалогів.
bsSingle	Вікно, розмір якого користувач не може змінити. Потягнувши курсором мишки край вікна, але може змінювати кнопками в смужці заголовка.
bsToolWindow	Те ж, що і <i>bsSingle</i> , але зі смужкою заголовка меншого розміру.
bsSizeToolWin	Те ж, що і <i>bsSizeable</i> , але зі смужкою заголовка меншого розміру і з відсутністю в ній кнопок для зміни розмірів.
bsNone	Без смужки заголовка. Вікно не допускає зміну розміру, але і не дозволяє перемістити його по екрану. Це бажано на використовувати.

В *C++ Builder* компонента *Form* є базовою і яку потрібно розуміти як типове вікно *Windows* на яке встановлюються і закріплюються та програмуються усі другі компоненти. Під час програмування компонента *Form* має сітку у вигляді крапок. Вікно *Form* має також властивості, які визначені для

вікон *Windows* в залежності від налаштувань та версії системи *Windows*, встановленої на комп'ютер. Вікно *Form* має смужку заголовку, де з лівої сторони розташовано меню керування, а з правої сторони кнопки згортання та розгортання вікна, а також кнопка для закриття вікна.

У списку вікна *Object Inspector* властивість *BorderIcons* визначає властивості, які наведено у таблиці 1-2 до кнопок, розташованих на смужці у заголовку вікна *Windows*.

Таблиця № 1-2.

Значення властивості	Властивість яка буде встановлена для вікна форми
bySystemMenu	Кнопка системного меню (це кнопка з хрестиком, що закриває вікно).
byMinimize	Кнопка «Згорнути», згортає вікно до піктограми.
byMaximize	Кнопка «Розгорнути», розгортає вікно на весь екран.
byHelp	Кнопка довідки.

Властивість форми *Windows State* визначає вигляд вікна на початку роботи прикладної програми. Значення до властивості *Windows State* наведені у таблиці № 1-3.

Таблиця № 1-3.

Значення властивості	Властивість яка буде встановлена для вікна форми
wsNormal	Нормальний вигляд вікна і використовується за замовчуванням.
wsMinimized	Вікно згорнуто.
wsMaximized	Вікно розгорнуто на повний екран дисплею.

Якщо властивість *Windows State* має значення *wsNormal* тоді положення вікна прикладної програми на екрані дисплею буде залежати від значення властивості *Position*, які наведені у таблиці № 1-4.

Таблиця № 1-4.

Значення властивості	Властивість яка буде встановлена для вікна форми
poDesigned	Початкові розміри і положення вікна під час виконання ті ж, що і під час проектування. Це значення приймається за замовчуванням, але зазвичай його слід змінити.
poScreenCenter	Вікно розташовується в центрі екрана. Розмір вікна той, який був спроектований для використання на багатьох моніторах. Вікно точно розташується на одному моніторі, з урахуванням якості <i>DefaultMonitor</i> .
poDesktopCenter	Вікно розташовується в центрі екрану. Розмір вікна той, який був спроектований. Не використовувати для роботи програми з безліччю моніторів.
poDefault	Місце знаходження і розмір вікна визначає система <i>Windows</i> , з огляду на розмір і дозвіл екрана монітора. При послідовних показах вікна його положення зсувається трошки вниз і вправо.
poDefaultPosOnly	Місце знаходження і розмір вікна визначає система <i>Windows</i> . При послідовних показах вікна його положення зсувається трошки вниз і вправо. Розмір вікна встановлюється спроектований.
poDefaultSizeOnly	Розмір вікна визначає система <i>Windows</i> , з огляду на розмір і дозвіл екрана монітора. Положення вікна встановлюється спроектоване.
poMainFormCenter	Вікно розташовується в центрі головної форми. Розмір вікна встановлюється спроектований. Використовується для додаткових форм програми. Для головної форми діє також, як <i>poScreenCenter</i> .
poOwnerFormCenter	Вікно розташовується в центрі форми. Діє як <i>poMainFormCenter</i> .

У таблиці № 1-5 наведені властивості, які майже завжди налаштовуються до компоненти *Form*.

Таблиця № 1-5.

Name	Ім'я форми з порядковим номером. У програмі ім'я форми використовується для управління формою і для доступу до розташованих на формі компонентів.
Caption	Текст до назви заголовка форми.
Width	Розмір ширини форми.
Height	Розмір висоти форми.
Top	Відстань від верхньої межі форми до верхньої межі екрану.
Left	Відстань від лівої межі форми до лівої межі екрану.
Icon	Значок в заголовку діалогового вікна, що позначає кнопку виведення системного меню.
Color	Колір фону форми.
Font	Шрифт, який використовується за умовчанням компонентами, що знаходяться на поверхні форми.
Canvas	Поверхня, на яку можна вивести графіку.

Уся інформація, яка визначає властивості і допустимі дії форми при роботі в *Windows* прикладної *C++* програми записано в трьох файлах проекту: файл з розширенням *.cpp* модуля форми *Unit*, заголовний файл з розширенням *.h*; файл з розширенням *.dfm* і при копіюванні віконних форм з різних проектів потрібно це враховувати.

2. АЛГОРИТМИ ПРОГРАМУВАННЯ І ОБРОБКИ РИСУНКІВ У ВІКНІ КОМПОНЕНТИ *FORM*

Більшість прикладних *C++* програм при роботі у *Windows* на полі вікна *Form* мають вбудоване графічне зображення. При розробках *C++* програм можна рисунок на форму вставити і закріпити за допомогою компоненти *Image*, яка вибирається за допомогою закладки *Additional* на сторінках палітри компонент бібліотеки *VLC* (рис. 2-1).

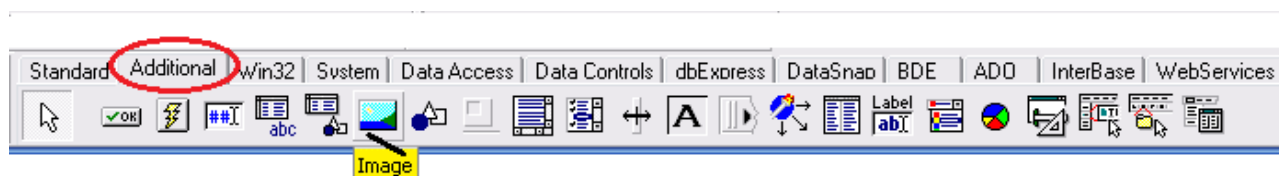


Рис. 2-1. Сторінки палітри компонент бібліотеки *VLC* з активною закладкою *Additional*.

Для встановлення компоненти достатньо лівою клав'яшею мишки потягнути значок *Image* на поле вікна *Form* і відразу ж з'явиться маркерная рамка з повідомленням назви та розміру встановленої компоненти *Image*. За кутовий елемент маркерної рамки компоненти *Image* можна лівою клав'яшею мишки збільшувати або зменшувати допустимий розмір для вбудовування рисунка на вікно *Form*.

Після подвійного щиклика мишкою у полі маркерної рамки, встановленої компоненти *Image* у вікно *Form*, відкривається діалогове вікно для вибору кнопкою *Load* необхідного файлу з рисунком. Якщо файл обраний, то в цьому випадку автоматично рисунок фіксується у вікні на *Form* і його місце розташування визначається координатами компоненти *Image* при переміщеннях

мишкою маркерної рамки на полі *Form*. Після завантаження рисунка з файлу у компоненту *Image* цей рисунок не тільки відображається компонентою, але і зберігається в прикладній програмі [2].

Компонента *Image* має відповідні властивості, які можна переглянути у вікні *Object Inspector* на закладці *Properties* (табл. № 2-1).

Таблиця № 2-1.

Значення властивості	Властивість яка буде встановлена для компоненти <i>Image</i>
AutoSize	Вказує, чи змінюється автоматично розмір компоненти, підлаштовуючись під розмір зображення рисунка. За замовчуванням <i>false</i> - не підлаштовувати.
Canvas	Визначає поверхню (полотно, канву) для малювання пером <i>Pen</i> і пензлем <i>Brush</i> , а також для накладення друг на друга декількох зображень. Доступно тільки для читання. Доступна властивість тільки, якщо у властивості <i>Picture</i> зберігається бітова матриця.
Center	Вказує, чи повинно зображення центруватися у полі компоненти, якщо розмір малюнка менше розмірів поля компоненти. При значенні <i>false</i> зображення малюнка розташовується у верхньому лівому кутку поля компоненти. Властивість не діє, якщо <i>AutoSize</i> встановлено в <i>true</i> і якщо <i>Stretch</i> встановлено в <i>true</i> і <i>Picture</i> містить не піктограму.
Incremental Display	Вказує, чи повинно зображення частково малюватися під час повільних операцій з великими зображеннями. Замість такого малювання часто можна використовувати індикацію процесу обробкою подій <i>OnProgress</i> .
Picture	Визначає, що відображає графічний об'єкт типу <i>TPicture</i> . Може завантажуватися з програми або під час проектування за допомогою <i>Picture Editor</i> .
Stretch	Вказує, чи повинні змінюватися розміри зображення, підлаштовуючись під розміри компоненти. Треба враховувати, що зміна розмірів зображення призведе до спотворення рисунка, якщо співвідношення сторін графічного зображення компоненти <i>Image</i> не однакові.
Transparent	Вказує, чи повинен бути колір фону зображення прозорим, щоб крізь нього було видно нижче розташоване зображення.

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

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

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

Компонента *Image* основну властивість *Picture* активізує у вікні інспектора об'єкта або на формі подвійним щигликом мишки на значку компоненти *Image*. Якщо установити властивість *AutoSize* у значення *true*, то розмір компоненти *Image* буде автоматично налаштовуватися під розмір завантаженого в неї графічного зображення. Якщо ж властивість *AutoSize* установити в значення *false*, то зображення рисунку може не поміститися в компоненту, або площа компоненти буде на багато більше площі зображення рисунка.

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

Установка в *true* властивості - *Center*, центрує зображення на полі *Image*, якщо розмір компоненти більше розміру рисунка ^[3].

Прозорість рисунка визначається властивістю - *Transparent*. Якщо *Transparent* дорівнює *true*, тоді зображення стає прозорим тільки для бітових матриць.

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

2.1 ТЕХНІКА ПРОГРАМУВАННЯ ОБРОБКИ КОМПОНЕНТ VLC ВСТАНОВЛЕНИХ НА ПОЛІ ВІКНА КОМПОНЕНТИ FORM

В модульній контрольній роботі № 2 до кредитного модуля «Візуальне програмування прикладних програм» згідно до постановки завдань до розробки прикладної *C++* програми у вікно основної форми *Form1* спочатку потрібно завантажити рисунок з заставкою і написами тексту відповідно до рис. В-2, а потім в туж саму компоненту *Image* на *Form1* необхідно завантажити рисунок мнемосхеми технологічного процесу з відповідного хімічного виробництва. Тому далі по кроках будемо розглядати, як можна запрограмувати обробку рисунків у вікні компоненти *Form1*.

Крок 1. Для розробки нової прикладної *C++* програми запускаємо у роботу інтегроване програмувальне середовище *C++ Builder*, яке автоматично утворює стартове вікно форми *Form1* і показує на дисплею. Перейдемо на форму до нової програми і для цього робимо щиглика мишкою на сіточки поля *Form1* у результаті зліва на екрані стане активним вікно *Object Inspector*, де у списку обираємо властивість *Caption* та напишемо назву «Оброка рисунка № 1 та

рисунка № 2», щоб цей текст з'явився у заголовку вікна форми *Form1*. Далі необхідно виконати такі дії:

- Виконуємо у меню команд *C++ Builder* команду *File | Save Project As...* для утворення проекту файлів до нової *C++* програми. На диску *D:* створюємо папку *PIC_1&2* і в цю папку зберігаємо до проекту файл з такою назвою *P_pic1#pic2.bpr* та файл до модуля форми *U_pic_1&2.cpp*. У результаті збереження файлів *.bpr* та *.cpp* інтегроване програмувальне середовище *C++ Builder* створює новий проект файлів до прикладної програми «Обробка рисунка № 1 та рисунка № 2».

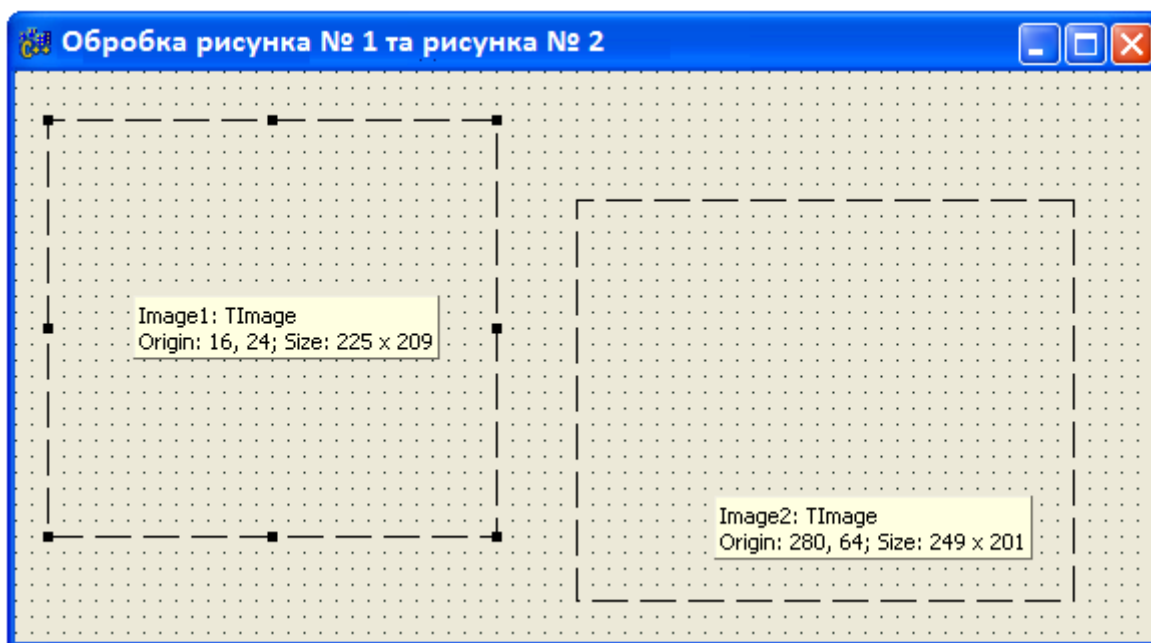


Рис. 2-2. Зображення вікна *Form1* з встановленими компонентами *Image1* та *Image2*.

Крок 2. В бібліотеці компонент на сторінці *Additional* (рис. 2-1) обираємо і розмістимо на форму вікна *Form1* компоненту *Image1* і маркерною рамкою визначимо розмір під рисунок № 1.

Крок 3. У бібліотеці компонентів *VCL* на сторінці *Additional* обираємо компоненту *Image2* і розміщуємо на форму *Form1* згідно до рис. 2-2 та маркерною рамкою визначаємо розмір полю для другого рисунку.

Крок 4. Потрібно відкрити вікно діалогу для вибору файлу з рисунком до *Image1* і для цього потрібно виконати наступні дії:

- На компоненті *Image1* робимо одного щиклика мишкою у результаті стане активною маркерная рамка компоненти і в окні *Object Inspector* буде покано список властивостей до *Image1*;
- Далі у вікні інспектора об'єктів у властивості *Picture* натискаємо кнопку з крапками, щоб активізувалося на екрані вікно "*Picture Editor*" (рис. 2-3) і потім за допомогою кнопки *Load* робимо вибір файлу із рисунком *athena.bmp* з наступних папок *Program Files \ Common Files \ Borland Shared \ Images \ Splash \ 16Color*.

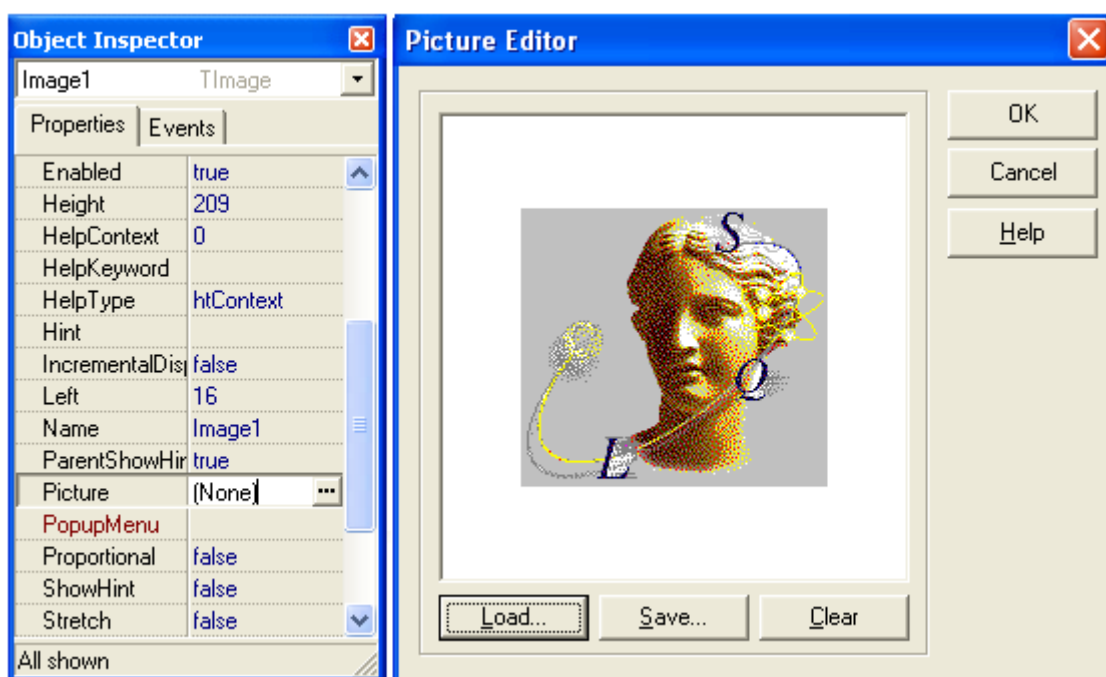


Рис. 2-3.

Крок 5. Потрібно відкрити вікно діалогу для вибору файлу з рисунком до *Image2* і для цього потрібно виконати наступні дії:

- На компоненті *Image2* робимо одного щиклика мишкою у результаті стане активною маркерная рамка компоненти і в окні *Object Inspector* буде покано список властивостей до *Image2*;
- Далі у вікні інспектора об'єктів у властивості *Picture* натискаємо кнопку з крапками, щоб активізувалося на екрані вікно "*Picture Editor*" і потім за допомогою кнопки *Load* робимо вибір файлу із рисунком *earth.bmp* з

наступних папок *Program_Files \ Common_Files \ Borland Shared \ Images \ Splash \ 16Color*.

Крок 6. У вікнах інспекторів об'єктів для *Image1* та *Image2* у наступних властивостях змінюємо значення:

- *Stretch* змінюємо режим *false* на *true*;
- *Transparent* змінюємо режим *false* на *true*;
- *AutoSize* змінюємо режим *false* на *true*;
- *Center* змінюємо режим *false* на *true*.



Рис. 2-4. Рисунки завантажені за допомогою кнопки Load.

Крок 7. Виконуємо команду *Run* в меню команд *C++ Builder* або натискаємо на клавіатурі клавішу *F9* для перегляду результату роботи програми. У результаті на екрані дисплея з'явиться вікно у вигляді рис. 2-4. У компоненту *Image* рисунок можна завантажити програмно за допомогою компоненти – діалог *OpenPictureDialog* тоді в цьому випадку можна в прикладній програмі використовувати наступний оператор.

```
.....  
if (OpenPictureDialog1->Execute( ) )  
Image1->Picture->LoadFromFile( OpenPictureDialog1->FileName );
```

.....

Даний оператор може завантажувати рисунки з файлу будь якого типу: бітова матриця, піктограма або метафайл. Якщо відомо, що будуть завантажуватися рисунки тільки з файлів бітових матриц, тоді оператор до завантаження можна записати у такому вигляді

.....

```
Image1->Picture->Bitmap->LoadFromFile( OpenPicTureDialog->FileName );
```

.....

Для файлів з піктограмками оператор для завантаження зображень може бути таким

.....

```
Image1->Picture->Icon->LoadFromFile( OpenPicTureDialog->FileName );
```

.....

а для метафайлів– може бути такій оператор

.....

```
Image1->Picture->Metafile->LoadFromFile( OpenPicTureDialog->FileName );
```

.....

При допомозі метода *SaveToFile* зображення рисунка з компоненти *Image* можна зберігати у файл з новим ім'ям, якщо записати у програму такій оператор

.....

```
if ( SavePictureDialog->Execute( ) )  
Image1->Picture->SaveToFile( SavePictureDialog->FileName );
```

.....

Якій обрати для МКР-2, з вище розглянутих методів обробки графічного зображення у компоненти *Image* на вікні *Form*, залежить від постановки завдання до розробки прикладної C++ програми.

Далі розглянемо приклад C++ програми якій пояснює, яким чином можна використовувати компоненти *Image*, *Button* та *Label* у вікні *Form*. Для цього прикладу з розробки C++ програми сформулюємо таку постановку завдання до програмування:

- Необхідно для роботи у *Windows* створити *C++* програму у якій на *Form1* повинно бути вікно з рисунком, кнопка з назвою "СТАРТ" та місце для напису текстового повідомлення;
- При роботі *C++* програми повинні відбуватися і оброблятися командами мишки наступні події:

перша подія – поява напису у полі для повідомлення з таким текстом "Дана програма створена у інтегрованому середовищі *C++ Builder* для роботи у *Windows*", якщо виконується мишкою натискання кнопки "СТАРТ";

друга подія – видалення *C++* програмою рисунка з поля *Image* у вікні *Form1*, якщо буде виконуватися переміщення курсору мишки по напису текстового повідомлення;

третья подія – відновлення рисунка у вікні *C++* програми, якщо мишка буде встановлена на полі *Form1* або переміщуватись по полю вікна *Form1*.

Для створення і програмування даної *C++* програми потрібно з бібліотеки *VLC* на вікно *Form1* встановити такі компоненти: *Image1*, *Button1* та *Label1* і запрограмувати виконання обробки подій до мишки.

Компоненту *Image1* потрібно обрати для вікна *Form1* діями з *VLC* відповідно до рис. 2-1 і встановити на форму.

Компоненту *Button1* потрібно обрати для вікна *Form1* діями з *VLC* відповідно до рис. 2-5 і встановити на форму.

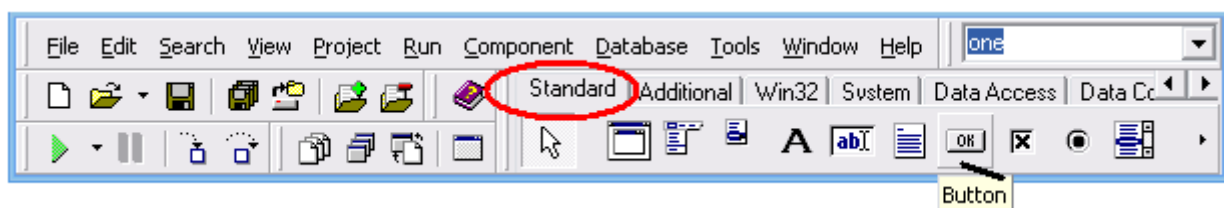


Рис. 2-5. Палітра бібліотеки *VLC* з закладкою *Additional* для обрання кнопки *Button*.

Компонента *Button* використовується у прикладних *C++* програмах для виконання користувачем потрібних команд за допомогою кнопки і у вікно *Form* встановлюється зі сторони *Standard* бібліотеки *VLC* (рис. 2-5). Компонента *Button* являє собою стандартну кнопку *Windows*, яка ініціалізує у програмі

подію (якась дія) відповідно до пояснюючого напису на кнопці. Для створення напису на кнопці використовується основна властивість - *Caption* на закладці *Properties* у вікні *Object Inspector* (рис. 2-6). Подія для кнопки виконується щикликом мишки на зображенні кнопки і необхідну подію обирати зі списку на закладці *Events* у вікні *Object Inspector* (рис. 2-7). Основна подія кнопки - *OnClick*, виникає і обробляється у програмі при натисканні мишкою по кнопці. При установці для кнопки властивості *Cancel* в значення *true*, означає, що дію для кнопки можна виконувати у програмі з клавіатури клавішею *Esc*.

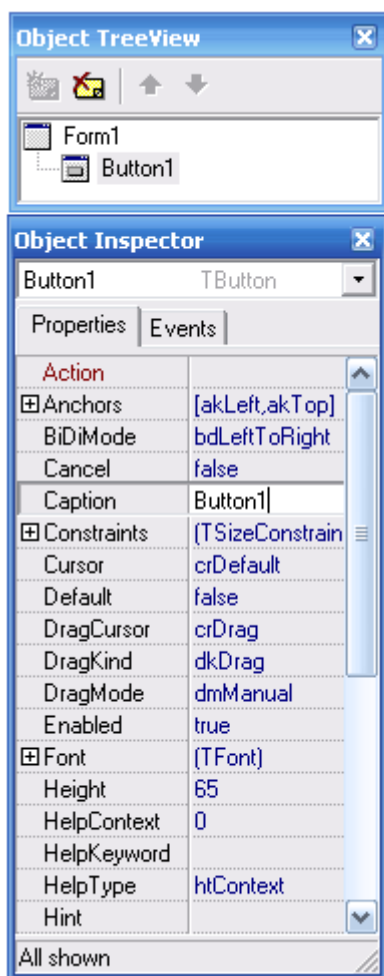


Рис. 2-6.

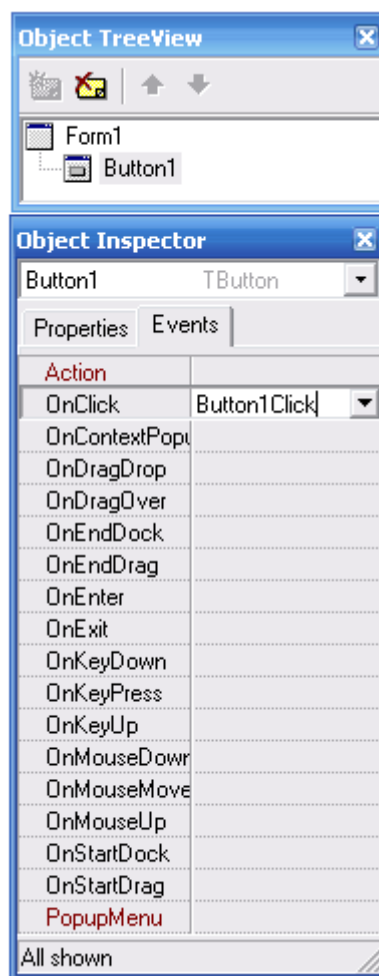


Рис. 2-7.

Також, якщо для кнопки властивість *Default* встановити в значення *true*, тоді дію для кнопки можна виконувати у програмі з клавіатури клавішею *Enter*. Основні властивості для кнопки показані в таблиці № 2-2.

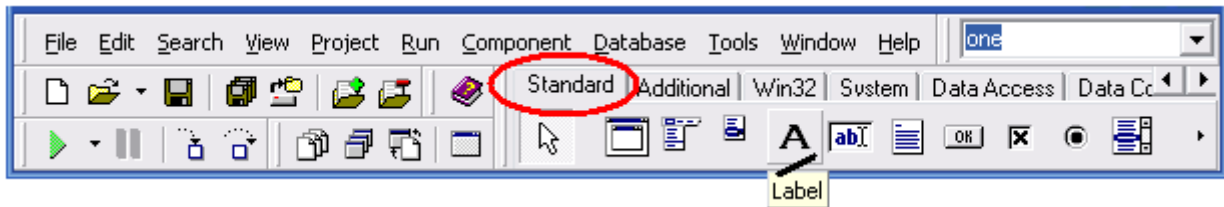


Рис. 2-8. Палітра бібліотеки *VLС* з закладкою *Additional* для обрання *Label1*.

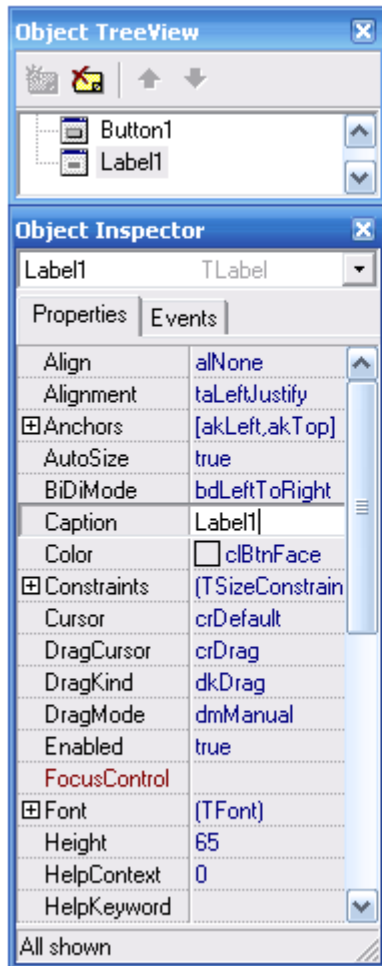


Рис. 2-9.

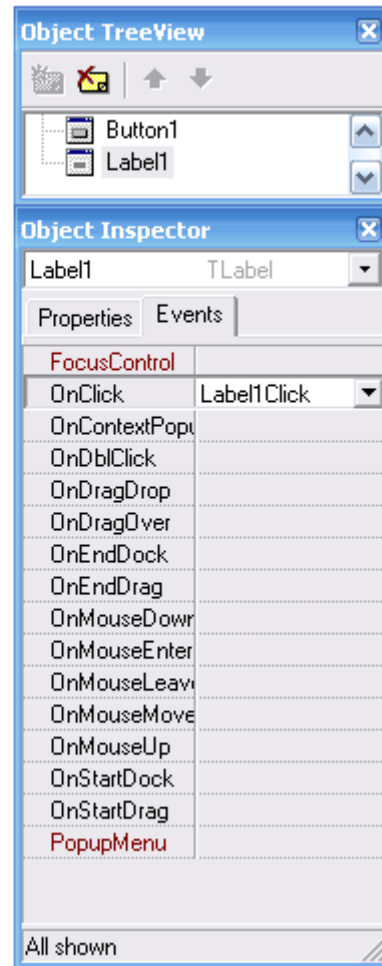


Рис. 2-10.

Таблиця № 2-2.

Значення властивості	Властивість яка буде встановлена для компоненти Button
Action	Дія (подія), пов'язане з даною кнопкою.
Cancel	Визначає, чи буде оброблятися для кнопки подія <i>OnClick</i> при натисканні клавіші <i>Esc</i> на клавіатурі.
Caption	Напис на кнопці.
Default	Значення визначає, що натискання користувачем на клавіатурі <i>Enter</i> буде еквівалентно натисканню на дану кнопку, навіть якщо дана кнопка у цей момент не перебуває в фокусі.
TabOrder	Вказується позиція компоненти у списку табуляції, який визначає порядок перемикання фокусу (маркірної рамки) між компонентами на <i>Form</i> при натисканні клавіші <i>Tab</i> . Спочатку порядок у списку відповідає послідовності додавання компонент на форму.

TabStop	Значення визначає можливість доступу користувача до кнопки за допомогою клавіші <i>Tab</i> .
---------	--

Компонента *Label* використовується у прикладних **C++** програмах для текстових написів або повідомлень на вікні *Form* і обирається вона зі старниці *Standart* бібліотеки *VLC* відповідно до рис. 2-8.

Таблиця № 2-3.

Значення властивості	Властивість яка буде встановлена для компоненти <i>Label</i>
AutoSize	Якщо це властивість встановлено в <i>true</i> , тоді вертикальний і горизонтальні розміри компоненти визначаються розміром напису. Якщо ж <i>AutoSize</i> має <i>false</i> , то в цьому випадку вирівнювання тексту всередині компоненти определяється властивістю <i>Alignment</i> .
Alignment	Властивість в межах поля <i>Label</i> управляє вирівнюванням тексту по горизонталі. Якщо властивість <i>AutoSize</i> задано в <i>false</i> і значеннями: <i>taLeftJustify</i> - вліво, <i>taRightJustify</i> - вправо, <i>taCenter</i> - по центру.
Caption	Рядок тексту, що відображається на полі <i>Label</i> .
Color	Значення задає колір фону для тексту напису.
Font	Визначаються атрибути шрифту для тексту напису.
Layout	Значення визначають вирівнювання тексту на полі <i>Label</i> по вертикалі.
ParentColor	Значення при <i>true</i> робить фон невидимим, а буде тільки видно напис з <i>Caption</i> .
ShowAccel Char	Значення визначає відображення символу <i>&</i> на полі <i>Label</i> .
Word Wrap	Значення вказує, чи переноситься текст на новий рядок, якщо він перевищує ширину поля <i>Label</i> , а висота поля дозволяє розмістити кілька рядків.

Компонента *Label* використовується, як відповідне поле (метка) на вікні *Form* для показу текстового напису. Текст до *Label* задається властивістю *Caption*, а шрифт та колір для напису встановлюються властивістю *Font* на закладці *Properties* у вікні *Object Inspector* (рис. 2-9). Колір фону для *Label* задається властивістю *Color*. Основні властивості до *Label* показані у таблиці № 2-3. Події які допустимі на полі для *Label* визначені у списку на закладці *Events* у вікні *Object Inspector* (рис. 2-10).

Таблиця № 2-4.

Позначення події	У програмі буде виконуватися обробка на подію для мишки
OnClick	Одне клацання миші на компоненте.
OnDbClick	Подвійне клацання миші на компоненті.
OnMouseDown	Натискання кнопки миші над компонентою. Можливо розпізнавання натиснутої кнопки і координат курсора миші.

OnMouseMove	Переміщення курсора миші над компонентою. Можливо розпізнавання натиснутої кнопки і координат курсора миші.
OnMouseUp	Відпускання раніше утримуваної кнопки миші над компонентою. Можливо розпізнавання натиснутої кнопки і координат курсора миші.
OnMouseDown	Починається перетягування (переміщення) об'єкту. Можливо розпізнавання перетягнутого об'єкту.
OnDragOver	Переміщення перетягнутого об'єкта над компонентою. Можливо розпізнавання перетягнутого об'єкта і координат курсора миші.
OnDragDrop	Відпускання раніше утримуваної кнопки миші після перетягування об'єкта. Можливо розпізнавання перетягнутого об'єкта і координат курсора миші.
OnEndDrag	Ще одна подія при відпусканні раніше утримуваної кнопки миші. Можливо розпізнавання перетягнутого об'єкта і координат курсора миші.
OnEnter	Подія у момент отримання елементом форми фокусу в результаті дій миші, при натисканні клавіші табуляції або програмної передачі фокусу.
OnExit	Подія у момент втрати елементом форми фокусу в результаті дій миші, натиснути клавішу табуляції або програмної передачі фокусу.

При обранні мишкою на закладці *Events* відповідної назви події для обробки до активної компоненти, яка виделена маркерною рамкою на *Form* автоматично інтегроване програмувальне середовище *C++ Builder* у текст файлу (.cpp) модуля форми *C++* програми додає шаблон для програмування обробки подій і підключає потрібні файли з розширенням (.h). Розглянемо далі, які події до мишки на закладках *Events* у вікні *Object Inspector* можна обірати для компоненти, встановленої у вікно *Form* та позначеної активною маркерною рамкою. Список таких подій показано у таблиці № 2-4.

2.2 ПРИКЛАД ПРОГРАМУВАННЯ ОБРОБКИ ПОДІЙ ДО КОМПОНЕНТ У ВІКНІ FORM

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 компоненти *Form*;
- Перейдіть у вікно інспектора об'єктів і у властивості *Caption* для заголовка вікна *Form1* задайте таку назву " Програма створена у *C++ Builder* ".

➤ Збережіть новий проект файлів до зміни заголовку у вікні *Form1*. В результаті *C++ Builder* запам'ятає шлях для швидкого збереження змін у проекті файлів нової *C++* програми;

➤ Для збереження файлів проекту необхідно на *D:* створити папку з назвою *Prog_1*;

➤ В *C++ Builder* у меню *File* виберіть команду *Save Project As* і з появою запиту на збереження, змініть назву файлу *Unit1* на файл *U_Prog1.cpp*, а назву проекту *Project1.bpr* замініть на *P_Prog1.bpr*;

Крок 2. У вікно форми *Form1* встановлюємо компоненту *Label1*:

➤ На закладці *Standart* палітри компонентів *VCL* курсором і одним щигликом миші виділяємо компоненту *Label*;

➤ На формі вікна з заголовком "Програма створена у *C++ Builder*" одним щигликом миші встановлюємо компоненту *Label1* і задаємо маркерною рамкою по горизонталі і вертикалі необхідний розмір компоненти *Label1* до відповідного текстового повідомлення (рис. 2-12). Маркерну рамку розтягніть методом зміщення чорного квадратика під розмір майбутнього напису тексту.

Крок 3. Налаштуємо у вікні інспектора об'єктів властивості до тексту, що буде виводитися на компоненті *Label1* (шрифт, розмір і колір напису):

➤ На закладці *Properties* подвійним щигликом мишки оберіть властивість *Font/(TFont)* для відкриття вікна "Шрифт" де обираємо для напису необхідний шрифт, розмір та колір (рис. 2-11).

Крок 4. На формі вікна у полі для текстового повідомлення видаляємо напис *Label1*, щоб він не накладався на текст повідомлення:

➤ На компоненту *Label1* встановить мишкою маркерну рамку;

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

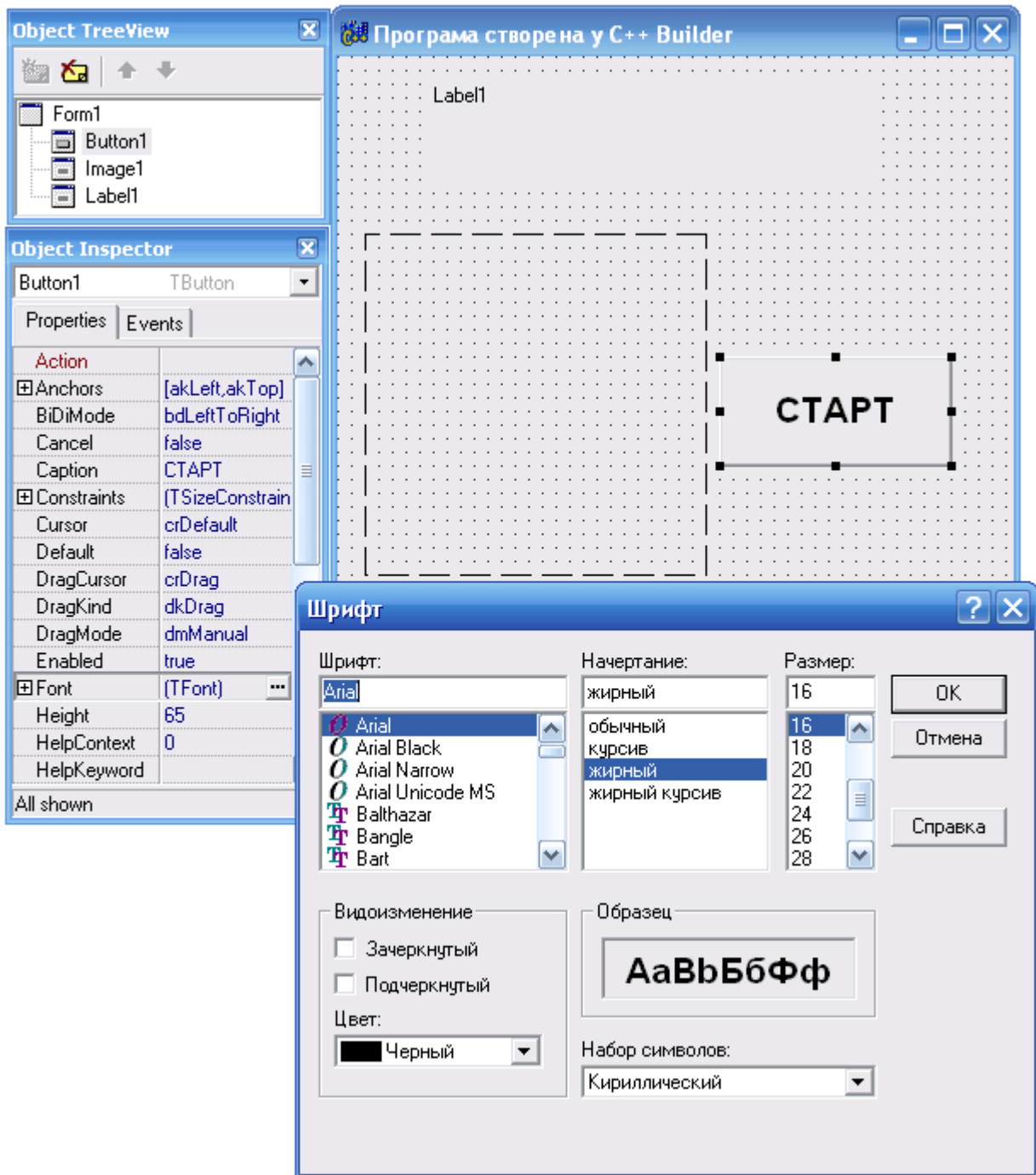


Рис. 2-11.

Крок 5. У вікно форми *Form1* встановлюємо компоненту кнопки *Button1*:

- На закладці *Standart* палітри компонентів *VCL* виберіть одним щигликом мишки компоненту "кнопка" з ярличком підказки *Button*;
- На полі форми із сіткою зробіть щиглика мишкою, щоб встановилася кнопка з назвою *Button1* з охопленою маркерною рамкою.

Крок 6. На кнопці *Button1* змінюємо назву на "СТАРТ":

- Клацніть по кнопці *Button1* на формі вікна для появи маркерної рамки;
- Перейдіть у вікно інспектора об'єктів (*Object Inspector*) і на сторінці властивостей об'єкта (*Properties*) встановить курсор у виділене поле *Caption* на напис *Button1* і замінюємо на назву "СТАРТ" (рис. 2-12).

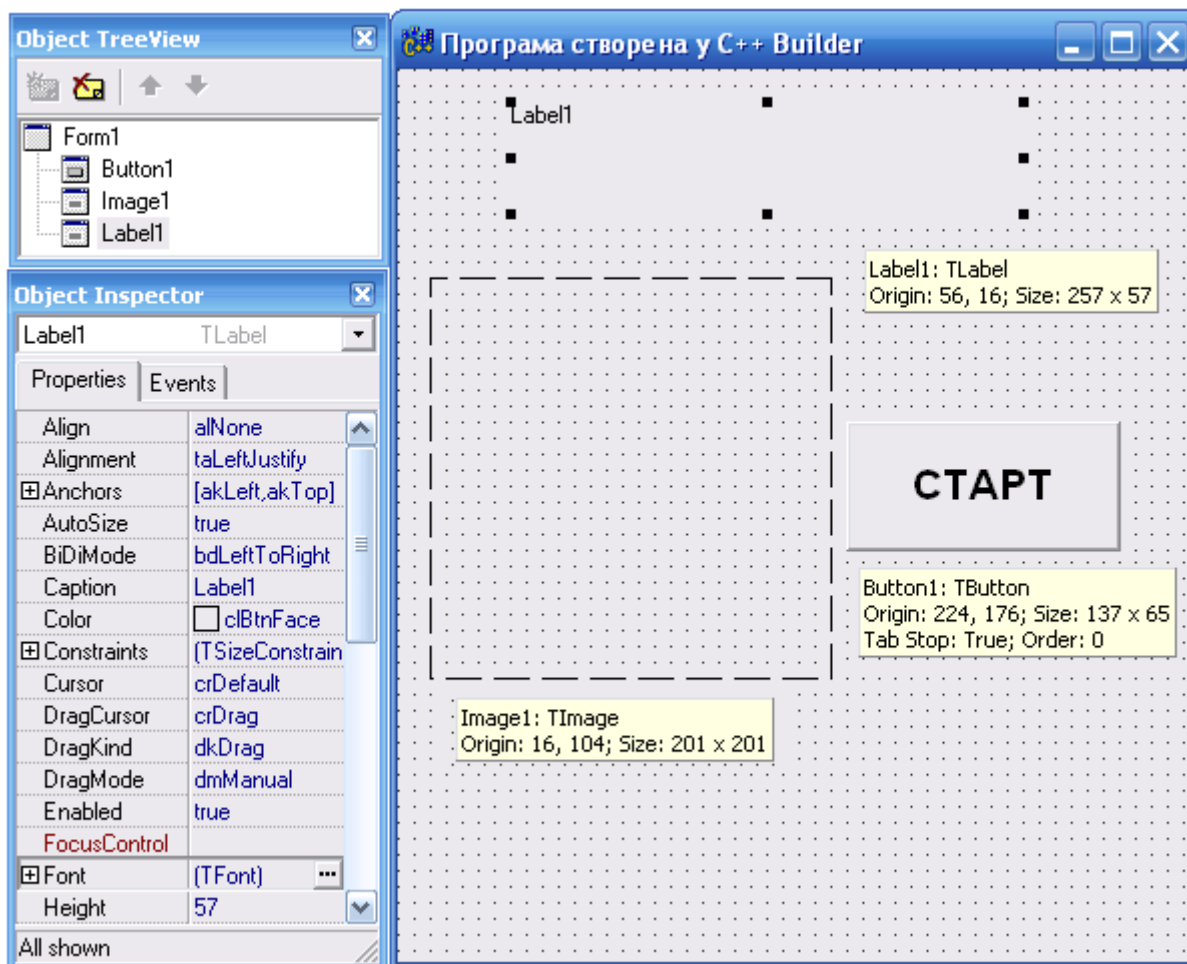


Рис. 2-12.

Крок 7. Для кнопки "СТАРТ" визначаємо подію для мишки до вивіду тексту «Дана програма створена у інтегрованому середовищі *C++ Builder* для роботи у *Windows*». Для цього є два варіанти з виконання команд:

Варіант 1.

- Активізуйте мишкою маркерну рамку на кнопці "СТАРТ";
- Перейдіть у вікно *Object Inspector* на закладку подій *Events* і виконайте подвійного щиклика мишкою на білому полі біля напису *OnClick*, щоби відбулося наступне:

- на білому полі з'явився напис події *Button1 Click*;
- у вікні редактора кодів до тексту програми було додано шаблон функції з обробки події :: *Button1 Click*.

Варіант 2.

- На формі по кнопці "СТАРТ" зробіть подвійного щиглика мишкою, щоб автоматично активізувалися:
 - інспектор об'єктів (*Object Inspector*), де на закладці *Events* вже буде вказана подія *Button1 Click*;
 - одночасно з подвійним щигликом мишки у вікно редактора кодів до тексту програми було додано шаблон функції з обробки події :: *Button1 Click*.

Крок 8. У вікні редактора кодів для файлу *U_Prog1.cpp* записуємо оператори команд між фігурними дужками функції до події :: *Button1 Click*:

- У редакторі кодів запишіть оператор для події – показ тексту повідомлення "Дана програма створена у інтегрованому середовищі *C++ Builder* для роботи у *Windows*". Оператор записується у визначення функції *TForm1::Button1 Click*, яка повинна мати такий вигляд:

```
//-----
void __fastcall TForm1::Button1Click(TObject *Sender)
{
  Label1->Caption = "Дана програма створена у інтегрованому середовищі
  C++ Builder для роботи у Windows ";
}
//-----
```

Крок 9. Командами *Run/Run* чи на клавіатурі клавішею *F9* виконуємо компіляцію вихідних файлів і запускаємо на виконання програму, щоб перевірити правильність виконання операторів з обробки першої події:

- Якщо не було помилок при запису оператора *Label1->Caption =* , тоді на екрані дисплея комп'ютера знімається вікно інспектора об'єктів і

з'являється вікно створюваної C++ програми з кнопкою "СТАРТ". Мишкою клацніть кнопку "СТАРТ", щоб у вікні на полі компоненти *Label1* з'явився напис "Дана програма створена у інтегрованому середовищі C++ *Builder* для роботи у *Windows*".

Крок 10. Налаштовуємо показ напису "Дана програма створена у інтегрованому середовищі C++ *Builder* для роботи у *Windows* " жирним шрифтом і червоним кольором у полі *Label1*:

- У вікні інспектора об'єктів вгорі у списку компонент до форми оберіть назву *Label1*, щоб для компоненти *Label1* 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 11. Перевіряємо правильність внесених змін у текст програми:

- Виконаєте компіляцію вихідних файлів і запуснете на виконання C++ програму командами *Run/Run* чи на клавіатурі натисніть клавішу *F9*.

Крок 12. 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, щоби відкрилося стандартне вікно "Шрифт" для зміни виду і розміру шрифту.

Крок 13. Командами *Run/Run* чи на клавіатурі клавішею *F9* виконуємо компіляцію вихідних файлів і для перевірки запускаємо на виконання C++ програму.

Крок 14. На закладці *Additional* панелі бібліотеки *VLC* збираємо компоненту *Image* і встановлюємо її в нижній частині вікна форми *Form1*.

Крок 15. Для *TForm1* задаємо функцію з відкриття вікна C++ програми:

- Виконайте щиглика мишкою на сітці форми *Form1* і перейдіть у вікно інспектора об'єктів на закладку подій *Events*;
- Виберіть подію *OnCreate* і зробіть подвійного щиглика мишкою, щоби у вікні редактора кодів був доданий наступний шаблон функції:

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

Крок 16. У вікні редактора кодів для файлу *U_Prog1.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 рисунка у поле Image1 при відкритті вікна
  //-----рисунок Prog1_On.bmp необхідно помістити у папку Prog_1
  Pict->Picture->LoadFromFile("Prog1_On.bmp");
  Image1->Canvas->CopyRect(Image1->ClientRect, Pict->Canvas,
    Rect(0,0,Pict->Width,Pict->Height));
  On_Off = 1; //----- признак стану – рисунок показано
}
//-----
```

Крок 17. Командами *Run/Run* або на клавіатурі клавішею *F9* виконуємо компіляцію вихідних файлів і запускаємо на виконання **C++** програму для перевірки роботи доданих операторів у функції.

Крок 18. Для компоненти *Label1* задаємо обробку події – видалення рисунка з поля вікна форми, якщо курсор мишки буде розташовано на текст напису "Дана програма створена у інтегрованому середовищі **C++ Builder** для роботи у *Windows*":

- Маркерною рамкою на формі виділяємо компоненту *Label1*;

- Перейдіть у вікно інспектора об'єктів на закладку подій *Events*;
- Оберіть подію *OnMouseMove* і зробіть подвійного щиглика мишкою, щоб

у вікні редактора кодів було додано шаблон для функції з обробки події:

```
//-----
void __fastcall TForm1::Label1MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
}
//-----
```

Крок 19. У вікні редактора кодів для файлу *U_Prog1.cpp* записуємо оператори у шаблон функції для обробки події - видалення рисунка з поля вікна програми при переміщенні курсора мишки над текстовим повідомленням:

```
//-----
void __fastcall TForm1::Label1MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
    TImage *Pict = new TImage(Form1);
    Pict->AutoSize = true;
    //----- видалення рисунка з поля Image1
    //рисунок Prog1_Off.bmp має зображення фону поля і збережено у папці Prog_1
    Pict->Picture->LoadFromFile("Prog1_Off.bmp");
    Image1->Canvas->CopyRect(Image1->ClientRect, Pict->Canvas,
        Rect(0,0,Pict->Width,Pict->Height));
    On_Off = 0; //---признак стану – рисунок видалено
}
//-----
```

Крок 20. Командами *Run/Run* чи на клавіатурі клавішею *F9* виконуємо компіляцію вихідних файлів і запускаємо на виконання програму.

Крок 21. Якщо з'явилося повідомлення, що не оголошена змінна *On_Off*. Додаємо для змінної таке оголошення *int On_Off = 0* на зовнішньому рівні програми (дивись крок № 24).

Крок 22. Для форми *Form1* запрограмуємо подію – відновлення рисунка на полі вікна форми, якщо курсор мишки буде розташовано на полі вікна програми:

- На сітці форми *Form1* зробіть щиглика мишкою;
- Переходимо у вікно інспектора об'єктів на закладку подій *Events* ;

- Оберіть 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 23. У вікні редактора кодів для файлу *U_Prog1.cpp* заповнюємо оператори у шаблон функції для обробки події – переміщення курсору мишки по полю вікна форми програми для відновлення рисунка на полі компоненти *Image*:

```
//-----
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 Image1
{
Pict->Picture->LoadFromFile("Prog1_On.bmp");
Image1->Canvas->CopyRect(Image1->ClientRect, Pict->Canvas,
Rect(0,0,Pict->Width,Pict->Height));
On_Off = 1; //----рисунок показано
}
}
//-----
```

Крок 24. У файлі *U_Prog1.cpp* перевіряємо такі вказівки препроцесору і оголошення:

```
//-----
#include <vcl.h>
#pragma hdrstop
#include "U_Prog1.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
int On_Off = 0; //--- признак для контролю стану рисунка (показано/видалено)
//-----
__fastcall TForm1::TForm1(TComponent* Owner)
: TForm(Owner)
```

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

Крок 25. Перевіряємо роботу C++ програми шляхом виконання компіляції вихідних файлів і для цього виконуємо команди *Run/Run* або на клавіатурі натискається клавіша *F9*.

3. ПРАВИЛА ВИКОРИСТАННЯ КОМПОНЕНТИ MAIN MENU ДЛЯ ПРОГРАМУВАННЯ МЕНЮ КОМАНД ПРИКЛАДНОЇ C++ ПРОГРАМИ

Практично будь-яка C++ програма завжди має основне меню команд та підменю команд, оскільки саме меню команд дає найбільш зручний інтерфейс для роботи мишкою з прикладною програмою. Також завжди у меню команд можна бачити перелік можливих до виконання дій користувача у прикладній програмі. Інтерфейси до команд у більшості C++ програм для *Windows* реалізуються такими видами меню:

- головне меню команд зі списком команд, що випадає;
- каскадні меню, у яких розділу первинного списку команд з меню ставиться у відповідність список підрозділів набору команд нижнього залежного рівня;
- спливаючі або контекстні меню, які з'являються, якщо користувач щелкне правою кнопкою мишки на якомусь елементі у вікні прикладної C++ програми.

Основна вимога до меню команд у C++ програмах це їхня стандартизація. Основна мета стандартизації меню команд у прикладних програмах це полегшити користувачу роботу з прикладною C++ програмою. Завжди треба, щоб меню команд прикладної програми чітко визначало для користувача його дії і не було потрібно додатково уточнювати, де шукати команди. Також стандартизація розташування меню команд на формі у вікні *Windows* виробляє у користувачів упевнену роботу з прикладними C++ програмами.

Для прикладної C++ програми до МКР №2 необхідно розробити основне меню команд з залежними підменю команд, які повинні мати структуру команд відповідно до зображення на рис. В-1.

У C++ *Builder* для створення і програмування основного меню команд і підменю команд для C++ програми маються такі компоненти: *ActionList* (рис. 3-1), *MainMenu* (рис. 3-2), *ImageList* (рис. 3-3) та інші. Зручно розробляти меню команд до прикладної C++ програми за допомогою диспетчеризації дій і подій на оснві компоненти *ActionList*. Компонента *ActionList* не додає ніяких додаткових можливостей, лише дозволяє розробнику прикладної програми систематизувати і упорядкувати розробку об'єктно-орієнтованих C++ програм шляхом організації зв'язку між діями (подіями) і їх ініціаторами, такими як клацання на кнопці або на елементе меню команд. Компонента *ActionList* є інтерфейсом розробника, що дозволяє йому впорядкувати свою роботу з діями (подіями) в процесі проектування шляхом створення списків дій. Компонента *ActionList* встановлюється у вікно *Form* зі сторони *Standard* бібліотеки *VLC*.

На початку проектування прикладної програми розробник повинен уявити собі список тих дій, які може виконувати користувач. Практична реалізація складеного списку дій починається з перенесення на проектувану форму компоненти *ActionList*. Зробивши на цій компоненті подвійне клацання мишкою створює у вікні форми відкриття вікна редактора дій, якій дозволяє вводити і впорядковувати список допустимих дій (подій) у програмі. Клацання правою кнопкою миші або клацання на маленькій кнопці зі стрілкою вниз правіше першої швидкої кнопки вікна редактора дозволяє вибрати оду із таких команд: *New Action* (нова дія) або *New Standard Action* (нова стандартна дія). Команда *New Action* допускає введення нової дії будь-якого типу. За замовчуванням ці дії будуть отримувати імена *Action1*, *Action2* і так далі. Команда *New Standard Action* відкриває вікно, в якому розробник може вибрати необхідні стандартні дії. В результаті у правому віконці *Actions* редактора з'являться імена обраних дій, а в лівому *Categories* список категорій дій. У вікні Інспектора Об'єктів для кожної дії розробник може встановити властивість *Name* - ім'я, а також ряд

властивостей, які автоматично перенесуться потім в усі компоненти, що будуть посилатися на дану дію.

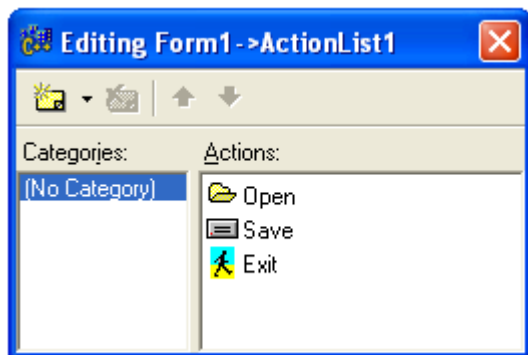


Рис. 3-1.

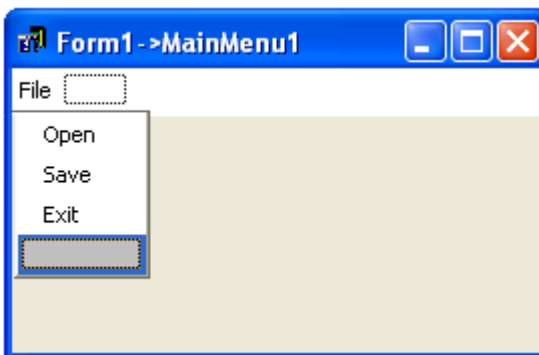


Рис. 3-2.

Застосування *ActionList* ще також дозволяє заощаджувати час на проектування меню команд до прикладної програми. Ця компонента має список дій (рис. 3-1), які передбачаються бути командами у меню команд створюваної програми. Дія - це деяка реакція C++ програми на вплив дії користувача та подій, запрограмованих для обробки інформації, наприклад, щиглик на кнопки або на напису назві команди в меню команд. При проектуванні прикладної програми необхідно скласти список тих дій, які прийдеться виконувати користувачу при роботі з прикладною програмою. В даному навчальному посібнику список обробки команд на події до мишки створюваних користувачем, визначено на рис. В-1 та індивідуальними завданнями до виконання модульної контрольної роботи з кредитного модуля «Візуальне програмування прикладних програм» навчальної дисципліни «Технології розробки прикладних програм».

Первісне складання списку дій (команд) майбутньої C++ програми виконується через встановлення на компоненту *Form1* невізуальної компоненти *ActionList*. При щелчку по значку *ActionList* у вікні форми *Form1* відкривається редактор дій, який дозволяє додавати дії і їх упорядковувати у списку.

Звичайно у більшості прикладних програм перед назвою команди у меню команд показуються відповідні значки, які допомагають користувачу краще розрізняти і сприймати відповідні команди. Для формування набору значків необхідно встановити на форму *Form1* компоненту *ImageList*, яка обирається на панелі бібліотеки компонент *VCL* на сторінці з закладкою *Win32*. Ця

компонента дозволяє організувати ефективне та ошадливе керування безліччю піктограмок майбутніх команд **C++** програми для роботи у *Windows*.

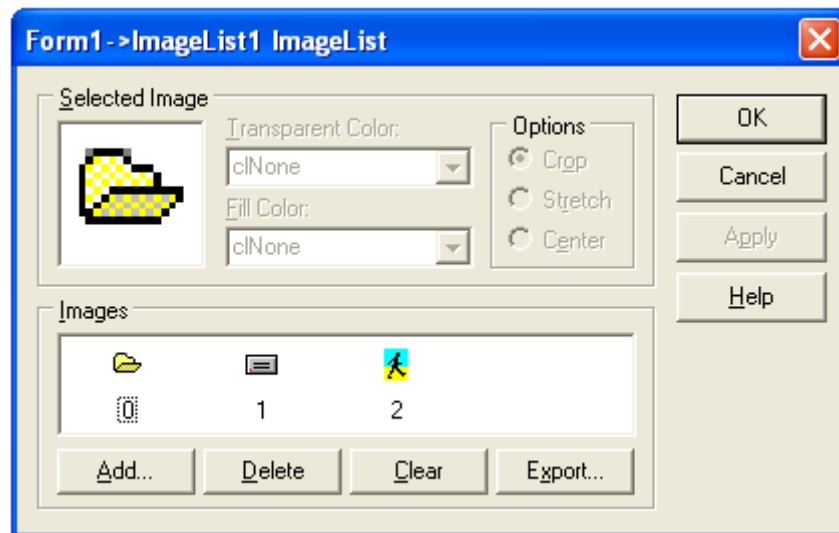


Рис. 3-3.

Після установки на форму *Form1* компоненти *ImageList* потрібно подвійним щелчком мишки на значку компоненти *ImageList* активізувати вікно компоновщика значків (рис. 3-3).

Відповідно до списку команд вказаному в *ActionList1* на рис. 3-1 необхідно файли зі значками обирати у *ImageList* за допомогою кнопки *Add* і можна відразу вставляти з файлу кілька відповідних значків до команд меню, а зайві обрані значки видаляються кнопкою *Delete*. При додаванні піктограмок у список значків автоматично задається цифровий індекс з прив'язки до назви команди меню (дії прикладної програми).

Формування меню команд та налаштування (програмування) його властивостей до дій користувача при роботі з програмою виконується за допомогою компоненти *MainMenu* (рис. 3-2), обираємої у бібліотеці компонент *VCL* на сторінці з закладкою *Standard*. Компонента *MainMenu* має особливість – вона невізуальна і після її встановлення на форму *Form1* до вікна розробляємої програми потрібно подвійним щелчком мишки на значку *MainMenu* активізувати вікно конструктора меню команд, якій показано на рис. 3-2. Поле

для заповнення найменування команди до меню команд показується автоматично курсорною пунктирною рамкою.

3.1 ПРИКЛАД РОЗРОБКИ ТА ПРОГРАМУВАННЯ ОБРОБКИ ПОДІЙ ДО МЕНЮ КОМАНД НА ОСНОВІ КОМПОНЕНТИ MAIN MENU

Розглянемо далі приклад до правил використання компоненти *MainMenu* для побудування у **C++** програмі меню команд *File* з підкомандами:

- *Open* (можна буде переглядати файли з рисунками);
- *Save As...* (можна буде зберегти рисунок у файл із заданим ім'ям);
- *Exit* (можна вийти і закрити програму).

Крок 1. Відкриваємо нову форму *Form* для **C++** програми і у властивості *Caption* напишемо таку назву "Програмування меню команд", щоб цей напис з'явився у заголовку вікна форми. Далі створюємо папку *Prog_MainMenu* на диску *D:* і виконуємо команду *File/Save Project As...* для збереження таких файлів проекту *P_ MainMenu.bpr* та файлу *U_ MainMenu.cpp*.

Крок 2. У бібліотеці компонентів *VCL* на сторінці *Standard* обираємо компоненту *ActionList* і встановлюємо на *Form*.

Крок 3. У бібліотеці компонентів *VCL* на сторінці *Win32* обираємо компоненту *ImageList* і встановлюємо на *Form* для завдання значків до меню команд **C++** програми.

Крок 4. На значку *ImageList1* у полі *Form* виконуємо подвійного щелчка мишкою. Далі у вікні, що відкрилося, через кнопку *Add...* обираємо з папок *Program_Files\Common_Files\Borland_Shared\Images\Buttons* значки для меню команд *File* з підкомандами: *Open*, *Save*, *Exit*. Для цих підкоманд обираємо для значків такі файли: *fldropen.bmp*, *harddisk.bmp*, *picture.bmp*.

Крок 5. У бібліотеці компонентів *VCL* на сторінці *Dialogs* обираємо і встановлюємо на форму вікна **C++** програми такі невізуальні компоненти *OpenPictureDialog* та *SavePictureDialog*.

Крок 6. Далі необхідно зв'язати диспетчер дій зі списком обраних значків для команд і тому виконуємо наступне:

- Встановлюємо маркерну рамку на значку *ActionList* ;
- У вікні Інспектора Об'єктів у полі властивості *Image* вкажіть *ImageList1*.

Крок 7. На значку *ActionList* у полі форми подвійним щелчком мишки відкриємо редактор дій для меню команд і виконаємо наступне:

- Клавішею *Insert* у полі *Action* додаємо елемент *Action1* і у властивостях на полі *Name* змінюємо назву на *Open*;
- У властивостях *Caption* та у полі *Hint* записуємо назви *Open*. Потім у властивості *Imageindex* зі списку вказуємо до команди *Open* номер значка, який визначено у компоненті *ImageList1*;
- Далі кнопкою *New Action(Ins)* додаємо у список назву *Action1* і у властивостях на полі *Name* змінюємо назву на *Save*;
- У властивостях *Caption* та у полі *Hint* записуємо назву *Save*. Далі у властивості *Imageindex* зі списку вкажіть номер значка для команди *Save*;
- Через кнопку *New Action(Ins)* додаємо у список назву *Action1* і у властивостях на полі *Name* змінюємо назву на *Exit* та далі у властивостях *Caption* та *Hint* укажіть назву *Exit*. У властивості *Imageindex* зі списку вкажіть номер значка для команди *Exit*;
- Перевірте свої результати в *ActionList*.

Крок 8. У бібліотеці компонентів *VCL* на сторінці *Standard* оберіть компоненту *MainMenu* – конструктор меню команд і встановить на форму *C++* програми.

Крок 9. Задайте для властивості *Image* значення *ImageList1*.

Крок 10. На формі програми по значку *MainMenu* виконуємо подвійний щелчок мишкою. Відкриється вікно з курсорною пунктірною рамкою і виконуємо наступне:

- У властивості *Caption* наберіть до меню команд назву *File* і натисніть клавішу *Enter*.
- У властивості *Name* з'явиться назва *File1*, а в курсорній рамці у вікні компоненти *MainMenu* з'явиться напис *File* та будуть додані дві курсорні рамки для заповнень наступних назв команд.

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

Крок 12. У створюваному меню команд нижчий курсор виділіть і потім у властивості *Caption* наберіть *Save* і натисніть клавішу *Enter*. Перевірте записи у властивостях: *Imageindex*, *Name* та *Action*.

Крок 13. У проектуємому меню команд нижчий курсор виділіть і у властивості *Caption* наберіть *Exit* і натисніть клавішу *Enter*. Перевірте записи у властивостях: *Imageindex*, *Name* та *Action*. Вікно редактора компоненти *MainMenu* закрийте і на формі програми з'явиться напис *File*, на який встановить курсор мишки, щоб з'явився список команд зі значками.

Крок 14. Перейдіть у вікно редактора коду і перегляньте, що *C++ Builder* вже додав у модуль файлу *U_MainMenu.cpp*.

Крок 15. На формі *C++* програми виконайте по значку *ActionList1* подвійного щелчка мишкою для активізації на екрані списку дій. Далі потрібно визначити до події шаблони функцій для записаних у меню команд:

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

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

- Виконайте команду *Run* і перегляньте стан меню команд програми;
- На назві *Save* зробіть подвійного щелчка мишкою і тоді на закладці *Events* у полі *OnExecute* автоматично заповнюється подія *SaveExecute* та в модулі форми буде доданий шаблон коду для заповнення потрібних операторів програми;
- У шаблоні коду *SaveExecute* необхідно додати наступні оператори:

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

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

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

- Виконаєте команду *Run* і переглянете стан меню команд програми.

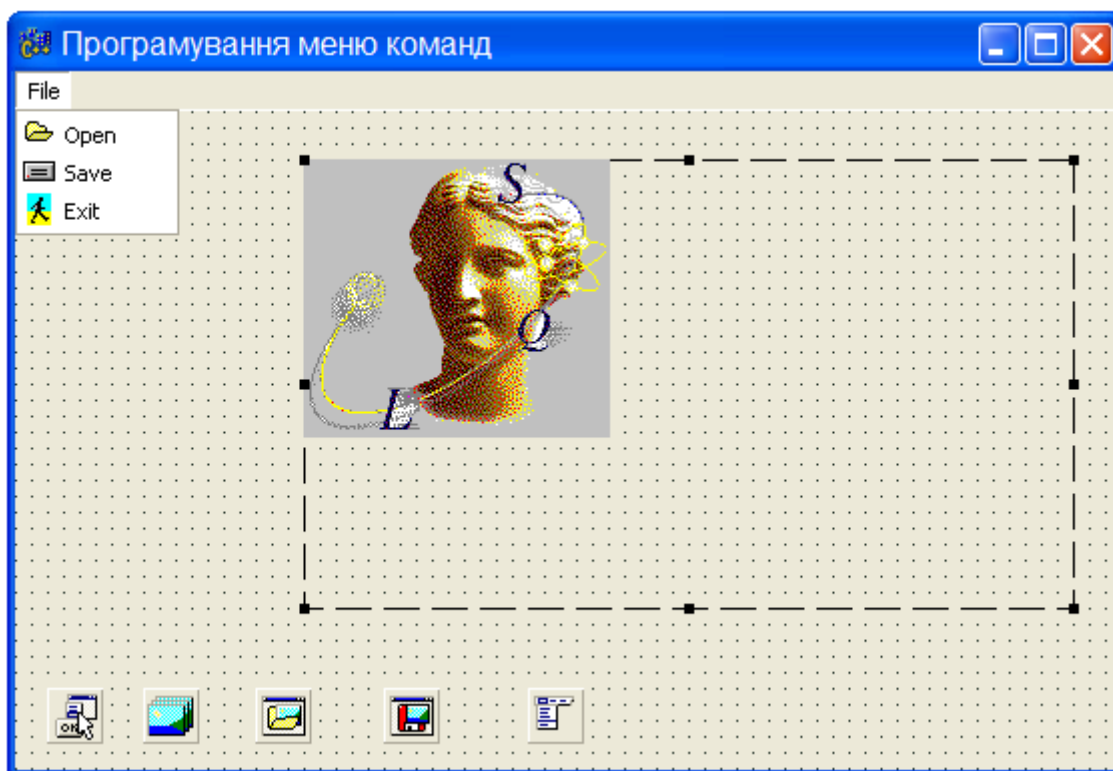


Рис. 3-4.

Крок 16. У бібліотеці *VCL* оберіть сторінку *Additional* і розмістіть на форму компоненту *Image1*, потім маркерною рамкою визначить розмір для рисунка.

Крок 17. Виконайте для форми (рис. 3-4) команду *Run* і перевірте роботу меню команд, сформованого за допомогою компоненти *MainMenu*. Файли рисунів обирайте з папок по такому шляху *Program_Files\Common_Files\ Borland_Shared \Images\Splash\16Color \....* .

4. НАВЧАЛЬНИЙ ПРИКЛАД ДО РОЗРОБКИ І ПРОГРАМУВАННЯ ПРИКЛАДНОЇ C++ ПРОГРАМИ ДЛЯ МОДУЛЬНОЇ КОНТРОЛЬНОЇ РОБОТИ КРЕДИТНОГО МОДУЛЯ

Навчання студентів з спеціальності 151 “Автоматизація та комп’ютерно-інтегровані технології” в навчальній дисципліні “Технології розробки програмного забезпечення-2” передбачено отримання практичного досвіду та знання з візуального програмування прикладних C++ програм, що можливо реалізувати за рахунок виконання модульної контрольної роботи за допомогою інтегрованого програмувального середовища *C++ Builder*.

Кредитний модуль “Візуальне програмування прикладних програм” відноситься до циклу «Навчальні дисципліни базової підготовки» і передбачає продовження вивчення сучасних технологій і мов програмування прикладних інженерно-технічних задач. Навчальний матеріал кредитного модуля «Візуальне програмування прикладних програм» відрізняється від інших частин даної навчальної дисципліни тим, що вивчається сучасна технологія і техніка застосування мови **C++** для об’єктно-орієнтованого та візуального програмування прикладних програм, а також студенти отримують практичний досвід з техніки розробки і виготовлення прикладної **C++** програми у інтегрованому програмувальному середовищі **C++ Builder**. Модульна контрольна робота з програмування до кредитного модуля виконується у вигляді розробки і виконання матеріалів до індивідуального завдання та з розробки і програмування прикладної **C++** програми, у якій відображаються:

- мнемосхема технологічних процесів з хімічного виробництва;
- описи технологічних процесів з мнемосхеми хімічного виробництва;
- схеми до конструкцій основних апаратів з мнемосхеми;
- описи процесів у основних апаратах з мнемосхеми;
- показуються у часі графіки зміни параметрів технологічного процесу у основних апаратах мнемосхеми.

Навчальний приклад з програмування **C++** програми до модульної контрольної роботи, якій розглядається у даному початковому посібнику, показує методичні вказівки і алгоритми роботи для студентів по виконанню індивідуальних завдань до модульної контрольної роботи № 2.

Також навчальний приклад у посібнику орієнтовано на отримання студентами наступних знань:

- ❖ правил з розробки алгоритму до поставленої задачі програмування та з побудови за допомогою умовних графічних позначень блок-схеми алгоритму програмуемого завдання;

- ❖ техніки програмування мовою **C++** шляхом написання і налагодження прикладної програми у середовищі **C++ Builder** згідно розробленого алгоритму до індивідуального завдання;
- ❖ правил оформлення записки до модульної контрольної роботи, листингов програми, блок-схем алгоритмів та рисунків згідно до відповідних стандартів з оформлення науково-технічної інформації.

Даний навчальний приклад до виконання модульної контрольної роботи № 2 з програмування прикладної програми в інтегрованому середовищі **C++ Builder** показує студентам методичні вказівки до самостійної роботи з розробки матеріалів до індивідуального завдання з програмування та знання по використанню інспектора об'єктів для налаштування вікон *Form* та параметрів компонент і функцій для обробки подій до команд мишки, виконуваних користувачем при роботі з прикладною **C++** програмою.

4.1 ПОСТАНОВКА ЗАВДАННЯ ДО НАВЧАЛЬНОГО ПРИКЛАДУ З ВИКОНАННЯ МОДУЛЬНОЇ КОНТРОЛЬНОЇ РОБОТИ

У інтегрованому середовищі **C++ Builder** необхідно розробити прикладну **C++** програму, яка буде виводити в окремих вікнах на екрані дисплею інформацію у вигляді мнемосхеми технологічного процесу та опису процесів хімічного виробництва. Також **C++** програма у вікні *Form* повинна показувати інформацію про властивості та призначення продукції хімічного виробництва. У меню команд **C++** програми повинна бути можливість перегляду такої інформації: призначення технологічних апаратів на мнемосхемі технологічних процесів з хімічного виробництва; опис і графіки зміни технологічних параметрів для завданих основних апаратів з мнемосхеми хімічного виробництва. Також обов'язково потрібно розмістити інформацію про використану літературу з виконання модульної контрольної роботи та меню “Про програму”. Структура і пункти меню команд прикладної **C++** програми мають бути побудовані за допомогою компоненти *MainMenu* інтегрованого програмувального середовища **C++ Builder**.

Завдання до модульної контрольної роботи студентам необхідно виконувати у три таких етапи:

- перший етап це підготовка графічних та текстових матеріалів до завданої схеми технологічного процесу хімічного виробництва;
- другий етап це виконання програмування прикладної C++ програми у *C++ Builder* відповідно до сформульованих завдань;
- третій етап це оформлення звіту та захист результатів з виконання завдань до модульної контрольної роботи.

Навчальний приклад і методика виконання завдань до модульної контрольної роботи буде далі у навчальному посібнику розглянуто у вигляді необхідних дій для студентів з підготовки схем, текстів з описами процесів та програмування C++ програми до мнемосхеми технологічного процесу з виробництва хлорметанів.

Перший етап до виконання модульної контрольної роботи передбачає роботу студента над виконанням таких завдань:

- розробку на комп'ютері у графічному редакторі рисунка мнемосхеми до технологічного процесу завданого хімічного виробництва;
- підготовку в текстовому редакторі *Word* опису схеми технологічного процесу до завданого хімічного виробництва;
- розробку у графічному редакторі рисунка схеми конструкції технологічного апарату № 1;
- розробку у програмі графічного редактора рисунка схеми конструкції технологічного апарату № 2;
- розробку у програмі графічного редактора рисунка схеми конструкції технологічного апарату № 3;
- підготовку в текстовому редакторі *Word* опису технологічного процесу в апараті № 1;
- підготовку в текстовому редакторі *Word* опису технологічного процесу в апараті № 2;
- підготовку в текстовому редакторі *Word* опису технологічного процесу в апараті № 3;

- аналіз значень вимірюваних технологічних параметрів у апаратів для програмування графіків їх зміни у часі.

Другій етап передбачає роботу студента по програмуванню C++ програми до індивідуального завдання з модульної контрольної роботи з урахуванням створених матеріалів на першому етапі.

На даному етапі передбачається програмування прикладної C++ програми до модульної контрольної роботи та її підготовки до захисту. Розроблена прикладна C++ програма з контрольної роботи подається до захисту з наступними матеріалами:

- виконавчого файлу (*.exe) прикладної C++ програми з основним меню команд, побудованим на основі компоненти *MainMenu*;
- файлів проекту *ProjectMKP2.bpr* до прикладної C++ програми та програмних модулів до віконних форм;
- записки з рисунками схем, описами процесів, блок-схемами алгоритмів та листингами розробленої C++ програми, зображення вікон з екрану дисплея комп'ютера (скріншоти).

Навчальний приклад, що буде розглядатися у посібнику передбачає виконання модульної контрольної роботи на тему: «*Обробка подій мишки у меню команд прикладної C++ програми до мнемосхеми технологічного процесу з виробництва хлорметанів*».

Відповідно до схеми технологічного процесу з виробництва хлорметанів необхідно для роботи у Windows розробити прикладну C++ програму в програмувальному середовищі C++ *Builder* і за допомогою компоненти *MainMenu* побудувати меню команд зі структурою, показаною на рис. 4-2.

Після запуску у роботу виконавчого файлу (*.exe) прикладної C++ програми з модульної контрольної роботи на екрані дисплея комп'ютера повинно з'явитися в основному вікні програми зображення заставки з відповідними написами, згідно прикладу на (рис.4-1).

Якщо користувач в прикладній C++ програмі у меню “Технологія” (рис. 4-2) виконує команду “Мнемосхема”, то в цьому випадку заставка у прикладній

C++ програми замінюється на рисунок мнемосхеми технологічного процесу виробництва хлорметанів і цей рисунок далі стає фоном основного вікна *Form1* і знизу у головному вікні повинні з'являтися підказки назв до технологічних апаратів і обладнання, на яких буде встановлено «стрілка» мишки.

**Модульна контрольна робота № 2 кредитного модуля
“Візуальне програмування прикладних програм”
навчальної дисципліни “Технології розробки програмного забезпечення – 2”**

**на тему: «Обробка подій мишки у меню команд прикладної
C++ програми до мнемосхеми технологічного
процесу виробництва хлорметанів»**

Розробник C++ програми студент(ка)

П.І.Б.

ІХФ гр. ЛА-№ XX

Київ
КПІ ім. Ігоря Сікорського
20_____ р.

Рис. 4-1. Приклад заставки C++ програми до навчального прикладу модульної контрольної роботи.

Коли користувач у прикладі C++ програми обирає наступні команди:

- «Апарати /Хлоратор / Схема конструкції», або «Апарати /Хлоратор / Опис процесу», або «Апарати /Хлоратор / Параметри», тоді інформація до цих команд повинна показуватись в окремому вікні C++ програми;
- «Апарати /Реактор / Схема конструкції», або «Апарати /Реактор / Опис процесу», або «Апарати /Реактор / Параметри», тоді також інформація до цих команд повинна показуватись в окремому вікні C++ програми;
- При виконанні у меню “Апарати” команди “Хлоратор / Параметри” повинні показуватися в окремому вікні графіки зміни параметрів процесу в технологічному апараті «Хлоратор»;

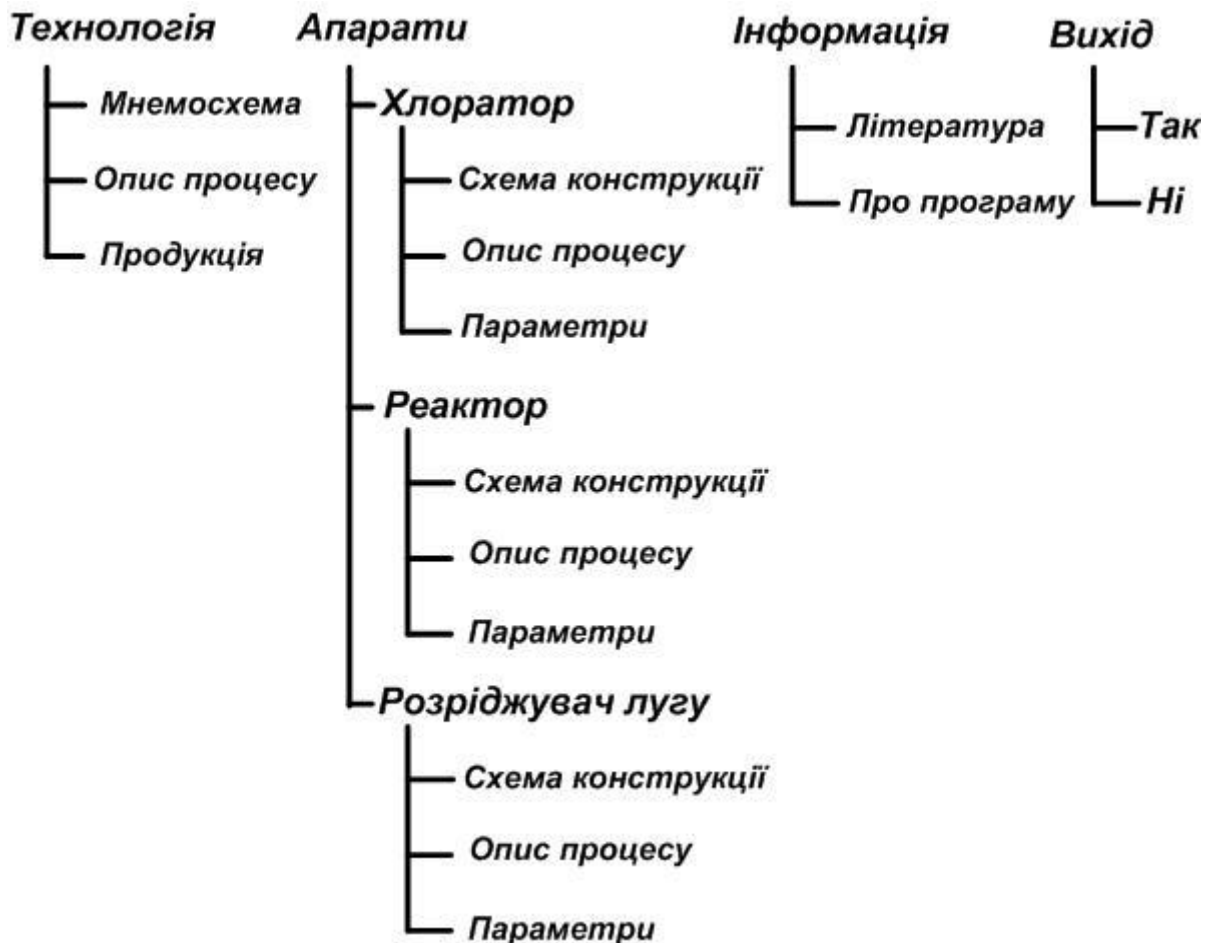


Рис. 4-2. Меню команд C++ програми до прикладу модульної контрольної роботи № 2.

- При виконанні у меню “Апарати” команди “Реактор / Параметри” повинні показуватися в окремому вікні графіки зміни параметрів процесу в технологічному апараті «Реактор»;
- Результати, листингі програми, матеріали та блок-схеми до алгоритмів з програмування прикладної C++ програми модульної контрольної роботи для захисту оформлюються у вигляді пояснювальної записки.

4.2 ПРИКЛАД З РОЗРОБКИ МАТЕРІАЛІВ ДО ПЕРШОГО ЕТАПУ ВИКОНАННЯ ЗАВДАНЬ МОДУЛЬНОЇ КОНТРОЛЬНОЇ РОБОТИ

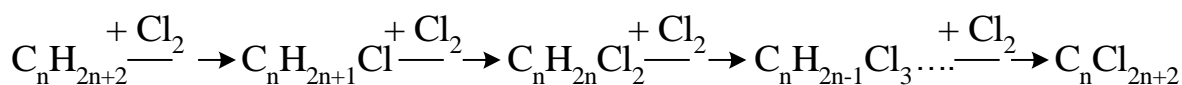
Методика і приклад виконання модульної контрольної роботи буде далі розглядатися у вигляді необхідних та відповідних кроків і дій для підготовки матеріалів до програмування прикладної C++ програми відповідного завдання

на тему: «Обробка подій мишки у меню команд прикладної C++ програми до мнемосхеми технологічного процесу виробництва хлорметанів».

4.2.1 НАВЧАЛЬНИЙ ПРИКЛАД.

Схема і опис технологічного процесу виробництва хлорметанів

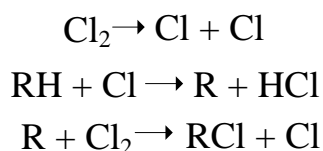
При хлоруванні парафінових вуглеводнів у реакції атоми хлору заміщають атоми водню, які відщеплюються, утворюючи хлористий водень. При цьому можна одержати моно-, ди-, три- і поліхлороподібні відповідно до таких реакцій:



Процес такого заміщення був відкритий французьким ученим Дюма і називається металепсією. При безпосередній взаємодії хлору з парафіновими вуглеводнями (або з боковими ланцюгами алкіл бензолів) молекула хлору насамперед дисоціює на атоми, які потім взаємодіють з вуглеводнем і при цьому утворюється вільний радикал, реагуючи з іншою молекулою хлору, утворює хлорпохідне вуглеводню, а один атом хлору звільняється.

Дисоціацію хлору на атоми можна спричинити тепловою або променистою енергією. Практично хлорування проводять у газовій та рідкій фазах з застосуванням каталізаторів або без них. Рідко фазне хлорування може здійснюватися також у розчині, в середовищі розплавлених солей.

Таким чином, хлорування парафінових вуглеводнів відбувається за ланцюговою реакцією:

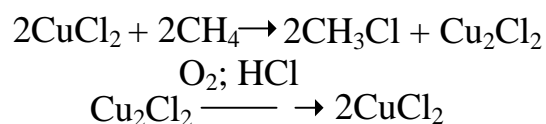


Фотохімічний метод хлорування парафінових вуглеводнів не набув значного поширення в промисловості через високу вартість потрібного устаткування. Основним способом є термічне хлорування в газовій фазі при температурі, достатній для помітної дисоціації хлору (понад 250-300 градусів). Реакція

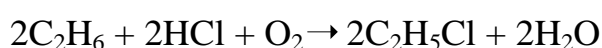
хлорування проходить з виділенням тепла, яке використовують для підігрівання реагентів. Реакція термічного хлорування відбувається з великою швидкістю. Температурний режим процесу підтримується за рахунок фізичного тепла надлишкового вуглеводню. Щоб уникнути місцевих перегрівів, що призводять до розщеплення молекул парафінів необхідне старанне змішування хлору з вуглеводнем і для цього хлор вводять у реакційну зону через сопла, розміщені в напрямку потоку вуглеводню.

Каталітичне хлорування парафінових вуглеводнів відбувається звичайно при нижчих температурах, ніж термічне хлорування, тому що каталізатори прискорюють утворення дихлорпохідних і продуктів за рахунок більш повного заміщення атомів водню хлором. В якості каталізаторів застосовують хлориди міді, сурми, олова, кремнію, йоду, сірки, які нанесені на високо пористі матеріали (активне вугілля, пемза, силікагель та ін.). Досить активним каталізатором є, наприклад, хлорна мідь CuCl_2 , нанесена на високо пористий матеріал.

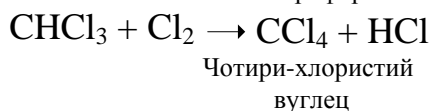
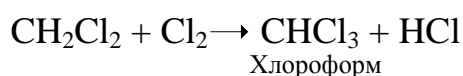
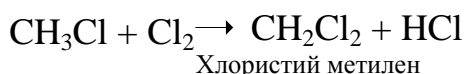
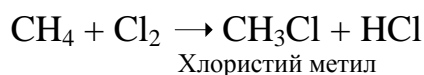
Хлорпохідні вуглеводнів утворюються при взаємодії вуглеводнів не тільки з елементарним хлором, але і з деякими його сполуками, наприклад, з хлористим сульфурилом SO_2Cl_2 , фосгеном COCl_2 і т.д.. Так, газоподібні вуглеводні можна хлорувати при 400 градусах хлорною міддю, яка в тонкоподрібненому стані суспендується в газі. Хлористу мідь, яка при цьому утворюється, окислюють повітрям при 325 - 400 градусах і обробляють хлористим воднем:



Регеновану хлорну мідь знову використовують для хлорування. Великий практичний інтерес становить метод окислювального хлорування вуглеводнів хлористим воднем, наприклад, одержання хлорпохідних пропусканням суміші вуглеводню, хлористого водню і кисню над каталізатором:

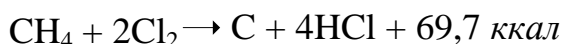


Використовуючи для цього процесу хлористий водень, який є побічним продуктом хлорування парафінових вуглеводнів газоподібним хлором, можна було б значно підвищити економічність виробництва аліфатичних хлорорганічних сполук. Такі процеси добре розроблені для ароматичного ряду сполук. При вивченні процесів хлорування парафінових вуглеводнів було відкрито ряд закономірностей, які справедливі для більшості реакцій. Встановлено, що коли температура хлорування нижча за температуру розщеплення хлорованого вуглеводню, хлорпохідне, яке при цьому утворюється, має стільки ж вуглецевих атомів, скільки вихідний вуглеводень. При більш високих температурах хлорування і надлишку хлору молекула вуглеводню розщеплюється на радикали (CH_3^* , C_2H_5^*), які потім перетворюються в полі хлорпохідні CCl_4 , C_2Cl_6 . Хлорування з одночасним розщепленням молекули вуглеводню називається деструктивним хлоруванням. При помірних температурах швидкість заміщення на хлор атомів водню у третинних вуглецевих атомів більша, ніж у вторинних, а у вторинних – більша ніж у первинних. З підвищенням температури хлорування ця різниця зменшується і при 600 градусах заміщення відбувається з однаковою швидкістю. При підвищенні тиску в процесі хлорування вуглеводнів у газуватій фазі збільшується відносна швидкість заміщення водню при первинному вуглецевому атомі. При безпосередньому хлоруванні потоку метану не вдається одержати яке-небудь одне індивідуальне хлорпохідне, тому що практично відповідно до хімічних реакцій завжди одержується суміш всіх чотирьох хлорпохідних:



Створюючи певні умови процесу, можна спрямувати реакцію в бік утворення переважного потрібного хлорпохідного. Процес здійснюється при великому молярному надлишку метану і порівняно при високій температурі (400 °C і вище), якщо потрібно одержати головним чином хлористий метил CH_3Cl , а для переважного одержання чотири хлористого вуглецю потрібно надлишок хлору.

При хлоруванні метану, як і інших вуглеводнів, слід уникати значного підвищення температури. При температурі вище 500 - 550 градусів може статись вибух з виділенням вуглецю і хлористого водню:



Тому необхідно уникати місцевих перегрівів реагуючих газів. Виходи різних хлорпохідних метану залежно від умов процесу наведені у таблиці 4.1-1. Хлористий метил (монохлорметан) можна одержати хлоруванням метану при 400-450 градусах в присутності каталізатора (хлориди металів, осаджені на пемзі) при десятикратному молярному надлишку метану. В цих умовах приблизно 80-85 % хлору, який надходить на хлорування, витрачається на утворення хлористого метилу.

Хлористий метил являє собою безбарвний газ, з запахом ефіру, горить безбарвним полум'ям; температура кипіння - 23,7 градусів; температура замерзання - 97,6 градусів; питома вага хлористого метилу при температурі кипіння дорівнює 0,992 г/см³; прихована теплота випаровування 430,5 кдж/кг. В 100 г води розчиняється 0,74 г CH_3Cl . Хлористий метил застосовується як холодоагент у холодильних установках. В промисловості органічного синтезу він використовується як метилюючий засіб. При пропусканні хлористого метилу з водяною парою над вапном при 300-350 градусах можна одержати з хорошим виходом метиловий спирт.

Таблиця 4.1-1

Виходи хлорпохідних метану					
Температура , °C	Молярне відношення Cl ₂ : CH ₄	Склад продуктів реакції, % мол.			
		CH ₃ Cl	CH ₂ Cl ₂	CHCl ₃	CCl ₄
440	1:2	62	30	7	1
440	1,1:1	37	41	19	3
440	1,68:1	19	43	33	4
440	1,98:1	11	35	45	9
440	2,28:1	5	29	52	14
440	3,02:1	3	15	53	29
440	3,31:1	—	6	43	51
460	3,88:1	—	—	4	96

Хлористий метилен (метиленхлорид, дихлорметан) CH₂Cl₂ одержують з хорошим виходом хлорування суміші метану та хлористого метилу. Хлористий метил і частина метану хлорують до метиленхлориду, а інша частина метану – до хлористого метилу. З продуктів реакції добувають хлористий метилен, а газ, що лишився (метан + хлористий метил), додають до свіжого газу, який містить метан, нагрівають суміш до 130 градусів і подають на змішування з хлором. Одержана суміш містить близько 20 % об'ємного хлору, 60 % метану (відношення метану до хлору 3:1), 6 % хлористого метилу і 0,1-0,2 % хлористого метилу, решта – азот, окис і двоокис вуглецю. Ця суміш газів і парів при 100 градусах надходить у хлоратор, де здійснюється хлорування при температурі близько 500 градусів.

Хлоратор являє собою вертикальний хімічний реактор 4 на рис. 4.1-1 у вигляді циліндричного апарату з сталевим корпусом, який футерований зсередини двома

шарами діабазових плиток і шаром шамотної цегли. Ззовні хлоратор ізольований шаром азбесту.

Перед введенням у хлоратор реакційних газів в топці спалюють горючий газ і продукти згоряння нагрівають футеровку апарату і цегляну насадку, які акумулюють тепло. Після досягнення потрібної температури в апарат 4 через верхній штуцер вводять суміш вуглеводнів і хлору. Нагрівання газів потрібне тільки для ініціювання реакції, а далі екзотермічна реакція хлорування відбувається автотермічно. Продукти реакції відводяться з нижньої частини хлоратора. В процесі хлорування незначна частина метану зазнає глибоких перетворень, при цьому утворюються смолисті продукти і сажа. Щоб запобігти потраплянню цих речовин у відповідний газопровід, в кільцевому просторі хлоратора укладається шар керамічних кілець висотою близько 200 мм. Періодично, в міру нагромадження в хлораторі сажі та смолистих речовин, їх випалюють. Для цього припиняють подачу реакційних газів у хлоратор, запалюють у топці горючий газ і подають продукти його згоряння разом з надлишковим повітрям в камеру хлорування.

Після видалення речовин, що містять вуглець, вогонь у топці гасять і в хлоратор знову подають реакційні гази. Продукти хлорування відводять з нижньої частини реактора 4, пропускають через сажовловлювач 5 і охолоджують у повітряному холодильнику 6 до 100 градусів.

У газоподібних продуктах хлорування міститься приблизно 55-57 % метану, що не прореагував, 17-18 % хлористого водню, 9-10 % хлористого метилу, 5-6 % метиленхлориду і близько 1,5 % вищих хлоридів, а решта це азот, окис і двоокис вуглецю.

З потоку газоподібних продуктів реакції насамперед добувають хлористий водень. Для цього використовуються три колонних апарати, з яких два перших (за ходом газів) апарати 7 і 8 є абсорберами, а апарат 9 здійснює нейтралізацію слідів хлористого водню. Абсорбер 8 зрошують розбавленою соляною кислотою. Концентрована соляна кислота, що витікає з абсорбера, охолоджується до 20-30 градусів у графітовому холодильнику 10. Гази, які відходять з абсорбера 7,

подають у другий абсорбер 8 для добування з них залишків хлористого водню, а розведена соляна кислота, що утворюється, надходить на зрошення абсорбера 7.

Для нейтралізації слідів хлористого водню газу промивають в колоні 9 холодним 8 % водним розчином лугу. Одночасно у процесі нейтралізації газу охолоджуються.

Для видалення хлорметанів газу, відділених від хлористого водню, стискають в одноступінчастому компресорі 12 до 2 – 8 атм. і пропускають в осушувачах 13 через твердий луг, який відбирає з газів вологу. Потім газу охолоджують в кожухотрубному конденсаторі 14 аміаком, що випаровується при $-53\text{ }^{\circ}\text{C}$ і при цьому відбуваються виморожування вологи та конденсація хлорметанів.

Після відокремлення льоду на фільтрі 15 і рідких хлорметанів у сепараторі 16 газу повертають на хлорування. Рідка суміш хлорметанів має приблизно такий склад (в % ваг.):

Хлористий метил 28 – 32; Хлороформ 12 – 14;
Метиленхлорид 50 – 53; Чотирихлористий вуглець ... 3 – 5.

З цієї суміші ректифікацією одержують індивідуальні сполуки. Хлористий метил повертають на хлорування (повністю або частково).

Хлористий метилен являє собою безбарвну рідину з запахом хлороформу; він кипить при $40,1\text{ }^{\circ}\text{C}$, замерзає при $-96,6\text{ }^{\circ}\text{C}$, пит. вага $1,335\text{ г/см}^3$ (при $15\text{ }^{\circ}\text{C}$); прихована теплота випаровування $78,7\text{ ккал/кг}$.

Метиленхлорид добре розчиняє ефіри целюлози, жири, масла і каучук, важко гідролізується, негорючий, його пари не утворюють вибухових сумішей з повітрям. Тому він є цінним промисловим розчинником і може використовуватись також для очистки мастил від парафіну.

Температура замерзання - $97,6$ градусів; питома вага хлористого метилу при температурі кипіння дорівнює $0,992\text{ г/см}^3$; прихована теплота випаровування $102,5\text{ ккал/кг}$. В 100 г води розчиняється $0,74\text{ г}$ CH_3Cl . Хлористий метил застосовується як холодоагент у холодильних установках. В промисловості органічного синтезу він використовується як метилуючий засіб.

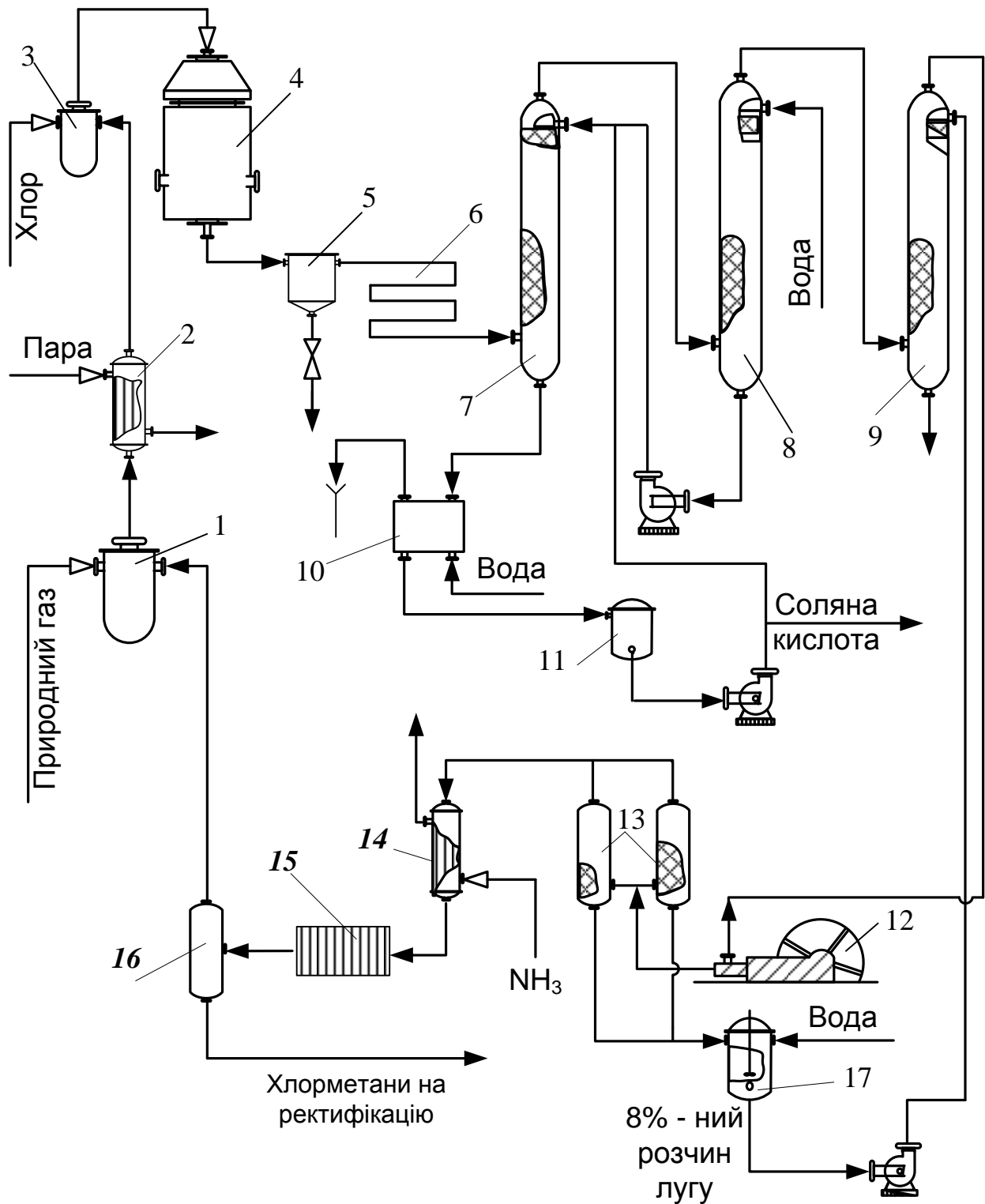


Рис. 4-3. Технологічна схема процесів виробництва хлорметанів:

1, 3 – змішувачі; 2 – підігрівник; 4 – хлоратор; 5 – сажовловлювач; 6 – повітряний холодильник; 7, 8 – абсорбери; 9 – нейтралізаційна колона; 10 – графітовий холодильник; 11 – збірник концентрованої соляної кислоти; 12 – компресор; 13 – осушувач; 14 – конденсатор; 15 – фільтр; 16 – сепаратор; 17 – розіджувач луѓу.

Хлористий метил являє собою безбарвний газ, з запахом ефіру, горить безбарвним полум'ям; температура кипіння - 23,7 градусів; температура

На рис. 4-3. зображена схема технологічна процесу для розробки прикладної C++ програми “Мнемосхема технологічного процесу виробництва хлорметанів” і далі потрібно виконати наступні завдання для меню команд у меню «Технологія»:

- Завдання № 1. Створити файл (.doc) з текстом опису технологічного процесу виробництва хлорметанів відповідно до схеми рис. 4-3;
- Завдання № 2. Розробити у графічному редакторі рисунок мнемосхеми процесу виробництва хлорметанів відповідно до її схеми на рис. 4-3.
- Завдання № 3. Створити файл (.doc) з текстом опису властивостей хімічної продукції – суміш хлорметанів.

Приклад виконання завдання № 1. На основі тексту з опису процесів переробки углеводів у процесі виробництва хлорметанів методом хлорування метану, наведених у пункті 4.2.1, формуємо у текстовому редакторі **Word** файл тексту для виводу в вікні **Form2** при виборі у меню “Технологія” команди “Опис схеми” .

Приклад до завдання № 1

Листинг з описом технологічного процесу з виробництва хлорметанів

Перед введенням у хлоратор реакційних газів в топці спалюють горючий газ; продукти згоряння нагрівають футеровку апарата і цегляну насадку, які акумулюють тепло. Суміш газів після змішувачів 1 і 3 містить близько 20% об'ємного хлору, 60% метану (відношення метану до хлору 3:1), 6% хлористого метилу і 0,1-0,2% хлористого метилену, решта — азот, окис і двоокис вуглецю та ін. Ця суміш газів і парів при 100 градусах надходить у реактор (хлоратор), де проводять хлорування при температурі близько 500 градусів. Після досягнення потрібної температури в апарат через верхній штуцер вводять суміш углеводнів і хлору. Нагрівання потрібне тільки для ініціювання реакції; далі екзотермічна реакція хлорування відбувається авто термічно. Продукти реакції відводяться з нижньої частини апарата.

В процесі хлорування незначна частина метану зазнає глибоких перетворень, при цьому утворюються смолисті продукти і сажа. Щоб

запобігти потраплянню цих речовин у відповідний газопровід, в кільцевому просторі хлоратора укладаються шар керамічних кілець висотою близько 200мм.

Періодично, в міру нагромадження в хлораторі сажі та смолистих речовин, їх випалюють. Для цього припиняють подачу в хлоратор реакційних газів, запалюють у топці горючий газ і подають продукти його згоряння разом з надлишковим повітрям в камеру хлорування.

Після видалення речовин, що містять вуглець, вогонь у топці гасять і в хлоратор знову подають реакційні гази. Продукти хлорування відводять з нижньої частини реактора 4, пропускають через сажовловлювач 5 і охолоджують у повітряному холодильнику 6 до 100 градусів.

У газоподібних продуктах хлорування міститься приблизно 55-57% об'ємно метану, що не прореагував, 17-18% хлористого водню, 9-10% хлористого метилу, 5-6% метиленхлориду і близько 1.5% вищих хлоридів, решта-азот, окис і двоокис вуглецю тощо.

З газоподібних продуктів реакції насамперед добувають хлористий водень. Для цього встановлюють три колонних апарати, з яких два перших (за ходом газів) апарати 7 і 8 є абсорберами, в третьому апараті 9 проводиться нейтралізація слідів хлористого водню.

Абсорбер 8 зрошують розбавленою соляною кислотою. Концентрована соляна кислота, що витікає з абсорбера, охолоджується до 20-30 градусів у графітовому холодильнику 10.

Гази, які відходять з абсорбера 7, подають у другий абсорбер 8 для добування з них залишків хлористого водню; розведена соляна кислота, що утворюється, надходить на зрошення абсорбера 7. Для нейтралізації слідів хлористого водню гази промивають в третій колоні 9 холодним 8%-ним водним розчином лугу. Одночасно з нейтралізацією гази охолоджуються.

Вище виделений текст, як листинг до опису процесів з виробництва хлорметанів зберігаємо у файлі з назвою **Scheme.rtf** і розміщуємо з доступом у наступних каталогах КР2-РMain_menu\Хлорування метану\Data\Main\.

Приклад виконання завдання № 2. За допомогою технологічної схеми (рис. 4-3) з виробництва хлорметанів у графічному редакторі робимо рисунок до мнемосхеми процесу виробництва хлорметанів, який показано на рис. 4-4. При виконанні у С++ програмі команди “Схема” з меню “Технологія” буде з’являтися рис. 4-4 у вікні основної форми **Form1** і далі цей рисунок буде залишатися у вікні як загальний фон. При наведенні мишки на якійсь апарат на мнемосхемі буде з’являтися підказка у вигляді назви апарату. Вище наведений рисунок 4-4 мнемосхеми виробництва хлорметанів зберігаємо у файлі з назвою

Scheme.bmp і розміщуємо з доступом у наступних каталогах KP2-

PMain_menu\Хлорування метану\Data>Main\ .

Приклад виконання завдання № 3. На основі тексту з опису процесів хлорування метану з виробництва суміши хлорметанів, наведених у пункті 4.2, у текстовому редакторі **Word** формуємо файл з текстом опису продукції для виводу у вікні **Form2** при виборі команди “Продукція” у меню “Технологія”.

Приклад до завдання № 3

Листинг з описом хлорметану, як хімічної продукція хлорування метану.

Практично не розчинний у воді (2 г/л), змішується з більшістю органічних розчинників. Легколетучий (40 °С), утворює азеотропну суміш із водою (т.стосів. 38,1 °С, 98,5 % М.). Метиленхлорид (дихлорметан) реагує із хлором з утвором хлороформу й чотирьоххлористим вуглецем. З йодом при 200 °С дає CH_2Cl_2 , із бромом при 25-30 °С у присутності алюмінію дає бромхлорметан. При нагріванні з водою гідролізується до CH_2PRO и HCl . При нагріванні зі спиртовим розчином NH_3 до 100-125 °С утворює гексаметилентетрамин. Реакція з водяним розчином NH_3 при 200 °С приводить до метиламіну, мурашиної кислоти й HCl . З ароматичними з'єднаннями в присутності AlCl_3 метиленхлорид вступає в реакцію Фріделя - Крафтса, наприклад з бензолом утворює дифенілметан. Дешевина, висока здатність розчиняти багато органічних речовин, легкість видалення, відносно мала токсичність привела до широкого застосування його як розчинник для проведення реакцій, екстракцій у тому числі й у лабораторіях. Використовують у сумішах для зняття лаку, знежирення поверхонь. У харчовій промисловості використовують для готування швидкорозчинного кави, екстракту хмелю й інших харчових препаратів. Для розчинення смол, жирів, бітуму. Його висока летючість використовується для зпінення поліуретанів. Також використовується в хроматографії.

Вище виделений текст з описом властивостей хлорметанів, як продукції хімічного виробництва, зберігаємо у файлі з назвою **Products.rtf** з доступом у каталогах KP2-PMain_menu\Хлорування метану\Data>Main\ .

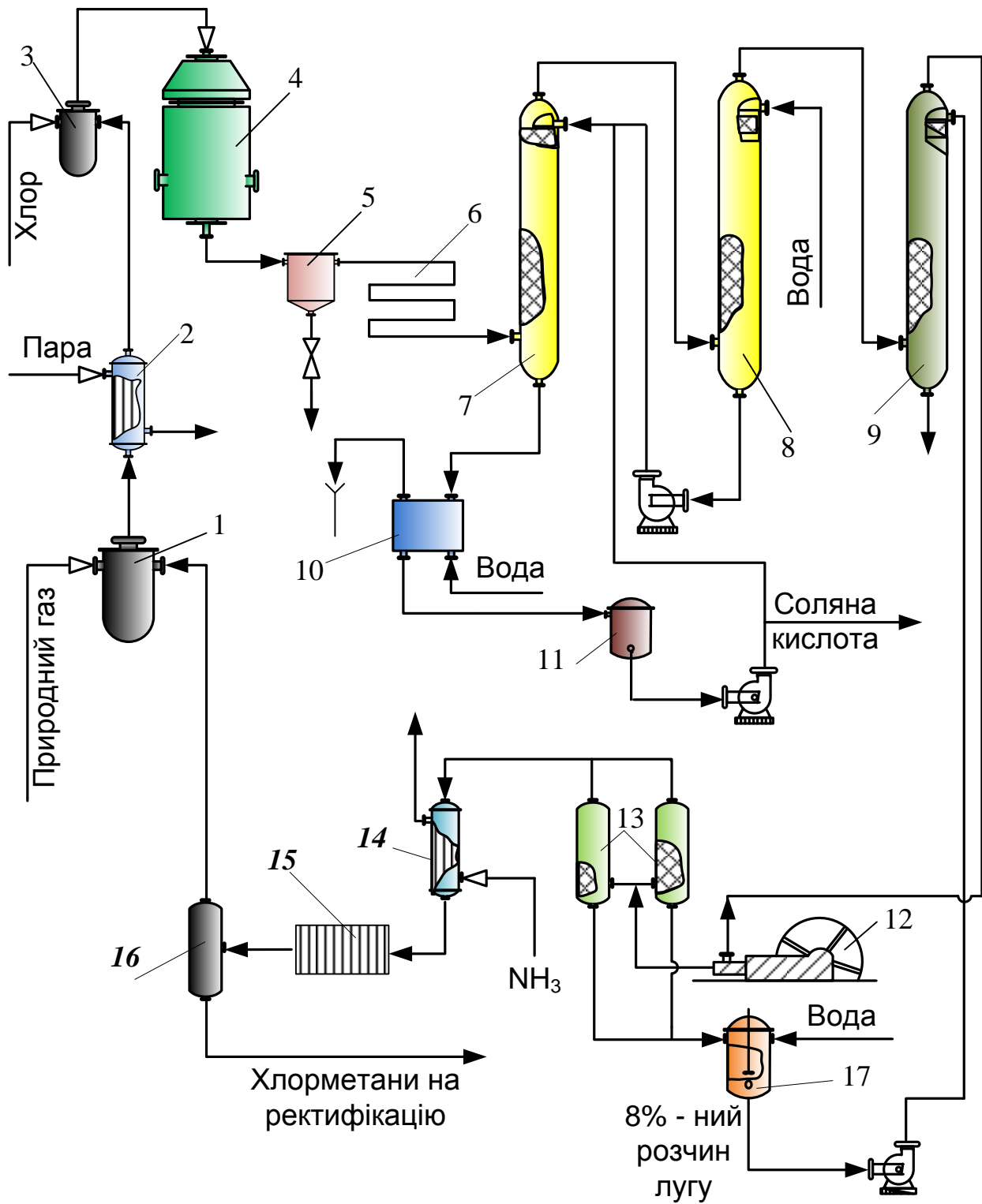


Рис. 4-4. Мнемосхема технологічного процесу виробництва хлорметанів (приклад до завдання № 2).

4.2.2 НАВЧАЛЬНИЙ ПРИКЛАД ДО РОЗРОБКИ І ОФОРМЛЕННЯ МАТЕРІАЛІВ ДЛЯ МЕНЮ «АПАРАТИ»

Для меню «АПАРАТИ» відповідно до навчального прикладу модульної контрольної роботи потрібно виконати наступні завдання до рисунків та текстових описів для технологічних процесів у апаратах.

Схема конструкції і текст опису процесу в технологічному апараті “Хлоратор”

- Завдання № 1. Розробити рисунок схеми з конструкції технологічного апарату “Хлоратор”;
- Завдання № 2. Створити файл (.doc) з текстом опису технологічного процесу в апараті “Хлоратор”.

Приклад виконання завдання № 1. За допомогою графічного редактора робимо рисунок схеми конструкції технологічного апарату “Хлоратор”, який показан на рис. 4-5. При виконанні команди “Конструкція” в підменю “Хлоратор” у меню команд “Апарати” буде у вікні відповідної форми з’являтися рис. 4-5.

Рисунок схеми конструкції технологічного апарату “Хлоратор” (рис. 4-5) зберігаємо у файлі з назвою **Device-1.bmp** і розміщуємо з доступом у наступних каталогах КР2-РMain_menu\Хлорування метану\Data\Device1\ .

Приклад виконання завдання № 2. За допомогою текстового редактора **Word** текст опису технологічного процесу в апараті “Хлоратор” формуємо у файл для виводу у відповідному вікні форми при виборі команди “Опис процесу” у підменю “Хлоратор” з меню “Апарати”.

Приклад виконання завдання № 3. За допомогою графічного редактора робимо рисунок схеми конструкції технологічного апарату “Абсорбер”, який показано на рис. 4-6. При виконанні команди “Конструкція” в підменю “Абсорбер” у меню команд “Апарати” буде у вікні відповідної форми з’являтися рис. 4-6.

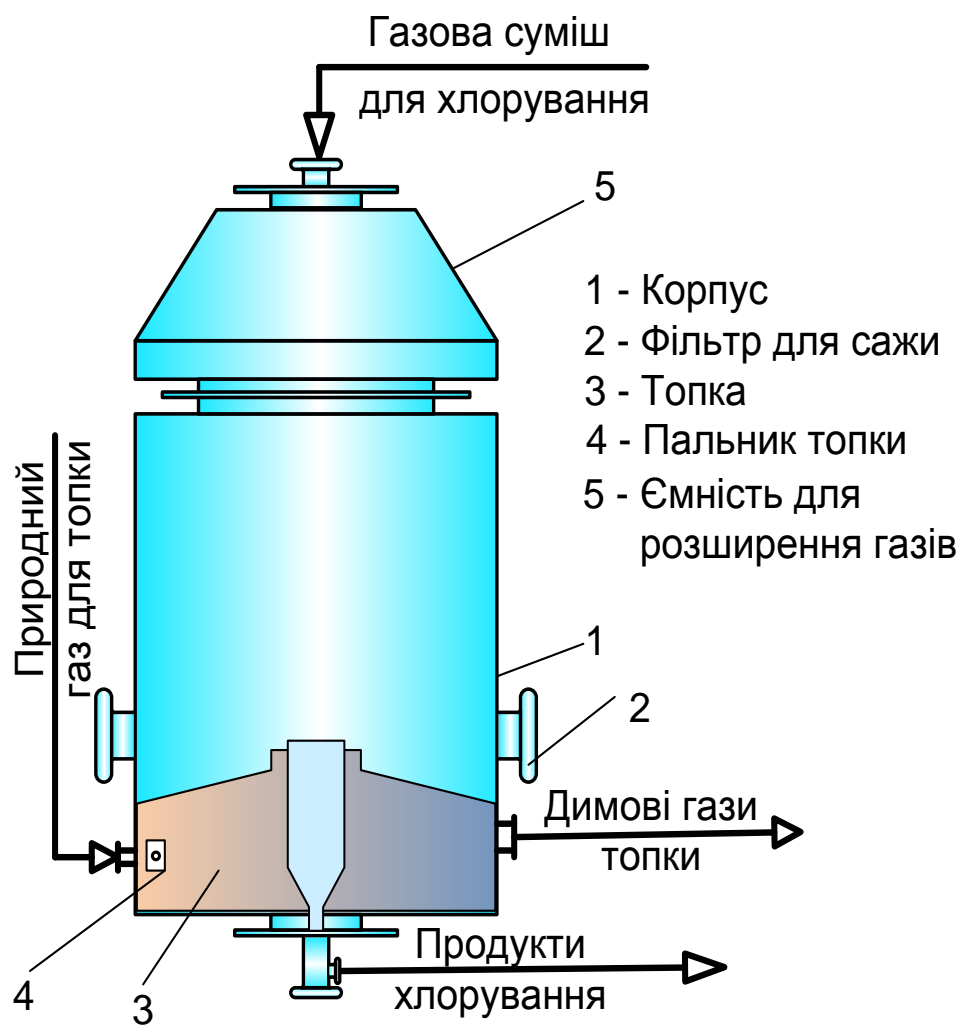


Рис. 4-5. Схема конструкції технологічного апарату “Хлоратор”.

Опис технологічного процесу в хлораторі

Потік суміші у хлоратор містить близько 20 % об'ємного хлору, 60 % метану (відношення метану до хлору 3:1), 6 % хлористого метилу і 0,1-0,2 % хлористого метилену, решта – азот, окис і двоокис вуглецю. Ця суміш газів і парів при 100 градусах надходить у реактор (хлоратор), де здійснюють хлорування при температурі близько до 500 градусів.

Перед введенням реакційних газів у хлоратор в топці спалюють горючий газ і продукти згоряння нагрівають футеровку апарату і цегляну насадку, які акумулюють тепло. Після досягнення потрібної температури в апарат через верхній штуцер подається суміш вуглеводнів і хлору. Нагрівання потрібне тільки для ініціювання реакції, а далі екзотермічна реакція хлорування відбувається автотермічно. Продукти реакції відводяться з нижньої частини апарату.

В процесі хлорування незначна частина метану зазнає глибоких перетворень, при цьому утворюються смолисті продукти і сажа. Щоб запобігти потраплянню цих речовин у відповідний газопровід, в кільцевому просторі хлоратора укладаються шар керамічних кілець висотою близько до 200 мм. Періодично, в міру нагромадження в хлораторі сажі та смолистих речовин, їх випалюють. Для цього припиняють подачу в хлоратор реакційних газів, запалюють у топці горючий газ і подають продукти його згоряння разом з надлишковим повітрям в камеру хлорування.

Після видалення речовин, що містять вуглець, у топці вогонь гасять і в хлоратор знову подають реакційні гази. Продукти хлорування відводять з нижньої частини реактора, пропускають через сажовловлювач і охолоджують у повітряному холодильнику до 100 градусів.

У газоподібних продуктах хлорування міститься приблизно 55-57 % об'ємно метану, що не прореагував, 17-18 % хлористого водню, 9-10 % хлористого метилу, 5-6 % метиленхлориду і близько 1,5 % вищих хлоридів, решта це азот, окис і двоокис вуглецю.

Виделений вище текст з опису технологічного процесу в апараті “Хлоратор” зберігаємо у файлі з назвою **Process-1.rtf** і розміщуємо з доступом у наступних каталогах КР2-РMain_menu\Хлорування метану\Data\Device1.

Схема конструкції і текст опису процесу в технологічному апараті “Абсорбер”

- Завдання № 3. Розробити рисунок схеми з конструкції технологічного апарату “Абсорбер”;
- Завдання № 4. Створити файл (.doc) з текстом опису технологічного процесу в апараті “Абсорбер”.

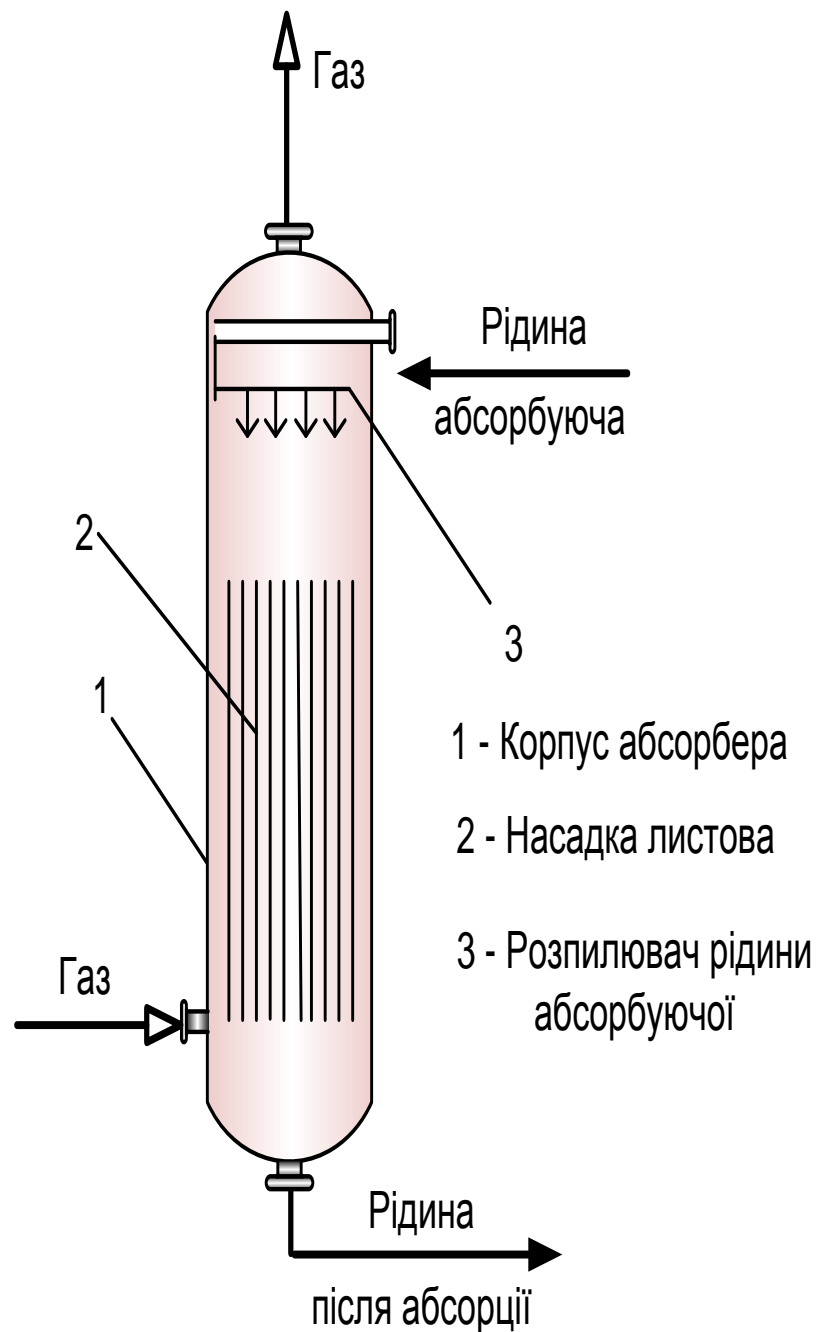


Рис. 4-6. Схема конструкції технологічного апарату “Абсорбер”.

Рисунок схеми конструкції технологічного апарату “Абсорбер” (рис. 4-6) зберігаємо у файлі з назвою **Device-2.bmp** і розміщуємо з доступом у наступних каталогах КР2-РMain_menu\Хлорування метану\Data\Device2\ .

Приклад виконання завдання № 4. За допомогою текстового редактора **Word** текст з описом технологічного процесу у апараті “Абсорбер” формуємо у файл для виводу у вікні відповідної форми при виборі команди “Опис процесу” в підменю “Абсорбер” у меню команд “Апарати”.

Виделений вище текст з описом технологічного процесу в апараті “Абсорбер” зберігаємо у файлі з назвою **Process-2.rtf** і розміщуємо з доступом у каталогах КР2-РMain_menu\Хлорування метану\Data\Device2\ .

- **Завдання № 5.** Розробити рисунок схеми з конструкції технологічного апарату “Розріджувач лугу”;
- **Завдання № 6.** Створити файл (.doc) з текстом опису технологічного процесу в апараті “Розріджувач лугу”.

Схема конструкції і текст опису процесу в технологічному апараті “Розріджувач лугу”

Приклад виконання завдання № 5. За допомогою графічного редактора робимо рисунок схеми з конструкції технологічного апарату “Розріджувач лугу”, який показано на рис. 4-7. При виконанні команди “Конструкція” у підменю “Розріджувач лугу” в меню “Апарати” буде з’являтися рис. 4-7 у вікні відповідної форми **C++** програми.

Рисунок схеми з конструкції технологічного апарату “Розріджувач лугу” зберігаємо у файл з назвою **Device.bmp** і розміщуємо з доступом у каталогах КР3MainMenu \ Хлорування метану \Data \ Device3 \ для розглядаємого навчального прикладу до модульної контрольної роботи.

Приклад виконання завдання № 6. За допомогою текстового редактора **Word** текст з описом технологічного процесу у апараті “Розріджувач лугу” формуємо

у файл для виводу у вікні відповідної форми при виборі команди “Опис процесу” в підменю “Розріджувач лугу” у меню команд “Апарати”.

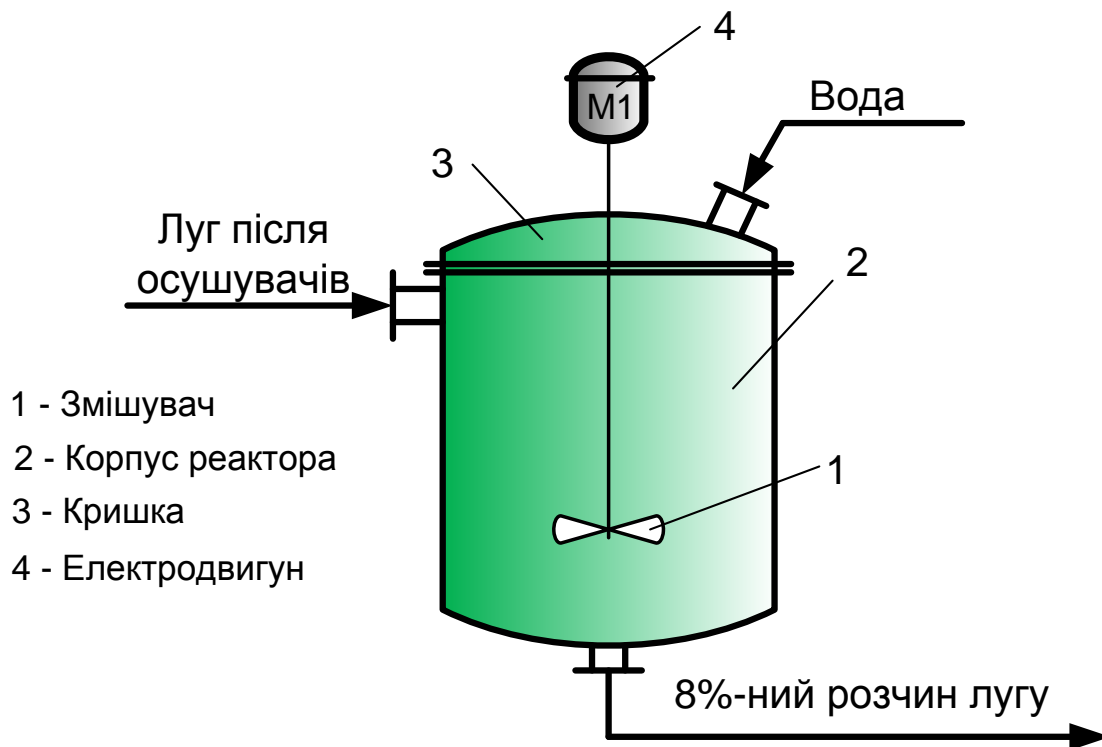


Рис. 4 -7. Схема конструкції апарату “Розріджувач лугу”.

Виделений вище у рамці текст буде описом технологічного процесу в апараті “Розріджувач лугу”. Зберігаємо цей текст у файл з назвою **Process.rtf** з доступом у каталогах КРЗMainMenu \ Хлорування метану \Data \ Device3\ для навчального прикладу до модульної контрольної роботи.

Опис технологічного процесу у абсорбері

Абсорбцією називається процес поглинання індивідуального газу, а також виборчого поглинання одного чи декількох компонентів газової суміші рідким поглиначем - абсорбентом. Поглинання газу може відбуватися або за рахунок його розчинення в абсорбенті, або в результаті його хімічної взаємодії з абсорбентом. У першому випадку процес називають фізичною абсорбцією, а в другому випадку - хемосорбцією. Можливо також сполучення обих цих процесів. Абсорбуємі компоненти газової суміші називають абсорбтивом, а неабсорбуємі - инертом. Абсорбентами служать індивідуальні чи розчини рідини активного компонента в рідкому розчиннику. В усіх випадках до абсорбентів пред'являються ряд вимог. Серед яких істотними є: висока абсорбційна здатність, селективність, низький тиск парів, хімічна інертність стосовно матеріалів з котрих виготовлений корпус абсорбера. У хімічній технології абсорбція використовується для здійснення ряду процесів, до яких відносяться: - одержання цільових продуктів (абсорбція HCl водою для одержання соляної кислоти;

- виділення коштовних компонентів з газових сумішей;
- видалення шкідливих домішок із сумішей, диктуємоє ходом виробничого процесу, наприклад, домішки CO і CO₂.

Необхідно відзначити, що один і той же тип апарату залежно від умов роботи може працювати в різних режимах. Так, наприклад, абсорбер з насадкою може працювати як в плівковому режимі, так і в барботажному.

Поверхневі абсорбери. Поверхня контакту фаз в поверхневих абсорберах створюється за рахунок фіксованої поверхні: або дзеркала рідини (власне поверхностіє абсорбери), або поточної плівки рідини (плівкові абсорбери), тобто поверхня контакту фаз в апараті у відомо ступеню визначається площею елементу апарату (наприклад, насадки), хоча зазвичай і не рівна їй. Ці апарати можна розділити на наступні типи: поверхневі абсорбери з горизонтальним дзеркалом рідини; абсорбери(з нерухомою насадкою) насадок; плівкові абсорбери; механічні плівкові абсорбери. Апарати з рухомою насадкою займають проміжне положення між насадками і барботажними абсорберами.

Барботажні абсорбери. У барботажних абсорберах поверхня міжфазного контакту розвивається потоками газових цівок або бульбашок, що розподіляються по рідині. Поверхня контакту в таких апаратах визначається гідродинамічним режимом (витратами газу і рідини).

Опис технологічного процесу з розріджування лугу

Процес розріджування лугу виконується в хімічному реакторі методом змішування потоку після осушувачів з потоком води. За допомогою мешалки змішуються речовини і таким чином робиться 8 % розчин лугу для подачі на нейтралізаційну колону. Процес у реакторі це перемішування речовин однакового агрегатного стану з одержанням гомогенного 8 % розчину лугу. Методи перемішування, конструкції пристроїв, що перемішують, і їхні робочі режими залежать від агрегатного стану і фізичних властивостей речовин, що перемішуються, а також від вимог, пред'являємих до одержання суміші. Суміш може бути однофазної (розчин) чи двофазної (іноді багатфазної). Дуже часто зустрічаються двофазні суміші, у яких суцільною фазою являється рідина, дисперсною масою - дрібні краплі іншої нерозчинної рідини чи газові пухирці, тверді частки. В усіх випадках пристрій, що перемішує робочу масу в апараті (хімічному реакторі), повинен забезпечувати одержання однорідної суміші при максимальній продуктивності і мінімальній витраті енергії. У хімічних виробництвах широко застосовуються чотири методи перемішування (змішування) в рідких середовищах: 1) за допомогою механічних мішалок з обертальним чи коливальним рухом; 2) барбатажний - шляхом подачі в рідке середовище газу чи пари; 3) розмішування в потоці нерухомих турболізуючих пристроїв; 4) з використанням струминних і відцентрових насосів. Для процесу розріджування лугу в реакторі використовується механічна мішалка, тому розглянемо особливості роботи механічних мішалок у рідкому середовищі. Перемішування рідини будь-яким методом зводиться до багаторазового відносного переміщення елементів в обсязі маси. Складний рух рідини, що виникає в апараті (реакторі) при обертанні мішалки, можна розкласти на складові: радіальну (обертання уздовж радіуса); тангенціальну (по касательній до окружності, описуваної кінцем мішалки); осьову (уздовж осі вала). У мішалках різних конструкцій ці складові знаходяться в різних співвідношеннях. Існують, однак, мішалки, у яких дві чи навіть усі три складові порівнянні, тому класифікують механічні мішалки по їх конструктивних ознаках. Найбільш простою конструкцією є плоско-лопостна мішалка, що складається з ряду вертикальних плоских лопатей прямокутної форми, закрплених до валу, що обертається електродвигуном. При роботі такої мішалки утворюється лійка і може всмоктуватися повітря з поверхні змішуваної маси.

5. НАВЧАЛЬНИЙ ПРИКЛАД ДО ПРАВИЛ ВИКОРИСТАННЯ КОМПОНЕНТИ MAINMENU ДЛЯ СТВОРЕННЯ МЕНЮ КОМАНД У ПРИКЛАДНІЙ C++ ПРОГРАМІ

Крок 1. Створюємо новий проект для розробки і програмування C++ програми для навчального прикладу до модульної контрольної роботи і для цього необхідно виконувати наступні дії:

- в основному вікні C++ Builder виконуємо команду **File/New/ Application** для створення чистої основної форми **Form1**;

- переходимо у вікно інспектора об'єктів і у полі **Caption** набираємо назву «Мнемосхема технологічного процесу виробництва хлорметанів» для заголовку вікна прикладної C++ програми;

- виконуємо команду **File/Save Project As** і пропонуємо для файлу назву **Project1.bpr** залишаємо, або задаємо назву проекту **P-xlormetans.bpr** та пропонуємо назву файлу **Unit1.cpp** залишаємо для програмного модуля першої відкритої форми **Form1**.

Крок 2. Для побудови основного меню команд програми C++ на полі **Form1** встановлюємо компоненту **MainMenu** з сторінки **Standard** бібліотеки VCL і для цього необхідно виконати такі дії:

- у бібліотеці вибираємо маніпулятором “мишка” компоненту **MainMenu**;

- робимо щелчок клавішею мишки на розмітчастій сітці вікна **Form1**.

Крок 3. Блок-схема алгоритму з виконання команд у меню C++ програми до модульної контрольної роботи показано на рис. 5-1. Відповідно до алгоритму рис. 5-1 та відповідно до структури меню команд на рис. 4-2 задаємо у компоненту **MainMenu** пункти команд такими діями:

- натискаємо лівою кнопкою мишки на формі по значку **MainMenu** та обираємо **Menu Designer**;

- обираємо курсором з двох рамок верхню та у вікні інспектора об'єктів в полі **Caption** вводимо назву “Технологія”, а у полі **name** вводимо назву “**Technology**”.

Далі створюємо пункт меню “Апарати”. Для цього натискаємо правою кнопкою мишки на пункті “Апарати” та обираємо **Create Submenu**, або ж просто натискаємо **Ctrl+Right** на клавіатурі (**Right** — стрілка вправо). Далі повторюємо аналогічні дії, як і при створенні меню “Технологія”. У вікні інспектора об'єктів в полі **Caption** вводимо назву “Апарати”, а у полі **name** вводимо назву “**Devices**”.

Створюємо далі основні пункти для меню “Інформація” та “Вихід” аналогічними діями, як і у пунктах меню “Технологія” та “Апарати”. У полі **name** для пункту меню “Інформація” вводимо назву “**Info**”, а для пункту меню “Вихід” вводимо назву “**Exit**”.

Крок 3.1 Створюємо підменю “Схема”, “Опис схеми” і “Продукція” для меню “Технологія”. Для цього заповнюємо комірки нижче назви меню “Технологія”. У полі **name** для пункту меню “Схема” вводимо назву “**Scheme**”, а для пункту меню “Опис схеми” вводимо назву “**Description**” і для пункту меню “Продукція” вводимо назву “**Products**”.

Крок 3.2 Створюємо підменю “Хлоратор”, “Абсорбер” і “Розріджувач лугу” для меню “Апарати” аналогічними діями, відповідно до дій для кроку 3.1.

У полі **name** для пункту меню “Хлоратор” вводим назву “**Device1**”, для пункту меню “Абсорбер” вводим назву “**Device2**” і для пункту меню “Розріджувач лугу” вводим назву “**Device3**”. Для кожного з цих підменю створюємо по три власних підкоманди за допомогою клавіш клавіатури **Ctrl+Right (Right** – стрілка вправо): “Конструкція”, “Опис процесу”, “Параметри”.

Для першого апарату в полі **name** для підкоманди “Конструкція” вводим назву “**Design1**”. Для підкоманди “Опис процесу” вводим назву “**Process1**” і для підкоманди “Параметри” вводим назву “**Characteristic1**”.

Для другого апарату в полі **name** для підкоманди “Конструкція” вводим назву “**Design2**”. Для підкоманди “Опис процесу” вводим назву “**Process2**” і для підкоманди “Параметри” вводим назву “**Characteristic2**”.

Для третього апарату в полі **name** для підкоманди “Конструкція” вводим назву “**Design3**”. Для підкоманди “Опис процесу” вводим назву “**Process3**” і для підкоманди “Параметри” вводим назву “**Characteristic3**”.

Крок 3.3 Створюємо підкоманди “Про програму” та “Література” для меню команд “Інформація” аналогічними діями, як і при виконанні кроку 3.2. У полі **name** для підменю “Про програму” вводим назву “**About**”, а для підменю “Література” вводим назву “**Literature**”. Для зручності роботи з інспектором об’єктів можна скористатись наступними таблицями:

Таблиця 5-1.

<i>Main Menu</i>							
Технологія		Апарати		Інформація		Вихід	
Caption	Name	Caption	Name	Caption	Name	Caption	Name
Технологія	Technology	Апарати	Devices	Інформація	Info	Вихід	Exit

Таблиця 5-2.

<i>Технологія</i>					
Схема		Опис схеми		Продукція	
Caption	Name	Caption	Name	Caption	Name
Схема	Scheme	Опис схеми	Description	Продукція	Products

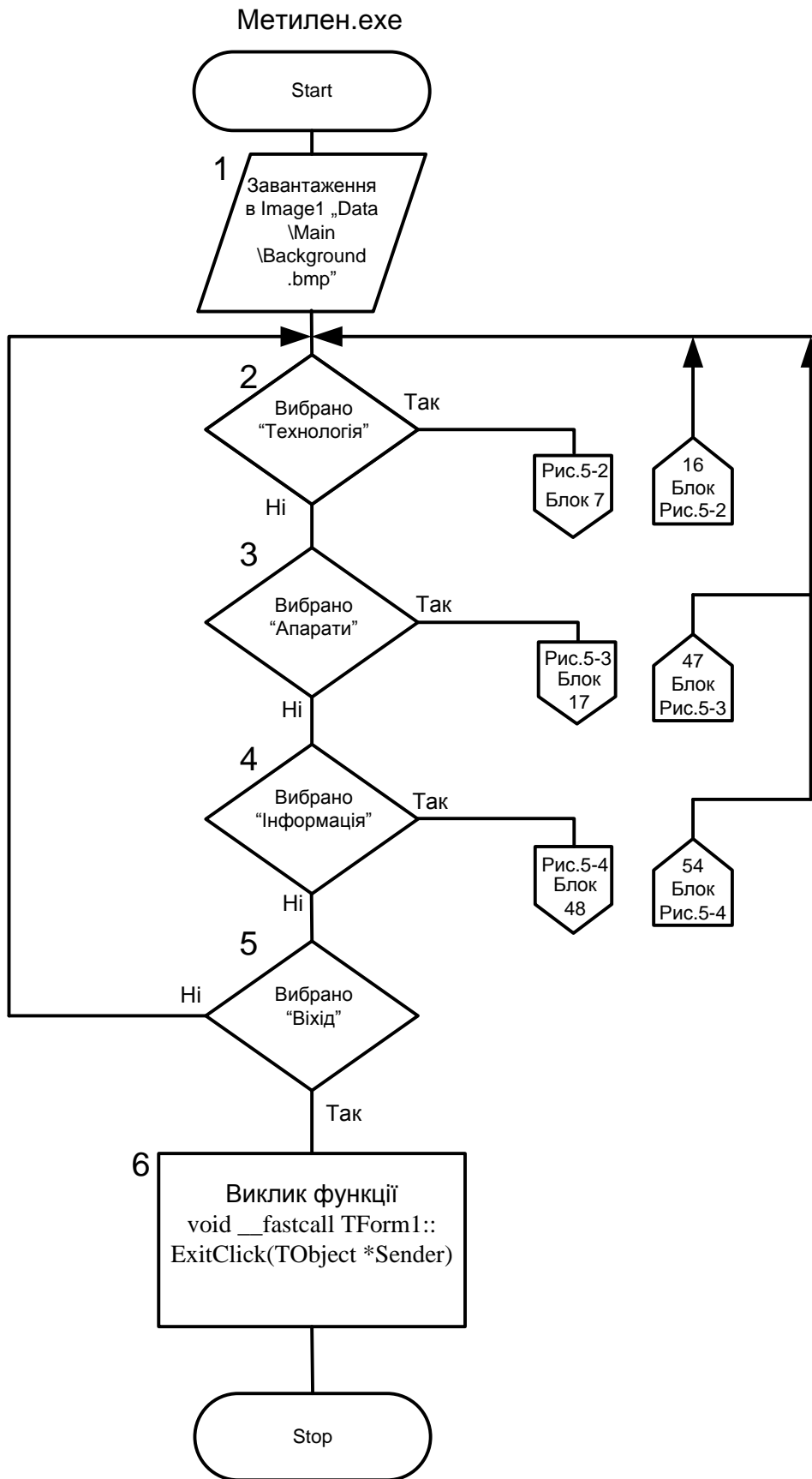


Рис. 5-1. Блок-схема алгоритму з виконання команд основного меню C++ програми до прикладу з модульної контрольної роботи.

Таблиця 5-3.

<i>Апарати</i>					
Хлоратор		Абсорбер		Розріджувач лугу	
Caption	Name	Caption	Name	Caption	Name
Хлоратор	Device1	Абсорбер	Device2	Розріджувач лугу	Device3

Таблиця 5-4.

<i>Хлоратор</i>					
Конструкція		Опис процесу		Параметри	
Caption	Name	Caption	Name	Caption	Name
Конструкція	Design1	Опис процесу	Process1	Параметри	Characteristic1

Таблиця 5-5.

<i>Абсорбер</i>					
Конструкція		Опис процесу		Параметри	
Caption	Name	Caption	Name	Caption	Name
Конструкція	Design2	Опис процесу	Process2	Параметри	Characteristic2

Таблиця 5-6.

<i>Розріджувач лугу</i>					
Конструкція		Опис процесу		Параметри	
Caption	Name	Caption	Name	Caption	Name
Конструкція	Design3	Опис процесу	Process3	Параметри	Characteristic3

<i>І н ф о р м а ц і я</i>			
Про програму		Література	
Caption	Name	Caption	Name
Про програму	About	Література	Literature

Крок 4. Встановлюємо на **Form1** компоненту **ImageList** зі сторінки **Win32** у бібліотеці компонентів **C++ Builder**.

Крок 5. Завантажуємо у компоненту **ImageList1** іконки (значки) для пунктів меню команд.

Натискаємо лівою кнопкою миші на формі по значку **ImageList1** та обираємо **Imagelist Editor**. За допомогою кнопки **Add** вказуємо шлях доступу до іконок та відкриваємо їх кнопкою “**Открыть**”.

Крок 6. Приєднуємо компоненту **ImageList1** до основного меню команд програми, завданих у **MainMenu**.

Натискаємо лівою кнопкою миші на формі по значку **MainMenu** та у вікні інспектора об’єктів обираємо закладку сторінки **Events**. В полі **Images** вводимо **ImageList1**.

Крок 7. Приєднуємо іконки до пунктів підменю. Натискаємо лівою кнопкою миші на формі по значку **ImageList1** та обираємо **Imagelist Editor**. Запам’ятовуємо порядковий номер іконки для певного пункту підменю і номер іконки, вказаний під самою іконкою. Нумерація іконок починається з нуля. Далі натискаємо лівою кнопкою миші на формі по значку **MainMenu** та обираємо **Menu Designer**.

Обираємо пункт меню, для якого ми запам’ятовували порядковий номер іконки та у вікні інспектора об’єктів у полі **ImageIndex** вводимо номер цієї самої іконки. Повторюємо аналогічні дії для усіх пунктів та підпунктів меню команд **C++** програми навчального прикладу до модульної контрольної роботи.

Крок 8. Встановлюємо на **Form1** компоненту **Image** з сторінки **Additional** у бібліотеці компонентів **C++ Builder** та маркерною рамкою визначаємо бажаний розмір поля для розміщення рисунка мнемосхеми технологічного процесу виробництва хлорметанів (рис. 4-4), як основного фону прикладної **C++** програми. Якщо потрібно розмір форми можна також змінити.

Крок 9. Встановлюємо на **Form1** компоненту **Label** з сторінки **Additional** у бібліотеці компонентів та розміщуємо у лівому нижньому куту форми **Form1**.

Крок 10. Натискаємо лівою кнопкою мишки на **Form1** у полі **Image1** та у вікні інспектора об'єктів в полі **Picture** натискаємо на значок з трьома крапками. Далі кнопкою **Load** відкриваємо список каталогів і вказуємо шлях доступу до рисунка (формат ***.bmp**) мнемосхеми технологічного процесу та натискаємо кнопку **“Открыть”** і у вікні **Picture Editor** натискаємо кнопку **“ОК”**. Правою кнопкою мишки натискаємо на полі **Image1** та виконуємо команду **Control/Send to back**.

Крок 11. На рисунку, що завантажился у **Image1**, за допомогою компоненти **Image** з сторінки **Additional** встановлюємо окремі поля (**Image2..Image18**) зверху навколо кожного технологічного апарату для визначення у **C++** програмі відповідної підказки до назви технологічного апарату, показаного на мнемосхемі хімічного виробництва хлорметанів.

Крок 12. Задаємо відображення у **Label1** підказок з назв технологічних апаратів при переміщенні курсора маніпулятора **“мишка”** по мнемосхемі технологічного процесу виробництва хлорметанів, яку завантажено у **Image1**.

Натискаємо на полі **Image1** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у функцію **Image1MouseMove** додаємо виконуємий оператор:

```
//-----  
void __fastcall TForm1::Image1MouseMove(TObject *Sender, TShiftState Shift,  
    int X, int Y)  
{  
Label1->Caption=("");
```

```
}
```

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

Натискаємо на полі **Image2** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у функцію **Image2MouseMove** додаємо виконуємий оператор:

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

```
void __fastcall TForm1::Image2MouseMove(TObject *Sender, TShiftState Shift,  
    int X, int Y)
```

```
{
```

```
Label1->Caption= ("3 - Змішувач");
```

```
}
```

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

Натискаємо на полі **Image3** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у функцію **Image3MouseMove** додаємо виконуємий оператор:

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

```
void __fastcall TForm1::Image3MouseMove(TObject *Sender, TShiftState Shift,  
    int X, int Y)
```

```
{
```

```
Label1->Caption= ("4 - Хлоратор");
```

```
}
```

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

Натискаємо на полі **Image4** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у функцію **Image4MouseMove** додаємо виконуємий оператор:

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

```

void __fastcall TForm1::Image4MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("1 - Змішувач");
}
//-----

```

Натискаємо на полі **Image5** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у функцію **Image5MouseMove** додаємо виконуємий оператор:

```

//-----
void __fastcall TForm1::Image5MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("2 - Підігрівник");
}
//-----

```

Натискаємо на полі **Image6** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у функцію **Image6MouseMove** додаємо виконуємий оператор:

```

//-----
void __fastcall TForm1::Image6MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("5 - Сажовловлювач");
}
//-----

```


Натискаємо на полі **Image7** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у функцію **Image7MouseMove** додаємо виконуємий оператор:

```
//-----  
void __fastcall TForm1::Image7MouseMove(TObject *Sender, TShiftState Shift,  
    int X, int Y)  
{  
Label1->Caption= ("16 - Сепаратор");  
}  
//-----
```

Натискаємо на полі **Image8** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у функцію **Image8MouseMove** додаємо виконуємий оператор:

```
//-----  
void __fastcall TForm1::Image8MouseMove(TObject *Sender, TShiftState Shift,  
    int X, int Y)  
{  
Label1->Caption= ("10 – Графітовий холодильник");  
}  
//-----
```

Натискаємо на полі **Image9** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у функцію **Image9MouseMove** додаємо виконуємий оператор:

```
//-----  
void __fastcall TForm1::Image9MouseMove(TObject *Sender, TShiftState Shift,  
    int X, int Y)
```

```
{  
Label1->Caption= ("7 – Абсорбер");  
}  
//-----
```

Натискаємо на полі **Image10** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у редактор коду, де у функцію **Image10MouseMove** додаємо виконуємий оператор:

```
//-----  
void __fastcall TForm1::Image10MouseMove(TObject *Sender, TShiftState Shift,  
    int X, int Y)  
{  
Label1->Caption= ("8 – Абсорбер");  
}  
//-----
```

Натискаємо на полі **Image11** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у редактор коду, де у функцію **Image11MouseMove** додаємо виконуємий оператор:

```
//-----  
void __fastcall TForm1::Image11MouseMove(TObject *Sender, TShiftState Shift,  
    int X, int Y)  
{  
Label1->Caption= ("9 – Розріджувач луку");  
}  
//-----
```

Натискаємо на полі **Image12** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у редактор коду, де у функцію **Image12MouseMove** додаємо виконуємий оператор:

```
//-----
void __fastcall TForm1::Image12MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("11 – Збірник концентрованої соляної кислоти");
}
//-----
```

Натискаємо на полі **Image13** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у редактор коду, де у функцію **Image13MouseMove** додаємо виконуємий оператор:

```
//-----
void __fastcall TForm1::Image13MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("12 – Компресор");
}
//-----
```

Натискаємо на полі **Image14** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у редактор коду, де у функцію **Image14MouseMove** додаємо виконуємий оператор:

```
//-----
void __fastcall TForm1::Image14MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("15 – Фільтр");
}
//-----
```

Натискаємо на полі **Image15** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у редактор коду, де у функцію **Image15MouseMove** додаємо виконуємий оператор:

```
//-----  
void __fastcall TForm1::Image15MouseMove(TObject *Sender, TShiftState Shift,  
    int X, int Y)  
{  
Label1->Caption= ("14 – Конденсатор");  
}  
//-----
```

Натискаємо на полі **Image16** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у редактор коду, де у функцію **Image16MouseMove** додаємо виконуємий оператор:

```
//-----  
void __fastcall TForm1::Image16MouseMove(TObject *Sender, TShiftState Shift,  
    int X, int Y)  
{  
Label1->Caption= ("13 – Осушувачі");  
}  
//-----
```

Натискаємо на полі **Image17** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у редактор коду, де у функцію **Image17MouseMove** додаємо виконуємий оператор:

```
//-----  
void __fastcall TForm1::Image17MouseMove(TObject *Sender, TShiftState Shift,  
    int X, int Y)
```

```

{
Label1->Caption= ("17 – Розріджувач лугу");
}
//-----

```

Натискаємо на полі **Image18** і у вікні інспектора об'єктів обираємо вкладку **Events** та в полі **OnMouseMove** подвійним натисканням кнопки мишки переходимо у редактор коду, де у функцію **Image18MouseMove** додаємо виконуємий оператор:

```

//-----
void __fastcall TForm1::Image18MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("6 – Повітряний холодильник");
}
//-----

```

Крок 13. Натискаємо лівою кнопкою мишки у вікні форми на полі **Image1** і у вікні інспектора об'єктів у полі **Picture** натискаємо на значок з трьома крапками. Кнопкою **Clear** видаляємо рисунок мнемосхеми технологічного процесу хімічного виробництва та натискаємо “**OK**”.

Крок 14. Задаємо завантаження рисунка мнемосхеми в **Image1** для фону основного вікна при роботі прикладної **C++** програми.

На вільному просторі форми (розмітчній сітці) натискаємо лівою кнопкою мишки. У вікні інспектора об'єктів обираємо вкладку **Events** і подвійним натисканням мишки у списку на назві **OnCreate** переходимо у вікно редактора коду прикладної програми.

У шаблон функції **FormCreate** необхідно додати наступні тексти операторів, яки забезпечат завантаження рисунка мнемосхеми, як фону в компоненту **Image1**, та на формі **Form1** поля від **Image2** до **Image18** стануть невидимими:

```

//-----

```

```

void __fastcall TForm1::FormCreate(TObject *Sender)
{
Image1->Picture->LoadFromFile("Data/Main/Background.bmp");
Image2->Visible = false;
Image3->Visible = false;
Image4->Visible = false;
Image5->Visible = false;
Image6->Visible = false;
Image7->Visible = false;
Image8->Visible = false;
Image9->Visible = false;
Image10->Visible = false;
Image11->Visible = false;
Image12->Visible = false;
Image13->Visible = false;
Image14->Visible = false;
Image15->Visible = false;
Image16->Visible = false;
Image17->Visible = false;
Image18->Visible = false;
}
//-----

```

Крок 15. Переходимо з вікна редактора коду на форму **Form1** і натискаємо кнопку мишки на полі **Image3**.

У вікні інспектора об'єктів в полі **Cursor** обираємо для курсору назву **crHandPoint**. Далі обираємо вкладку **Events** і подвійним натисканням мишки на **OnClick** переходимо у вікно редактора коду, де у шаблон функції **Image3Click** додаємо текст з наступними строками:

```

//-----
void __fastcall TForm1::Image3Click(TObject *Sender)

```

```

{
Form3->Caption="Конструкція хлоратора";
Form3->Image1->Picture->LoadFromFile("Data/Device1/Device.bmp");
Form3->ShowModal();
}
//-----

```

Крок 16. Переходимо на форму **Form1** і натискаємо кнопку мишки на полі **Image9**.

У вікні інспектора об'єктів в полі **Cursor** обираємо для курсора назву **crHandPoint**. Далі обираємо вкладку **Events** і в полі **OnClick** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у шаблон функції **Image9Click** додаємо текст з наступними строками:

```

//-----
void __fastcall TForm1::Image9Click(TObject *Sender)
{
Form3->Caption="Конструкція абсорбера";
Form3->Image1->Picture->LoadFromFile("Data/Device2/Device.bmp");
Form3->ShowModal();
}
//-----

```

Крок 17. Переходимо на форму **Form1** і натискаємо кнопку мишки на полі **Image10**.

У вікні інспектора об'єктів в полі **Cursor** обираємо для курсора назву **crHandPoint**. Далі обираємо вкладку **Events** і в полі **OnClick** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у шаблон функції **Image10Click** додаємо текст з наступними строками:

```

//-----
void __fastcall TForm1::Image10Click(TObject *Sender)
{
Form3->Caption="Конструкція абсорбера";

```

```
Form3->Image1->Picture->LoadFromFile("Data/Device2/Device.bmp");
Form3->ShowModal();
}
//-----
```

Крок 18. Переходимо на форму **Form1** і натискаємо кнопку мишки на полі **Image17**.

У вікні інспектора об'єктів в полі **Cursor** обираємо для курсора назву **crHandPoint**. Далі обираємо вкладку **Events** і в полі **OnClick** подвійним натисканням кнопки мишки переходимо у вікно редактора коду, де у шаблон функції **Image11Click** додаємо текст з наступними строками:

```
//-----
void __fastcall TForm1::Image17Click(TObject *Sender)
{
Form3->Caption="Конструкція розріджувача лугу";
Form3->Image1->Picture->LoadFromFile("Data/Device3/Device.bmp");
Form3->ShowModal();
}
//-----
```

У функціях **Image3Click**, **Image9Click**, **Image10Click** і **Image11Click** оператори додані для відкриття вікна форми **Form3** з відповідним рисунком схеми технологічного апарату, коли на мнемосхемі курсор мишки у вигляді значка **crHandPoint** буде встановленим на апарат “Хлоратор”, або на апарат “Абсорбер”, або на – “Розріджувач лугу”, яки позначени додатково на **Form1** відповідними встановленими полями **Image** (дивись крок 11 з пункту 5).

5.1 НАВЧАЛЬНИЙ ПРИКЛАД З ПРОГРАМУВАННЯ ФУНКЦІЙ ДЛЯ ВИКОНАННЯ КОМАНД У МЕНЮ «ТЕХНОЛОГІЯ»

Відповідно до рис. 4-2 меню “Технологія” у програмі C++ складається з трьох підменю: “Схема”, “Опис схеми” і “Продукція”. У компоненту **MainMenu**

потрібно додати виконавчий код для кожної команди з меню команд прикладної програми C++. Спочатку створимо дві додаткові форми, які нам знадобляться для відображення інформації, яка буде відповідати цим командам.

Крок 1. Виконуємо команду **File/New/Form** для створення нової чистої форми **Form2**.

Встановлюємо на **Form2** компоненту **RichEdit** з сторінки **Win32** у бібліотеці компонентів C++ Builder. Натискаємо мишкою на значок новоствореної компоненти **RichEdit1** і у вікні інспектора об'єктів в полі **Align** обираємо **alClient**. Таким чином вікно компоненти **RichEdit1** при відкритті буде встановлюватися відповідно до розмірів форми **Form2**.

Крок 2. Виконуємо команду **File/New/Form** для створення нової чистої форми **Form3**.

У вікні дерева об'єктів обираємо **Form3** і далі у полі **AutoSize** вікна інспектора об'єктів обираємо режим відображення **true**.

Зі сторінки **Additional** у бібліотеці **VCL** встановлюємо на **Form3** компоненту **Image** для показу схеми конструкції технологічного апарату. Натискаємо кнопку мишки на новоствореній компоненті **Image1** та у полі **Width** вікна інспектора об'єктів вводимо розмір 385, а у полі **Height** вводимо розмір 497, які відповідають розмірам рисунка з конструкції технологічного апарату. Блок-схему алгоритму з виконання команд у меню "Технологія" показано на рис. 5-2.

Налаштування функції до підменю "Схема" з меню "Технологія":

Крок 1. Виконуємо далі дії відповідно до алгоритмів наведених на рис. 5-1 та рис. 5-2. У верхній частині **Form1** натискаємо на значку компоненти **MainMenu** і потім на назві "Технологія" та у підменю обираємо пункт "Схема". У вікні редактора коду, що відкриється для функції **SchemeClick** додаємо наступні строки:

```
//-----  
void __fastcall TForm1::SchemeClick(TObject *Sender)  
{  
Image1->Picture->LoadFromFile("Data/Main/Scheme.bmp");  
Image2->Visible = true;
```

```

Image3->Visible = true;
Image4->Visible = true;
Image5->Visible = true;
Image6->Visible = true;
Image7->Visible = true;
Image8->Visible = true;
Image9->Visible = true;
Image10->Visible = true;
Image11->Visible = true;
Image12->Visible = true;
Image13->Visible = true;
Image14->Visible = true;
Image15->Visible = true;
Image16->Visible = true;
Image17->Visible = true;
Image18->Visible = true;
Label1->Visible = true;
}
//-----

```

Сформована вище функція, яка буде виконуватись при натисканні мишки на команді “Схема” у меню “Технологія”. Ця функція буде завантажувати у **Image1** рисунок мнемосхеми (рис. 4 - 4) і зробить компоненти від **Image2** до **Image18** та **Label1** невидимими у вікні **Form1**, щоб при наведенні курсора маніпулятора “мишка” на зображення апарату з’являлась в **Label1** відповідна підказка з назвою технологічного апарату, обраного мишкою на мнемосхемі технологічного процесу виробництва хлорметанів.

Налаштування функції до підменю “Опис схеми” з меню “Технологія”:

Крок 1. У верхній частині **Form1** натискаємо на значку компоненти **MainMenu** і потім на назві “Технологія” та у підменю обираємо команду “Опис схеми”. У вікні редактора коду, що відкриється додаємо у шаблон функції **DescriptionClick** наступні строки тексту:

```

//-----
void __fastcall TForm1::DescriptionClick(TObject *Sender)
{
Form2->RichEdit1->Lines->LoadFromFile("Data/Main/Scheme.rtf");
Form2->Caption="Опис процесу виробництва хлорметанів";

```

```
Form2->ShowModal();
```

```
}
```

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

Таким чином буде відкриватись вікно форми **Form2** для відображення тексту з файлу **Scheme.rtf**, у якому буде зберігатися відповідний опис процесу виробництва хлорметанів.

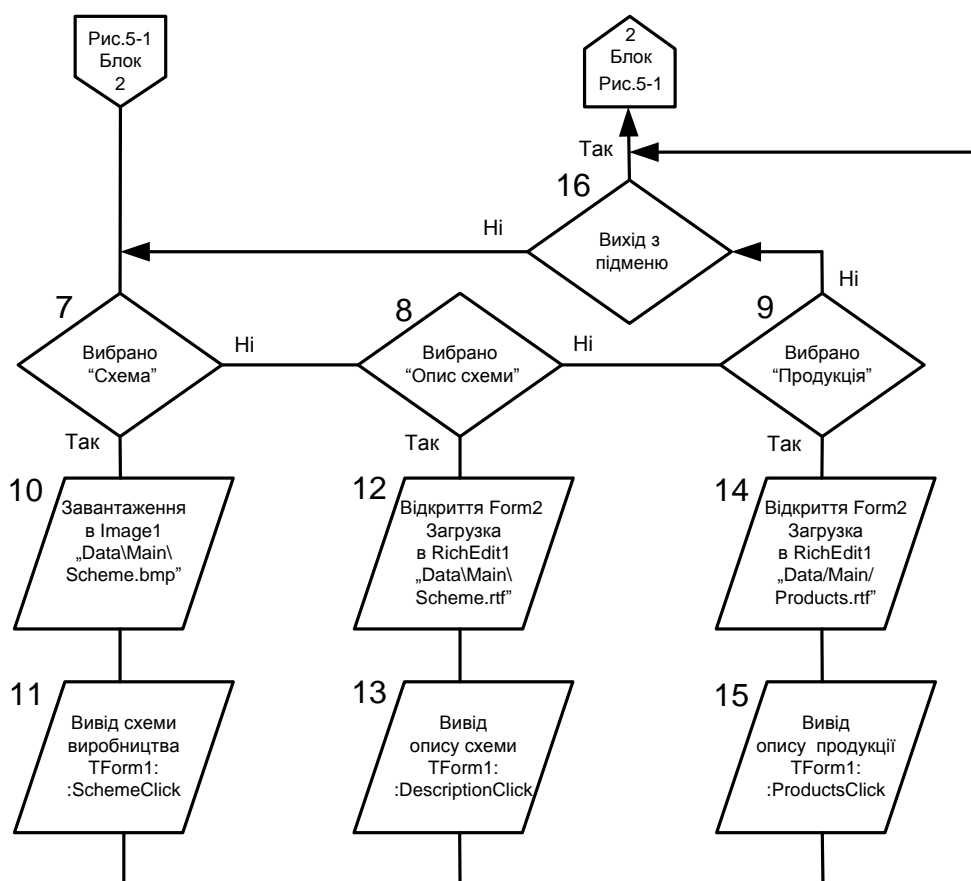


Рис. 5-2. Блок-схема алгоритму з виконання команд у меню “Технологія”.

Налаштування функції до підменю “Продукція” з меню “Технологія”:

Крок 1. У верхній частині **Form1** натискаємо кнопкою мишки на значку **MainMenu**, потім на назві “Технологія” і у підменю обираємо команду “Продукція”. У вікні редактора коду, що відкриється додаємо у шаблон функції **ProductsClick** наступні строки тексту:

```
//-----
void __fastcall TForm1::ProductsClick(TObject *Sender)
{
Form2->RichEdit1->Lines->LoadFromFile("Data/Main/Products.rtf");
Form2->Caption="Продукція";
Form2->ShowModal();
}
//-----
```

Таким чином дана функція буде у вікні форми **Form2** відображати з файлу **Products.rtf** текст з описом властивостей продукції з суміши хлорметанів. Вище виконані налаштування функцій з виконання команд у підменю і підкомандах з меню “Технологія” відповідно до блок-схеми алгоритму рис. 5-2.

5.2 НАВЧАЛЬНИЙ ПРИКЛАД З ПРОГРАМУВАННЯ ФУНКЦІЙ ДЛЯ ВИКОНАННЯ КОМАНД У МЕНЮ «АПАРАТИ»

У розглядаємому навчальному прикладі, відповідно до рис. 4-2 меню команд “Апарати” буде складатися з трьох підменю: “Хлоратор”, “Абсорбер” і “Розріджувач луку”. Кожне з цих підменю має по три підкоманди: “Конструкція”, “Опис процесу” і “Параметри”. Далі потрібно у компоненту **MainMenu** додати виконавчий код для цих команд, тобто необхідно налаштувати 9 команд, щоб відповідно до кожного з 3-х підпунктів і до 3-х назв технологічних апаратів у вікні форми виводилась наступна інформація: з схеми конструкції апарату; текст опису процесу в технологічному апараті; графіки зміни у часі технологічних параметрів.

Спочатку створюємо три додаткові форми, які нам знадобляться для відображення інформації до команд підменю з меню “Апарати”.

Відкриття форми Form4.

Крок 1. Виконуємо команду **File/New/Form** для створення нової чистої форми **Form4**.

Встановлюємо на **Form4** компоненту **Chart** з сторінки **Additional** у бібліотеці компонентів *C++ Builder*.

Крок 2. Натискаємо правою кнопкою мишки на новостворену компоненту **Chart1** та обираємо **Edit Chart**. Переходимо на вкладку **Chart/Series**, натискаємо **Add** і обираємо **Fast Line** та натискаємо “**OK**”. Далі переходимо на вкладку **Chart/Titles** та у полі вводу набираємо назву технологічного параметру “**Температура**”. Переходимо на вкладку **Chart/Legend** та знімаємо галочку з пункту **Visible**. Натискаємо **Close** для виходу з меню **Edit Chart**.

Крок 3. Встановлюємо на **Form4** другу компоненту **Chart** з сторінки **Additional** у бібліотеці **VCL** для виконання налаштувань до графіку другого технологічного параметру.

Крок 4. Натискаємо правою кнопкою мишки на новостворену компоненту **Chart2** та обираємо **Edit Chart**. Переходимо на вкладку **Chart/Series**, натискаємо **Add**, обираємо **Fast Line** та натискаємо “**OK**”. Переходимо на вкладку **Chart/Titles** та у полі вводу набираємо назву “**Витрата сировини**”. Переходимо на вкладку **Chart/Legend** та знімаємо галочку з пункту **Visible**. Натискаємо **Close** для виходу з меню **Edit Chart**.

Крок 5. У вікні дерева об’єктів обираємо **Form4** і потім у вікні інспектора об’єктів обираємо вкладку **Events** та подвійним натисканням мишки у полі **OnCreate** переходимо у вікно редактора коду. У вікні редактора коду, що відкриється з шаблоном функції **FormCreate** додаємо наступні строки тексту:

```
//-----  
void __fastcall TForm4::FormCreate(TObject *Sender)  
{  
for (int i=0; i<=10; i++)  
{  
Series1->AddXY(i+cos(i),13*i*i,"",clRed);  
Series2->AddXY(12*i*i+25, i*i+15*i,"",clRed);  
}  
}
```

//-----

Відкриття форми Form5.

Крок 1. Виконуємо команду **File/New/Form** для створення нової чистої форми **Form5**. Встановлюємо на **Form5** компоненту **Chart** з сторінки **Additional** у бібліотеці **VCL**.

Крок 2. Натискаємо правою кнопкою мишки на новостворену компоненту **Chart1** та обираємо **Edit Chart**. Переходимо на вкладку **Chart/Series**, натискаємо **Add** і обираємо **Fast Line** та натискаємо **“OK”**. Переходимо на вкладку **Chart/Titles** та у полі вводу набираємо назву **“Температура”**. Переходимо на вкладку **Chart/Legend** та знімаємо галочку з пункту **Visible**. Натискаємо **Close** для виходу з меню **Edit Chart**.

Крок 3. Встановлюємо на **Form5** другу компоненту **Chart** з сторінки **Additional** у бібліотеці компонентів **VCL**.

Крок 4. Натискаємо правою кнопкою мишки на новостворену компоненту **Chart2** та обираємо **Edit Chart**. Переходимо на вкладку **Chart/Series**, натискаємо **Add**, обираємо **Fast Line** та натискаємо **“OK”**. Переходимо на вкладку **Chart/Titles** та у полі вводу набираємо **“Витрата сировини”**. Переходимо на вкладку **Chart/Legend** та знімаємо галочку з пункту **Visible**. Натискаємо **Close** для виходу з меню **Edit Chart**.

Крок 5. У вікні дерева об'єктів обираємо **Form5** і потім у інспекторі об'єктів обираємо вкладку **Events** та подвійним натисканням мишки у полі **OnCreate** переходимо у вікно редактора коду. У вікні редактора коду, що відкриється для функції **FormCreate** додаємо наступні строки тексту:

//-----

```
void __fastcall TForm5::FormCreate(TObject *Sender)
{
for (int i=0; i<=10; i++)
{
Series1->AddXY(i+sin(i),13*i*i,"",clRed);
```

```
Series2->AddXY(14*i+25*i*sin(i), cos(i)*i*i*i+15*i,"",clRed);
}
}
//-----
```

Відкриття форми Form6.

Крок 1. Виконуємо команду **File/New/Form** для створення нової чистої форми **Form6**. Встановлюємо на **Form6** компоненту **Chart** з сторінки **Additional** у бібліотеці **VCL**.

Крок 2. Натискаємо правою кнопкою мишки на новостворену компоненту **Chart1** та обираємо **Edit Chart**. Переходимо на вкладку **Chart/Series**, натискаємо **Add** і обираємо **Fast Line** та натискаємо “**OK**”. Переходимо на вкладку **Chart/Titles** та у полі вводу набираємо назву “**Температура**”. Далі переходимо на вкладку **Chart/Legend** та знімаємо галочку з пункту **Visible**. Натискаємо **Close** для виходу з меню **Edit Chart**.

Крок 3. Встановлюємо на **Form6** другу компоненту **Chart** з сторінки **Additional** у бібліотеці **VCL**.

Крок 4. Натискаємо правою кнопкою мишки на новостворену компоненту **Chart2** та обираємо **Edit Chart**. Переходимо на вкладку **Chart/Series**, натискаємо **Add** і обираємо **Fast Line** та натискаємо “**OK**”. Переходимо на вкладку **Chart/Titles** та у полі вводу набираємо “**Витрата сировини**”. Переходимо на вкладку **Chart/Legend** та знімаємо галочку з пункту **Visible**. Натискаємо **Close** для виходу з меню **Edit Chart**.

Крок 5. У вікні дерева об’єктів обираємо **Form6** і потім у вікні інспектора об’єктів обираємо вкладку **Events** та подвійним натисканням мишки у полі **OnCreate** переходимо у вікно редактора коду. У редакторі коду, що відкриється з шаблоном для функції **FormCreate** додаємо наступні строки тексту:

```
//-----
void __fastcall TForm6::FormCreate(TObject *Sender)
{
```

```

for (int i=0; i<=10; i++)
{
Series1->AddXY(i+tan(i),13*i*i,"",clRed);
Series2->AddXY(13*i+25*tan(i), exp(i)*i*i+15*i,"",clRed);
}
}
//-----

```

Далі можна перейти до заповнення виконавчим кодом команд у компоненті **MainMenu** до відповідних 9 підпунктів.

Налаштування виконавчого коду до команд підменю “Хлоратор”.

Крок 1. У верхній частині **Form1** натискаємо на компоненті **MainMenu** і на пункті “Апарати” та у підменю обираємо команду “Хлоратор” і потім обираємо підпункт “Конструкція”. У вікні редактора коду, що відкриється додаємо у шаблон функції **Design1Click** наступний текст:

```

//-----
void __fastcall TForm1::Design1Click(TObject *Sender)
{
Form3->Caption="Конструкція хлоратора";
Form3->Image1->Picture->LoadFromFile("Data/Device1/Device.bmp");
Form3->ShowModal();
}
//-----

```

Крок 2. У верхній частині **Form1** натискаємо кнопкой мишки на **MainMenu** і на назві “Апарати” та у підменю обираємо команду “Хлоратор” і потім обираємо підпункт “Опис процесу”. У вікні редактора коду, що відкриється додаємо у шаблон функції **Process1Click** наступний текст:

```

//-----
void __fastcall TForm1:: Process1Click(TObject *Sender)
{

```



```

Form2->RichEdit1->Lines->LoadFromFile("Data/Device1/Process.rtf");
Form2->Caption="Опис процесу в хлораторі";
Form2->ShowModal();
}
//-----

```

Крок 3. У верхній частині **Form1** натискаємо на значку **MainMenu** і на назві “Апарати” та у підменю обираємо команду “Хлоратор” і обираємо підпункт “Параметри”. У вікні редактора коду, що відкриється додаємо у шаблон функції **Characteristic1Click** наступний текст:

```

//-----
void __fastcall TForm1::Characteristic1Click(TObject *Sender)
{
Form4->Caption="Параметри хлоратора";
Form4->ShowModal();
}
//-----

```

Налаштування виконавчого коду до команд підменю “Абсорбер”.

Крок 1. У верхній частині **Form1** натискаємо кнопкою мишки на **MainMenu** і на назві “Апарати” та у підменю обираємо команду “Абсорбер” і підпункт “Конструкція”. У вікні редактора коду, яке відкриється додаємо у шаблон функції **Design2Click** наступний текст:

```

//-----
void __fastcall TForm1::Design2Click(TObject *Sender)
{
Form3->Caption="Конструкція абсорбера";
Form3->Image1->Picture->LoadFromFile("Data/Device2/Device.bmp");
Form3->ShowModal();
}
//-----

```

Крок 2. У верхній частині **Form1** натискаємо на **MainMenu** і вибираємо назву “Апарати” та у підменю обираємо команду “Абсорбер” і підпункт “Опис процесу”. У вікні редакторі коду, що відкриється додаємо у шаблон функції **Process2Click** наступний текст:

```
//-----  
void __fastcall TForm1::Process2Click(TObject *Sender)  
{  
Form2->RichEdit1->Lines->LoadFromFile("Data/Device2/Process.rtf");  
Form2->Caption="Опис процесу в абсорбери";  
Form2->ShowModal();  
}  
//-----
```

Крок 3. У верхній частині **Form1** натискаємо кнопкою мишки на **MainMenu** і на назві “Апарати” та у підменю обираємо команду “Абсорбер” і підпункт “Параметри”. У вікні редактора коду, що відкриється додаємо у шаблон функції **Characteristic2Click** наступний текст:

```
//-----  
void __fastcall TForm1::Characteristic2Click(TObject *Sender)  
{  
Form5->Caption="Параметри абсорбера";  
Form5->ShowModal();  
}  
//-----
```

Налаштування виконавчого коду до команд
підменю “Розріджувач луку”.

Крок 1. У верхній частині **Form1** натискаємо кнопкою мишки на **MainMenu** і на назві “Апарати” та у підменю обираємо команду “Розріджувач луку” і підпункт “Конструкція”. У вікні редактора коду, що відкриється додаємо у шаблон функції **Design3Click** наступний текст:

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

```

void __fastcall TForm1::Design3Click(TObject *Sender)
{
Form3->Caption="Конструкція розріджувача лугу";
Form3->Image1->Picture->LoadFromFile("Data/Device3/Device.bmp");
Form3->ShowModal();
}
//-----

```

Крок 2. У верхній частині **Form1** натискаємо кнопкою мишки на **MainMenu** і на назві “Апарати” та у підменю обираємо команду “Розріджувач лугу” і підпункт “Опис процесу”. У вікні редактора коду, що відкриється додаємо у шаблон функції **Process3Click** наступний текст:

```

//-----
void __fastcall TForm1::Process3Click(TObject *Sender)
{
Form2->RichEdit1->Lines->LoadFromFile("Data/Device3/Process.rtf");
Form2->Caption="Опис процесу в розріджувачі лугу";
Form2->ShowModal();
}
//-----

```

Крок 3. У верхній частині **Form1** натискаємо кнопкою мишки на **MainMenu** і на назві “Апарати” та у підменю обираємо команду “Розріджувач лугу”, в якому обираємо підпункт “Параметри”. У вікні редактора коду додаємо у шаблон функції **Characteristic3Click** наступний текст:

```

//-----
void __fastcall TForm1::Characteristic3Click(TObject *Sender)
{
Form6->Caption="Параметри розріджувача лугу";
Form6->ShowModal();
}
//-----

```

Відповідно до налаштувань функцій, описаних вище у пункті 5.2, блок-схема алгоритму з виконання команд і підкоманд у меню “Апарати” буде мати блоки і структуру, показані на рис. 5-3.

5.3 НАВЧАЛЬНИЙ ПРИКЛАД З ПРОГРАМУВАННЯ ФУНКЦІЙ ДЛЯ ВИКОНАННЯ КОМАНД У МЕНЮ «ІНФОРМАЦІЯ»

Відповідно до рис. 4-2 меню команд “Інформація” складається з двох підменю: “Про програму” і “Література”. Для цих підменю потрібно при допомозі **MainMenu** сформувати виконавчий код. Спочатку створимо форму додаткову, яка необхідна для відображення інформації, відповідно до команд з меню “Інформація”.

Крок 1. Виконуємо команду **File/New/Other** і обираємо **About box** для створення нової форми **AboutBox**. У вікні дерева об’єктів мишкою обираємо **AboutBox** і потім у вікні інспектора об’єктів у полі **Caption** вводимо назву “Про програму”.

Крок 2. У вікні дерева об’єктів обираємо **Copyright** та у полі **Caption** вікна інспектора об’єктів вводимо “Розробив C++” програму (прізвисьце, ім’я та по батькові студента).

Крок 3. У вікні дерева об’єктів обираємо **Label1** та у полі **Caption** інспектора об’єктів вводимо напис “Учбова група (позначення) і № групи”.

Крок 4. У вікні дерева об’єктів обираємо **ProductName** та у полі **Caption** інспектора об’єктів вводимо напис “Модульна контрольна робота № 2”.

Крок 5. У дереві об’єктів обираємо **Version** та у полі **Caption** інспектора об’єктів вводимо напис “Версія 1.0”.

Далі можна перейти до **Main Menu** для заповнення у меню команд виконавчого кода відповідно до команд з меню “Інформація”.

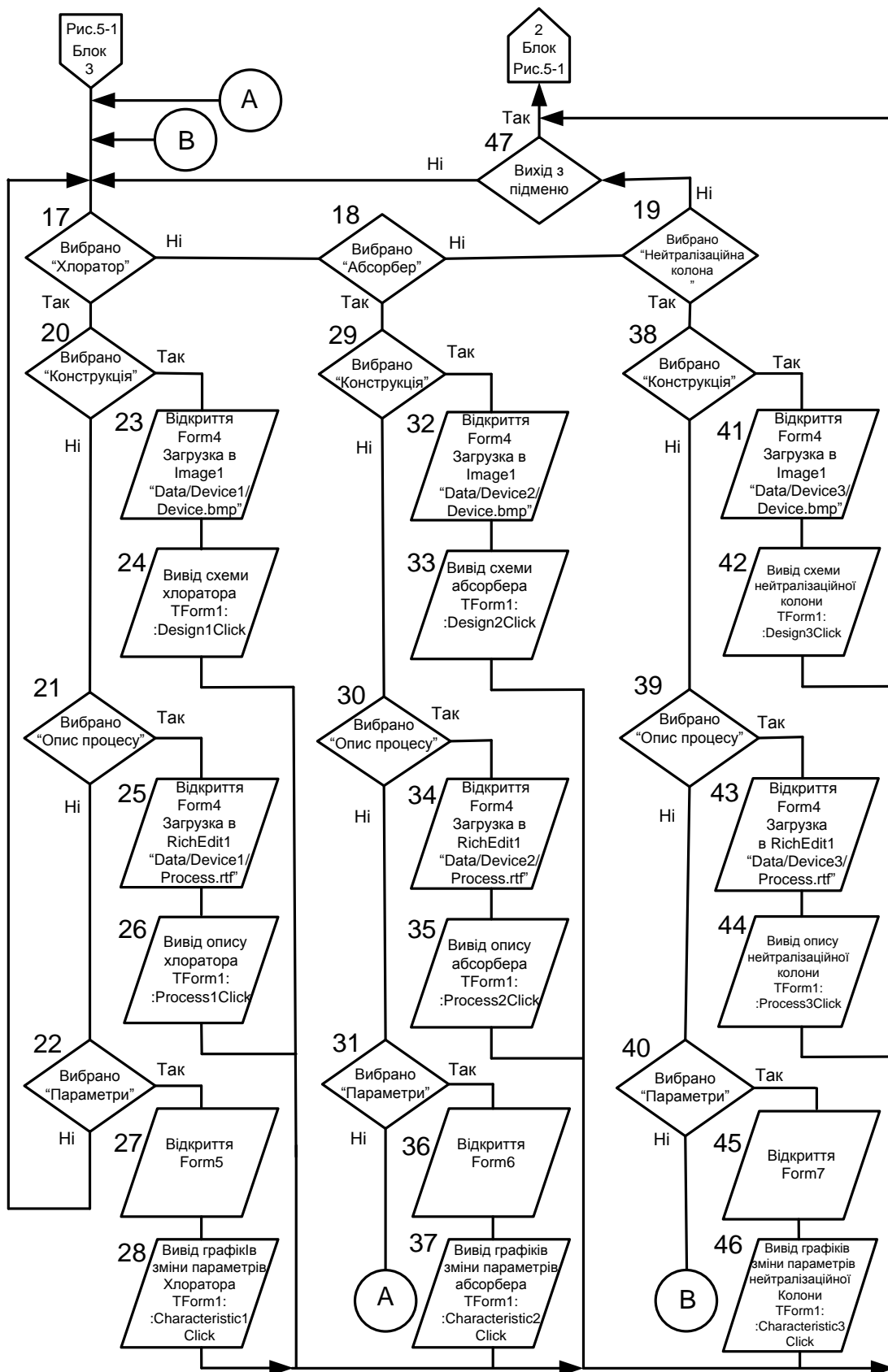


Рис. 5-3. Блок-схема алгоритму з виконання команд у меню "Апарати".

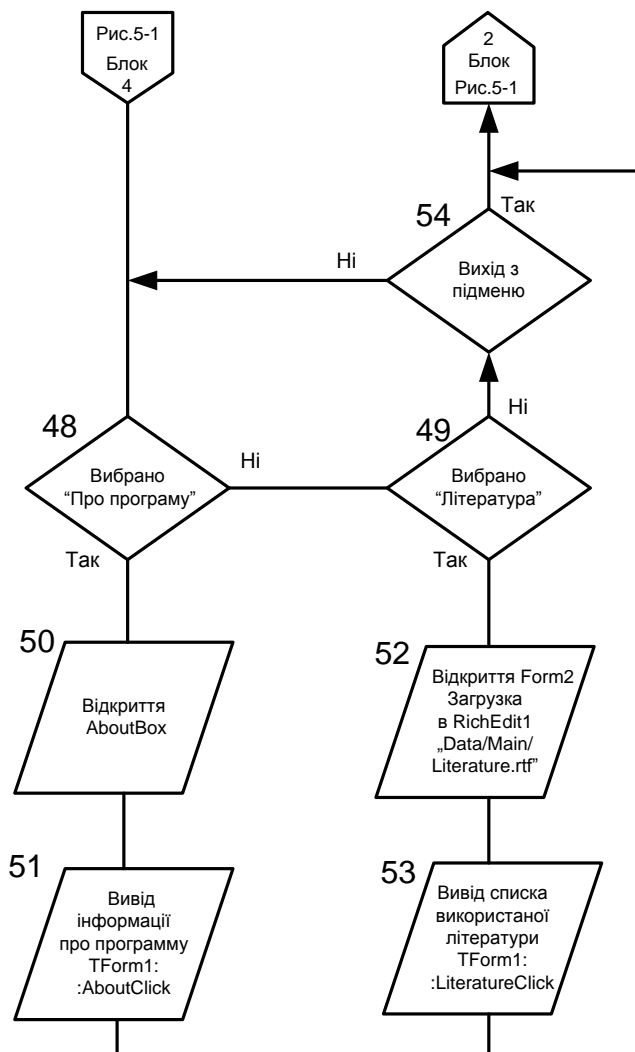


Рис. 5-4. Блок-схема алгоритму з виконання команд у меню “Інформація”.

Налаштування виконавчого коду до команд підменю “Про програму”.

Крок 1. Блок-схема алгоритму з виконання команд у меню “Інформація” показано на рис. 5-4. Відповідно до алгоритмів рис. 5-1 та рис. 5-4 у верхній частині **Form1** натискаємо кнопкою мишки на **MainMenu** і на назві “Інформація” та у підменю обираємо команду “Про програму”. У вікні редактора коду у шаблон функції **AboutClick** додаємо наступний текст:

```

//-----
void __fastcall TForm1::AboutClick(TObject *Sender)
{ AboutBox->ShowModal();
}
//-----

```

Налаштування виконавчого коду до команд підменю “Література”.

Крок 1. У верхній частині **Form1** натискаємо кнопкою мишки на **MainMenu** і потім вибираємо назву “Інформація” та у підменю обираємо команду “Література”. У вікні редактора коду у шаблон функції **LiteratureClick** додаємо наступний текст:

```
//-----  
void __fastcall TForm1::LiteratureClick(TObject *Sender)  
{  
Form2->Caption= ("Література");  
Form2->RichEdit1->Lines->LoadFromFile("Data/Main/Literature.rtf");  
Form2->ShowModal();  
}  
//-----
```

Відповідно до налаштувань функцій, описаних вище, блок-схема алгоритму з виконання команд у меню “Інформація” буде мати блоки і структуру, які показані на рис. 5-4.

5.4 НАВЧАЛЬНИЙ ПРИКЛАД З ПРОГРАМУВАННЯ ФУНКЦІЙ ДЛЯ ВИКОНАННЯ КОМАНД У МЕНЮ «ВИХІД»

Крок 1. У верхній частині **Form1** натискаємо кнопкою мишки на **MainMenu** і на назві “Вихід”. У вікні редактора коду відкриється шаблон функції **ExitClick**, куди і додаємо у наступний текст:

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

Примітка 1. У редакторі коду у верхній частині лістинга до **Form1** необхідно додати наступний текст у вигляді вказівок препроцесору:

```
//-----
#include <vcl.h>
#pragma hdrstop
#include "Unit1.h"
#include "Unit2.h"
#include "Unit3.h"
#include "Unit4.h"
#include "Unit5.h"
#include "Unit6.h"
#include "Unit7.h"
//-----
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//-----
```

Дані вказівки препроцесору забезпечують прикладній C++ програмі відкриття вікон для форм: **Form2**, **Form3**, **Form4**, **Form5**, **Form6** та **Form7**, знаходячись на у вікні форми **Form1**.

6. ФАЙЛИ C++ ПРОГРАМИ ДО НАВЧАЛЬНОГО ПРИКЛАДУ З МОДУЛЬНОЇ КОНТРОЛЬНОЇ РОБОТИ

У процесі проектування і настроювання прикладної програми інтегроване середовище C++ Builder автоматично створює коди (тексти) головного файлу проекту, коди (тексти) окремих модулів форм і коди їхніх заголовних файлів.

6.1 СТРУКТУРА ГОЛОВНОГО ФАЙЛУ ПРОЕКТУ ДО НАВЧАЛЬНОГО ПРИКЛАДУ З МОДУЛЬНОЇ КОНТРОЛЬНОЇ РОБОТИ

Головний файл проекту, призначений для роботи у середовищі Windows, містить головну функцію **WinMain()**. У програмні модулі форм мовою C++ вводяться необхідні оператори для створення оброблювачів різних подій. У заголовні файли цих програмних модулів необхідно вводити відповідні

оголошення. Головний програмний модуль з проекту прикладної C++ програми до навчального прикладу, як правило, майже не бачите і навіть не торкаєтесь його тексту. У виняткових випадках можливо щось змінювати у тексті головного програмного модуля проекту, сгенерованого автоматично за допомогою C++ *Builder*. Тим не менш, розроблювач C++ програми повинен представляти вид, структуру елементів і розуміти, що означають його оператори. Текст файлу головного програмного модуля проекту до навчального прикладу модульної контрольної роботи можна побачити і для цього треба у меню команд C++ *Builder* виконати **Project | View Source**.

Головний програмний модуль проекту до навчального прикладу C++ програми з модульної контрольної роботи № 2 у файлі з назвою **Project1.cpp** наводиться нижче:

```
//-----  
#include <vcl.h>  
#pragma hdrstop  
//-----  
USEFORM("Unit1.cpp", Form1);  
USEFORM("Unit2.cpp", Form2);  
USEFORM("Unit3.cpp", Form3);  
USEFORM("Unit4.cpp", Form4);  
USEFORM("Unit5.cpp", Form5);  
USEFORM("Unit6.cpp", Form6);  
USEFORM("Unit7.cpp", AboutBox);  
//-----  
WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int)  
{  
    try  
    {  
        Application->Initialize();  
        Application->CreateForm(__classid(TForm1), &Form1);
```

```

Application->CreateForm(__classid(TForm2), &Form2);
Application->CreateForm(__classid(TForm3), &Form3);
Application->CreateForm(__classid(TForm4), &Form4);
Application->CreateForm(__classid(TForm5), &Form5);
Application->CreateForm(__classid(TForm6), &Form6);
Application->CreateForm(__classid(TAboutBox), &AboutBox);
Application->Run();
}
catch (Exception &exception)
{
    Application->ShowException(&exception);
}
catch (...)
{
    try
    {
        throw Exception("");
    }
    catch (Exception &exception)
    {
        Application->ShowException(&exception);
    }
}
return 0;
}
//-----

```

Починається файл головного програмного модуля рядками із символами # , тобто вказівками препроцесору і серед них важливе місце займає директива **#include <vcl.h>**, яка містить заголовний файл бібліотеки компонент **C++ Builder**. Після директив препроцесору у файлі **Project1.cpp** розміщуються

рядки, що починаються назвами **USERES** і **USEFORM**. Данні рядки являють собою макроси, які забезпечують підключення ресурсів і інших даних до проекту файлів форм. Препроцесор розгортає ці макроси у відповідний код. Ви бачите також макрос **USEFORM**, який підключає спроектовані форми у навчальному прикладі **C++** програми до модульної контрольної роботи № 2. Інтегроване середовище **C++ Builder** автоматично формує рядки з макросами **USEFORM** для кожної відкритої нової форми у проекті прикладної **C++** програми. Перший параметр макросу містить ім'я файлу програмного модуля до відповідної форми, наприклад, файл “**Unit1.cpp**”, а другий параметр це ім'я форми – “**Form1**”.

Далі розміщуються рядки головної функції програми - **WinMain()** і першим параметром є дескриптор даної прикладної **C++** програми. Параметр дескриптор - це деякий унікальний показник, що дозволяє **Windows** розбиратись серед багатьох відкритих вікон різних програм. Другий параметр функції **WinMain()** являє собою дескриптор програми, запущеної у роботу вашою програмою. Третій параметр є показником на рядок з нульовим символом наприкінці, що містить параметри, передані у програму через командний рядок. Іноді такі параметри використовуються для переключення режимів роботи програми, або для завдання різних опцій при запуску програми з диспетчера програм чи функцією **WinExec**. Останній параметр головної функції визначає вікно прикладної програми. Цей параметр може надалі передаватися у функцію **ShowWindow**. Ім'я **WINAPI** показує, що будуть використовуватися дескриптори по зверненню до різних функцій **API Windows** користувальницького інтерфейсу **Windows**.

Після заголовка основної функції розташоване її тіло, укладене у фігурні дужки. У тілі основної функції **WinMain()** перший виконуваний оператор **Application->Initialize** ініціалізує об'єкти компонентів прикладної **C++** програми. Наступні оператори **Application->CreateForm** створюють об'єкти форм. Форми у прикладній програмі створюються у послідовності, в якій розташовані ці оператори. Перша зі створюваних форм є головною.

Останній оператор **Application-Run** починає власне виконання програми. Далі програма C++ очікує події і виконує для кожної події відповідну обробку, що ми і бачимо на дисплеє комп'ютерна у вигляді роботи програми [1].

Розглянуті оператори тіла функції **WinMain()** укладені в блок з ім'ям **try**, після якого розташований блок з назвою **catch**, який є структурою, зв'язану з обробкою так званих виключень з аварійних ситуацій, які виникають при роботі C++ програми. Можуть з'являтися різного виду помилки: ділення значення на нуль; переповнення результату обчислень; спроба відкрити неіснуючий файл і т.п.. При виникненні таких помилок C++ програма генерує тимчасовий об'єкт спеціального виду з уточнюючим повідомленням, так званими "виключення" і подальше обчислення в даному блоці припиняється. Якщо така аварійна ситуація виникає, то будуть виконані оператори, записані в блоці **catch**. За замовчуванням у цей блок записується стандартний оброблювач виключень за допомогою функції **Application->ShowException**.

Останнім оператором у тілі основної функції **WinMain()** є оператор **return(0)**, який завершує виконання функції з кодом **0**, що означає про успішне закінчення роботи C++ програми [5]. Усі описані вище оператори головного файлу були записані у нього автоматично в процесі проектування прикладної C++ програми до навчального прикладу з модульної контрольної роботи № 2. Якщо додається нова форма у проект прикладної C++ програми, то в цьому випадку автоматично додається відповідний макрос **USERFORM** і оператор конструктора **Application->CreateForm** для створення відповідної нової форми.

6.2 СТРУКТУРА ФАЙЛІВ ПРОГРАМНИХ МОДУЛІВ ВІКОННИХ ФОРМ

Кожен програмний модуль форми складається з двох файлів: заголовного файлу, з описом класу форми, і файлу з реалізації форми [1]. Заголовний файл **Unit1.h** до навчального прикладу з модульної контрольної роботи № 2 має наступний текст:

```
//-----  
#ifndef Unit1H
```

```

#define Unit1H
//-----
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <ExtCtrls.hpp>
#include <Menus.hpp>
#include <Graphics.hpp>
#include <ImgList.hpp>
#include <ComCtrls.hpp>
//-----
class TForm1 : public TForm
{
__published:      // IDE-managed Components
    TMainMenu *MainMenu1;
    TLabel *Label1;
    TImage *Image1;
    TMenuItem *Technology;
    TMenuItem *Scheme;
    TMenuItem *Devices;
    TMenuItem *Info;
    TMenuItem *Device1;
    TMenuItem *Device2;
    TMenuItem *Device3;
    TMenuItem *Products;
    TMenuItem *Description;
    TMenuItem *Design1;
    TMenuItem *Characteristic1;
    TMenuItem *Design2;

```

```
TMenuItem *Process2;
TMenuItem *Characteristic2;
TMenuItem *Design3;
TMenuItem *Process3;
TMenuItem *Characteristic3;
TMenuItem *About;
TMenuItem *Exit;
TMenuItem *Literature;
TImage *Image2;
TImage *Image3;
TImage *Image4;
TImage *Image5;
TImage *Image6;
TImage *Image7;
TImage *Image8;
TImage *Image9;
TImage *Image10;
TImage *Image11;
TImage *Image12;
TImage *Image13;
TImage *Image14;
TImage *Image15;
TImage *Image16;
TImage *Image17;
TImage *Image18;
TImageList *ImageList1;
TMenuItem *Process1;
void __fastcall ExitClick(TObject *Sender);
void __fastcall AboutClick(TObject *Sender);
void __fastcall FormCreate(TObject *Sender);
```

```
void __fastcall SchemeClick(TObject *Sender);
void __fastcall DescriptionClick(TObject *Sender);
void __fastcall ProductsClick(TObject *Sender);
void __fastcall Design1Click(TObject *Sender);
void __fastcall Design2Click(TObject *Sender);
void __fastcall Design3Click(TObject *Sender);
void __fastcall Process1Click(TObject *Sender);
void __fastcall Process2Click(TObject *Sender);
void __fastcall Process3Click(TObject *Sender);
void __fastcall Characteristic1Click(TObject *Sender);
void __fastcall Characteristic2Click(TObject *Sender);
void __fastcall Characteristic3Click(TObject *Sender);
void __fastcall Image2MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y);
void __fastcall Image1MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y);
void __fastcall Image3MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y);
void __fastcall Image4MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y);
void __fastcall Image5MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y);
void __fastcall Image7MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y);
void __fastcall Image6MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y);
void __fastcall Image8MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y);
void __fastcall Image9MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y);
```

```

void __fastcall Image10MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y);
void __fastcall Image11MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y);
void __fastcall Image12MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y);
void __fastcall Image13MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y);
void __fastcall Image14MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y);
void __fastcall Image15MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y);
void __fastcall Image16MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y);
void __fastcall Image17MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y);
void __fastcall Image18MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y);
void __fastcall LiteratureClick(TObject *Sender);
void __fastcall Image11Click(TObject *Sender);
void __fastcall Image3Click(TObject *Sender);
void __fastcall Image9Click(TObject *Sender);
void __fastcall Image10Click(TObject *Sender);
private:    // User declarations
public:    // User declarations
    __fastcall TForm1(TComponent* Owner);
};
//-----
extern PACKAGE TForm1 *Form1;
//-----

```


#endif

Відповідний заголовний файл для форми **Form1** починається з директив препроцесору, які додаються автоматично. Інтегроване середовище *C++ Builder* само додає необхідні директиви **#include** для підключення зазначених копій файлів, у яких описані ті компоненти, перемінні, константи і функції, які використовуються в даному модулі форми. Однак для деяких функцій таке автоматичне підключення не виконується і у таких випадках розробник *C++* програми сам повинен вручну додавати відповідні директиви **#include**.

Після директив препроцесору **#include** розташовується опис класу форми. Наприклад, ім'я класу головної форми позначається **TForm1**. Клас форми складається з трьох розділів: **_published** - відкритий розділ, що містить оголошення розміщених на формі компонентів і для них оброблювачів відповідних подій, другий - **private**, що вказує на закритий розділ класу, і **public** - відкритий розділ класу [5]. У наведеному заголовному файлі в розділі **_published** можна побачити оголошення покажчиків на компоненти: **TMainMenu**, **TLabel**, **TImage**, **TMenuItem** та інші. Далі у розділі **_published** розташовані оголошення прототипів функцій, які формуються розробником прикладної *C++* програми. Усе, що записано у розділі **_published**, вносить у нього *C++ Builder* автоматично у процесі проектування форми до програми. Як правило, розробнику прикладної програми не приходиться працювати з цим розділом.

У розділі **private** і **public** розробник програми може додавати такі свої оголошення: типів даних, перемінних і функцій. Оголошення, які записані в розділі **public**, будуть доступні для інших класів і модулів. Те, що оголошено у розділі **private**, доступно тільки в межах даного програмного модуля [5].

C++ Builder самостійно включає у розділ **public** оголошення (прототип) конструктора, до спроектованої форми. Після рядка **extern PACKAGE** можуть міститися оголошення: типів даних, перемінних і функцій, які не включені у клас форми і до них може бути доступ з інших блоків.

Файл реалізації програмного модуля форми **Form1** до навчального прикладу з модульної контрольної роботи № 2 має назву **Unit1.cpp** і наступний текст:

```
//-----  
#include <vcl.h>  
#pragma hdrstop  
#include "Unit1.h"  
#include "Unit2.h"  
#include "Unit3.h"  
#include "Unit4.h"  
#include "Unit5.h"  
#include "Unit6.h"  
#include "Unit7.h"  
//-----  
#pragma package(smart_init)  
#pragma resource "*.dfm"  
TForm1 *Form1;  
//-----  
__fastcall TForm1::TForm1(TComponent* Owner)  
    : TForm(Owner)  
{  
}  
//-----  
void __fastcall TForm1::ExitClick(TObject *Sender)  
{  
    Close();  
}  
//-----  
void __fastcall TForm1::AboutClick(TObject *Sender)  
{  
    AboutBox->ShowModal();
```

```

}
//-----
void __fastcall TForm1::FormCreate(TObject *Sender)
{
Image1->Picture->LoadFromFile("Data/Main/Background.bmp");
Image2->Visible = false;
Image3->Visible = false;
Image4->Visible = false;
Image5->Visible = false;
Image6->Visible = false;
Image7->Visible = false;
Image8->Visible = false;
Image9->Visible = false;
Image10->Visible = false;
Image11->Visible = false;
Image12->Visible = false;
Image13->Visible = false;
Image14->Visible = false;
Image15->Visible = false;
Image16->Visible = false;
Image17->Visible = false;
Image18->Visible = false;
}
//-----
void __fastcall TForm1::SchemeClick(TObject *Sender)
{
Image1->Picture->LoadFromFile("Data/Main/Scheme.bmp");
Image2->Visible = true;
Image3->Visible = true;
Image4->Visible = true;

```

```

Image5->Visible = true;
Image6->Visible = true;
Image7->Visible = true;
Image8->Visible = true;
Image9->Visible = true;
Image10->Visible = true;
Image11->Visible = true;
Image12->Visible = true;
Image13->Visible = true;
Image14->Visible = true;
Image15->Visible = true;
Image16->Visible = true;
Image17->Visible = true;
Image18->Visible = true;
Label1->Visible = true;
}
//-----
void __fastcall TForm1::DescriptionClick(TObject *Sender)
{
Form2->RichEdit1->Lines->LoadFromFile("Data/Main/Scheme.rtf");
Form2->Caption="Опис процесу отримання хлористого метилену";
Form2->ShowModal();
}
//-----
void __fastcall TForm1::ProductsClick(TObject *Sender)
{
Form2->RichEdit1->Lines->LoadFromFile("Data/Main/Products.rtf");
Form2->Caption="Продукція";
Form2->ShowModal();
}

```

```

//-----
void __fastcall TForm1::Design1Click(TObject *Sender)
{
Form3->Caption="Конструкція хлоратора";
Form3->Image1->Picture->LoadFromFile("Data/Device1/Device.bmp");
Form3->ShowModal();
}
//-----

void __fastcall TForm1::Design2Click(TObject *Sender)
{
Form3->Caption="Конструкція абсорбера";
Form3->Image1->Picture->LoadFromFile("Data/Device2/Device.bmp");
Form3->ShowModal();
}
//-----

void __fastcall TForm1::Design3Click(TObject *Sender)
{
Form3->Caption="Конструкція розріджувача луѓу";
Form3->Image1->Picture->LoadFromFile("Data/Device3/Device.bmp");
Form3->ShowModal();
}
//-----

void __fastcall TForm1::Process1Click(TObject *Sender)
{
Form2->RichEdit1->Lines->LoadFromFile("Data/Device1/Process.rtf");
Form2->Caption="Опис процесу в хлораторі";
Form2->ShowModal();
}
//-----

void __fastcall TForm1::Process2Click(TObject *Sender)

```

```

{
Form2->RichEdit1->Lines->LoadFromFile("Data/Device2/Process.rtf");
Form2->Caption="Опис процесу в абсорбери";
Form2->ShowModal();
}
//-----

void __fastcall TForm1::Process3Click(TObject *Sender)
{
Form2->RichEdit1->Lines->LoadFromFile("Data/Device3/Process.rtf");
Form2->Caption="Опис процесу в розріджувачі луку";
Form2->ShowModal();
}
//-----

void __fastcall TForm1::Characteristic1Click(TObject *Sender)
{
Form4->Caption="Параметри хлоратора";
Form4->ShowModal();
}
//-----

void __fastcall TForm1::Characteristic2Click(TObject *Sender)
{
Form5->Caption="Параметри абсорбера";
Form5->ShowModal();
}
//-----

void __fastcall TForm1::Characteristic3Click(TObject *Sender)
{
Form6->Caption="Параметри розріджувача луку";
Form6->ShowModal();
}

```

```

//-----
void __fastcall TForm1::Image2MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("3 - Змішувач");
}
//-----

void __fastcall TForm1::Image1MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption=("");
}
//-----

void __fastcall TForm1::Image3MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("4 - Хлоратор");
}
//-----

void __fastcall TForm1::Image4MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("1 - Змішувач");
}
//-----

void __fastcall TForm1::Image5MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("2 - Підігрівник");
}

```

```

//-----
void __fastcall TForm1::Image7MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("16 - Сепаратор");
}
//-----
void __fastcall TForm1::Image6MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("5 - Сажовловлювач");
}
//-----
void __fastcall TForm1::Image8MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("10 – Графітовий холодильник");
}
//-----
void __fastcall TForm1::Image9MouseMove(TObject *Sender, TShiftState Shift,
    int X, int Y)
{
Label1->Caption= ("7 - Абсорбер");
}
//-----
void __fastcall TForm1::Image10MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
Label1->Caption= ("8 - Абсорбер");
}

```



```

//-----
void __fastcall TForm1::Image11MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
Label1->Caption= ("9 – Розріджувач лугу");
}
//-----
void __fastcall TForm1::Image12MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
Label1->Caption= ("11 – Збірник концентрованої соляної кислоти");
}
//-----
void __fastcall TForm1::Image13MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
Label1->Caption= ("12 - Компресор");
}
//-----
void __fastcall TForm1::Image14MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
Label1->Caption= ("15 - Фільтр");
}
//-----
void __fastcall TForm1::Image15MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
Label1->Caption= ("14 - Конденсатор");
}

```

```

//-----
void __fastcall TForm1::Image16MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
    Label1->Caption= ("13 - Осушувачі");
}
//-----
void __fastcall TForm1::Image17MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
    Label1->Caption= ("17 – Розріджувач лугу");
}
//-----
void __fastcall TForm1::Image18MouseMove(TObject *Sender,
    TShiftState Shift, int X, int Y)
{
    Label1->Caption= ("6 – Повітряний холодильник");
}
//-----
void __fastcall TForm1::LiteratureClick(TObject *Sender)
{
    Form2->Caption= ("Література");
    Form2->RichEdit1->Lines->LoadFromFile("Data/Main/Literature.rtf");
    Form2->ShowModal();
}
//-----
void __fastcall TForm1::Image11Click(TObject *Sender)
{
    Form3->Caption="Конструкція розріджувача лугу";
    Form3->Image1->Picture->LoadFromFile("Data/Device3/Device.bmp");
}

```

```

Form3->ShowModal();
}
//-----
void __fastcall TForm1::Image3Click(TObject *Sender)
{
Form3->Caption="Конструкція хлоратора";
Form3->Image1->Picture->LoadFromFile("Data/Device1/Device.bmp");
Form3->ShowModal();
}
//-----
void __fastcall TForm1::Image9Click(TObject *Sender)
{
Form3->Caption="Конструкція абсорбера";
Form3->Image1->Picture->LoadFromFile("Data/Device2/Device.bmp");
Form3->ShowModal();
}
//-----
void __fastcall TForm1::Image10Click(TObject *Sender)
{
Form3->Caption="Конструкція абсорбера";
Form3->Image1->Picture->LoadFromFile("Data/Device2/Device.bmp");
Form3->ShowModal();
}
//-----

```

Директиви **#include** використовуються для включення копії файлу в те місце, де знаходиться ця директива препроцесора, а директиви **#pragma** вказують на необхідність використання пакетів (**package**) і файлів ***.dfm**. Після автоматично включених директив препроцесору в файлі модуля форми розташовуються також автоматично приєднанні оголошення покажчика на об'єкт форми, наприклад, **Form1**, а потім йде виклик конструктора для даної форми. Тіло

відповідної функції порожнє, але розробник програми може вставляти у тіло якісь свої оператори. При створенні вікна даної форми ці оператори будуть виконуватися. У ці оператори також можна вставити початкові настроювання якісь властивостей форми. Після конструктора форми розташовуються описи усіх функцій, оголошених у заголовному файлі форми **Unit1.h**. У цій частині можна додавати оголошення любых типів даних, констант, перемінних, котрі не оголошені в заголовному файлі, та також робити описи будь-яких функцій, не згаданих у заголовному файлі.

По команді **Save As** імена файлам модулів форм прикладної програми призначає сама **C++ Builder** за замовчуванням: для модуля першої форми задається назва "**Unit1.cpp**", для модуля другої форми - "**Unit2.cpp**" і т.д.. Ці імена за бажанням розробник **C++** програми може змінювати, при цьому залишаючи першу букву з назви, наприклад, для першого модуля **U_нове-ім'я-1.cpp**, а для другого модуля задати - **U_нове-ім'я-2.cpp** . Такі зміни назв файлів для модулів форм **C++ Builder** враховує автоматично з записом нових імен у відповідних указівках препроцесору в заголовному файлі форми.

6.3 КОМПІЛЯЦІЯ ФАЙЛІВ ПРОЕКТУ І КОМПОНУВАННЯ ВИКОНАВЧОГО КОДУ C++ ПРОГРАМИ ДО НАВЧАЛЬНОГО ПРИКЛАДУ МОДУЛЬНОЇ КОНТРОЛЬНОЇ РОБОТИ

Перетворення файлів проекту прикладної програми **C++** у виконавчий код (файл з розширенням **.exe**) містить у собі два послідовних процеси: **компіляцію** і **компонування проекту**. Компіляція, у свою чергу, містить у собі роботу препроцесора, що перетворює і перевіряє початкові тексти файлів, і власне процес їх компіляції. Компіляція прикладної програми в **C++ Builder** може виконуватися декількома способами. Компіляція з наступним виконанням прикладної програми здійснюється по команді **Run | Run**, або швидкою кнопкою (кнопка з зеленим трикутником), чи "гарячою " клавішею **F9**. Якщо не виявлені непоправні помилки, тоді у цьому випадку виконується компіляція **C++** програми і компонування проекту з створенням виконуємого файлу (***.exe**), який запускається на виконання прикладної програми. У процесі компіляції і

компонування на екрані з'являється вікно, показане на рис. 6.1. У його верхньому рядку видне ім'я компілюемого проекту. У наступному рядку відображається назва поточної операції: компіляція визначеного модуля форми чи компоновка (**Linking**). У третьому рядку вікна відображається поточний рядок програмного модуля (**Current line**), який обробляється компілятором, і загальне число рядків у програмному модулі (**Total Lines**). В нижньому рядку відображається виявлена кількість зауважень (**Hints**), попереджень (**Warnings**) і помилок (**Errors**).

Кнопка **Cancel** унизу вікна дозволяє перервати процес компіляції та компоновання. Якщо у файлах компілюемого проекту зустрілися непоправні помилки, тоді виконуємий файл не буде створений. Якщо таких помилок немає, файл що виконується створюється, але в цьому випадку компілятор виводить повідомлення (попередження і зауваження), які необхідно переглянути і вивчити.

Компілятор у *C++ Builder* налаштован на режим компіляції проекту прикладної програми, яка складається з декількох програмних модулів форм і будуть компілюватися тільки ті файли модулів, тексти яких були змінені з моменту попереднього компоновання проекту. Такий режим компілятора заощаджує час компіляції файлів у великих проектах прикладних програм.

При виконанні команди для компіляції файлів можна задати командний рядок, якщо прикладна програма передбачає передачу в неї параметрів. Для цього спочатку виконується команда **Run | Parameters** і у вікні, що відкриється, необхідно написати необхідний командний рядок. Не завжди треба компілювати проект і відразу виконувати прикладну програму. Часто важливіше просто перевірити, чи не містять останні зміни яких-небудь помилок тексту програмного модуля форми. У цьому випадку нема рації гаяти час на виконання

прикладної програми і краще скористатися такими командами *C++ Builder*: **Project | Compile Unit**, **Project | Make Project** або **Project | Build Project**.

Команда **Compile** виконує компіляцію тільки того програмного модуля форми, що показується у вікні "Редактор Коду" чи вибран у вікні "Менеджер Проектів". Ця команда дозволяє найбільш швидко перевірити наявність

помилки чи зауважень при компіляції програмного модуля форми і також не здійснюється компонування прикладної програми та не компілюються ніякі інші модулі. Якщо компіляція пройшла успішно, створюється об'єктний файл **.obj** до відкомпільованого програмного модуля форми.

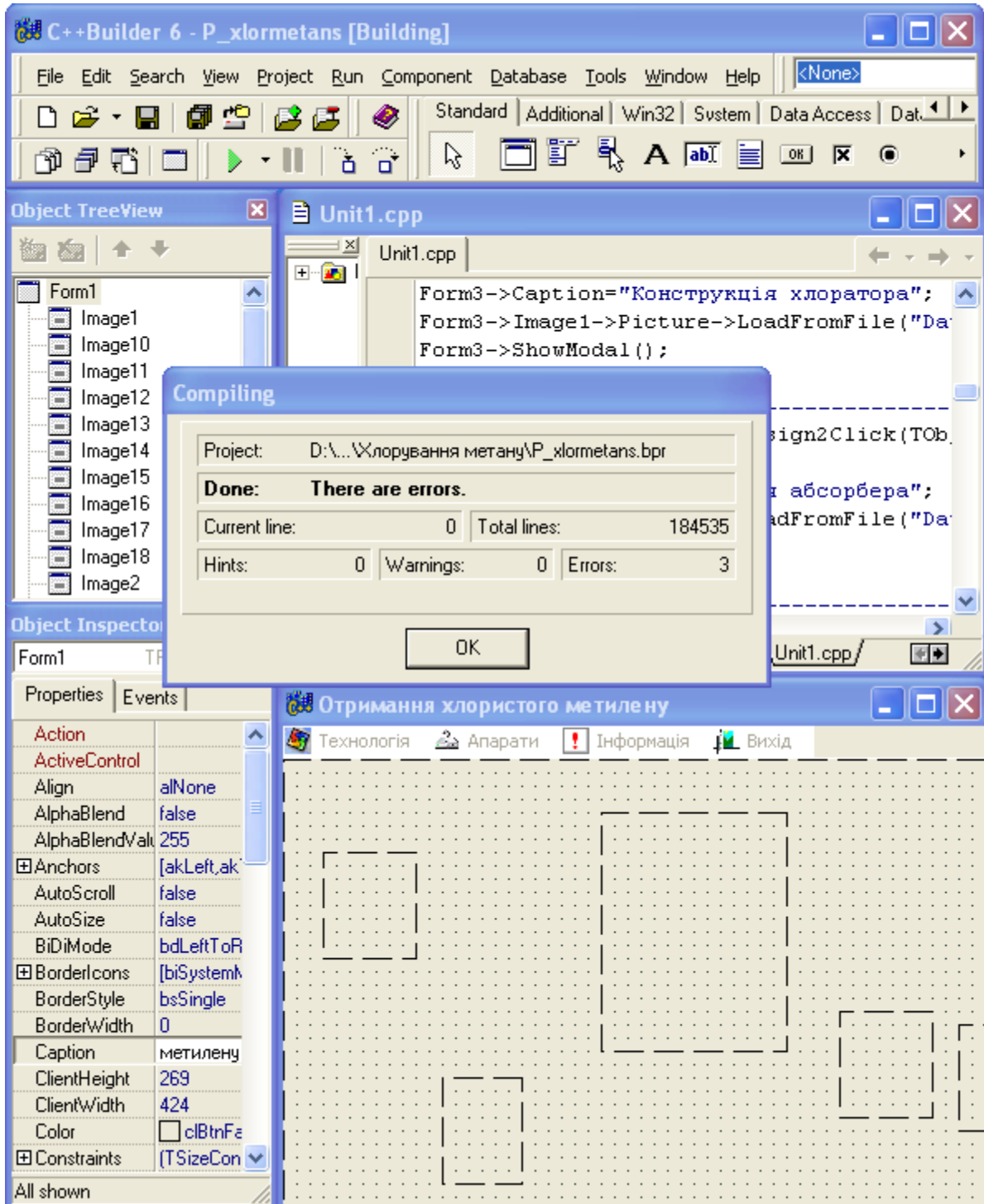


Рис. 6.1. Вікна *C++ Builder* і вікно компілятора з інформацією про процес компіляції та компонування виконавчого коду прикладної C++ програми.

Команда **Make** виконує компіляцію тільки тих програмних модулів форм, тексти у яких були змінені з моменту попереднього компонування проекту.

Якщо компіляція пройшла успішно, то створюються об'єктні файли **.obj** програмних модулів форм і здійснюється компювання прикладної програми. Якщо успішно пройшло компонування, то створюється виконуємий файл **.exe**. Таким чином, відмінність команди **Make** від **Run** тільки в тім, що після компонування виконуємий файл **.exe** не виконується.

Команда **Build** подібна команді **Make** за одним виключенням - компілюються всі програмні модулі форм, незалежно від того, коли вони останній раз змінювалися. Така ситуація виникає, наприклад, коли змінювалися параметри настроювань *C++ Builder* у процесі проектування і налагодження прикладної **C++** програми.

За замовчуванням усі ці команди *C++ Builder* виконує у фоновому режимі. Ця можливість застосовується при компіляції великих проектів прикладних програм, але необхідно враховувати, що при виконанні інших робіт фонові компіляції здійснюються повільніше. Крім того, по завершенні компіляції у фоновому режимі вікно компілятора зникає на екрані та при цьому не показуються результати компіляції: чи пройшла вона успішно, чи маються зауваження. Фоновий режим компіляції можна відключити за допомогою команди **Tools | Environment Options** і на сторінці **Preferences** потрібно зняти опцію **Background Compilation**. При компіляції остаточного варіанту розробленої прикладної програми необхідно враховувати, що *C++ Builder* може створювати **C++** програми двох видів: автономні виконувані файли **.exe** і програми з підтримкою пакетів (**packages**) часу виконання. При проектуванні прикладної програми для **Windows C++ Builder** створює проект (комплект файлів) і цей список наводиться у таблиці 6-1.

Якщо передбачається виконання прикладної програми на якомусь іншому комп'ютері, то в цьому випадку потрібно створювати автономний виконуємий файл, у котрому буде розміщатися **C++** програма і всі її необхідні ресурси.

Автономний файл не вимагає наявності на комп'ютері користувача не тільки середовища *C++ Builder*, але і яких-небудь спеціальних бібліотек.

Таблиця № 6-1. Список файлів проекту прикладної програми.

Головний файл проекту (.cpp)	C++ Builder створює файл .cpp для головної функції WinMain, ініціалізується програма і запускається на виконання.
Файл опцій проекту (.bpr)	Цей текстовий файл містить установки опцій проекту і вказівки на те, які файли повинні компілюватися і компонуватися у проекті. 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 програмного модуля форми (.cpp)	Кожній створюваній формі відповідає текстовий файл програмної реалізації модуля форми. Можна також створювати програмні модулі, не зв'язані з формами.
Заголовний файл модуля (.h)	Кожній створюваній формі відповідає не тільки файл реалізації програмного модуля форми, але ще і заголовні файли з описом класу форми.
Файл форми(.dfm)	Двоічний або текстовий файл, з даними про створені форми C++ Builder. Цей файл можна переглядати в текстовому вигляді чи у вигляді форми.
Заголовний файл компоненти (.hpp)	Файл нової створеної компоненти. Також часто ці файли підключаються до проекту з бібліотеки компонентів, розташованих у каталозі Include/VCL.
Файл групи проектів(.bpg)	Текстовий файл, створюваний до групи проектів.
Файли пакетів(.bpl) і (.brc)	Ці файли використовує C++ Builder при роботі з пакетами: .bpl - файл самого проекту;

	.brk - файл, що визначає компіляцію і компонування проекту.
Файл робочого столу проекту (.dsk)	У текстовому файлі C++ Builder зберігає інформацію: про останній сеанс роботи..
Файли резервних копій (.~br, .~df, .~cr, .~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 повернутися до попереднього не зіпсованому варіанту проекту.

Таблиця № 6-2. Група файлів, створюваних у процесі компіляції.

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

Для роботи прикладної програми у Windows можуть у склад проекту C++ *Builder* входити файли, які наводяться у таблиці 6-3.

Таблиця № 6-3. Додаткові файли до проекту прикладної програми.

Файли довідки (.hlp)	Стандартні файли довідок з Windows, які можуть використовуватися в прикладній програмі.
Файли зображень або графічні файли (.wmf, .bmp, .ico)	Дані файли звичайно використовуються в програмах для створення привабливого і дружнього інтерфейсу в Windows.

Пакети (**packages**) – це спеціальні бібліотеки, які при виконанні файлу **.exe** динамічно приєднуються і у цих бібліотеках розташовані візуальні та невізуальні компоненти, інші об'єкти, функції, процедури та інші файли. Ці, створювані бібліотеки DLL, при розробці прикладної програми дозволяють створювати дуже мали виконувані програмні модулі, що звертаються за підтримкою до пакетів. Файли пакетів мають розширення **.bpl** (*Borland package library*), щоб відрізнити їх від стандартних бібліотек DLL.

Пакети розділяються на два види: *пакети часу проектування* і *пакети часу виконання*. Пакети часу проектування викликає тільки сама **C++Builder** у процесі розробки і налагодження прикладної **C++** програми. У пакетах часу виконання розміщуються бібліотеки візуальних і невізуальних компонент з **VCL**, що використовуються на формах, створюваної **C++** програми. Виконавчий файл (**.exe**), що створюється при настроюванні компіляції в режимі (**packages**) часу виконання не може працювати без бібліотек з цих пакетів. Даний режим компіляції програмних модулів форм відрізняється створенням малого по розмірам файлу (**.exe**) за рахунок того, що майже вся частина кодів прикладної програми розміщується у пакетах часу виконання ^[3].

Для одержання автономного виконавчого файлу (**.exe**) необхідно перед компіляцією програмних модулів задати настроювання для автономного режиму і для цього необхідно: виконати команду **Project | Options**; перейти на сторінку з закладкою **Packages**; зняти прапорець для режиму **Build with runtime packages** (будувати з підтримкою пакетів часу виконання); кнопка **OK**. Які пакети і бібліотеки DLL використовує прикладна програма можна довідатись за

допомогою програми **tdump.exe**, що входить у ...**BIN**, але працює вона в MS DOS.

7.ЛІТЕРАТУРА

1. Архангельский, А. Я. Программирование в С++ Builder 6. [Текст] / А. Я. Архангельский // – М.: ЗАО «Издательство БИНОМ», 2002. – 1152 с. Библиогр.: с. 1150–1151. 4000 экз. ISBN 5-7989-0239-0.
2. Архангельский, А. Я. С++ Builder 6. Справочное пособие. Книга 1. Язык С++. [Текст] / А. Я. Архангельский // – М.: ЗАО «Издательство БИНОМ», 2002. – 554 с. Библиогр.: с. 541–543. 4000 экз. ISBN 5-9518-0007-2.
3. Архангельский, А. Я. С++ Builder 6. Справочное пособие. Книга 2. Классы и компоненты [Текст] / А. Я. Архангельский // – М.: ЗАО «Издательство БИНОМ», 2002. – 528 с. Библиогр.: с. 525–526. 4000 экз. ISBN 5-9518-0009-9.
4. Культин, Н. Б. Самоучитель С++ Builder [Текст] / Н. Б. Культин // – СПб.: БХВ-Петербург, 2004. – 320 с. Библиогр.: с. 317. 4000 экз. ISBN 5-94157-378-2.
5. Шилд, Г. Полный справочник по С++ [Текст] / Г. Шилд // 4-е издание. : Пер. с англ. – М.: Издательский дом “Вильямс”, 2006. – 800 с. 3000 экз. ISBN 5-8459-0489-7.

Д1. Приклад титульної сторінки

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО”**

*Інженерно-хімічний факультет,
кафедра “Автоматизації хімічних виробництв”*

ЗАПИСКА МОДУЛЬНОЇ КОНТРОЛЬНОЇ РОБОТИ

з курсу «Технології розробки програмного забезпечення-2»
кредитного модуля “Візуальне програмування прикладних програм” на тему:
**«Обробка подій мишки у меню команд прикладної C++ програми до
мнемосхеми технологічного процесу виробництва хлорметанів»**

Виконана:

Прізвисьце, Ім'я, По батькові

Студент(ка) X курсу ІХФ

Група ЛА-ХХ

Залікова книжка № ЛА-XXXX

*Керівник модульної
контрольної роботи:*

доц. Ковалевський В.М.

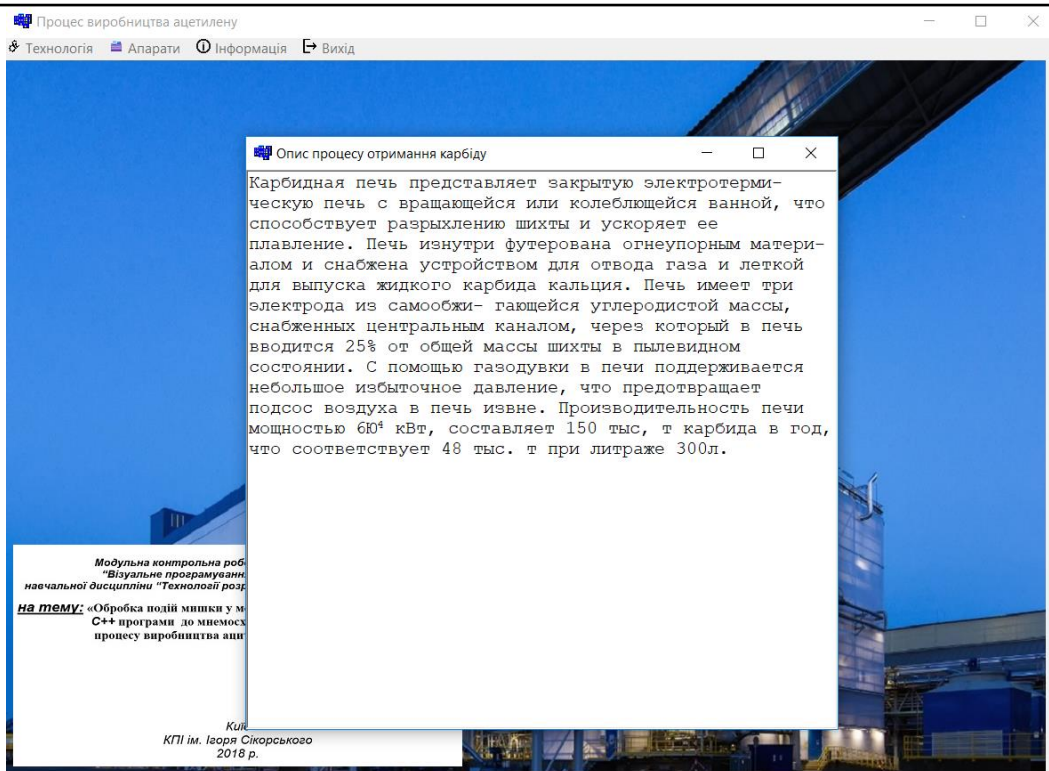
Нарахована сума балів: _____

Дата _____ Підпис _____

Київ
КПІ ім. Ігоря Сікорського
2018

Зміст	Стор.
1. Завдання до модульної контрольної роботи № 2	2
2. Меню команд та інформація до програми «Обробка подій мишки у меню команд прикладної С++ програми до мнемосхеми технологічного процесу(назва хімічного виробництва)»	3
2.1 Структура команд та інформація до меню команд С++ програми (показується рисунок меню команд та відповідна інформація, яка повинна виводитися у вікні до обраних мишкою команд у меню команд).....	.3
2.2 Опис алгоритму до обробки подій мишки при встановленнях і зміщеннях курсору на зображенні мнемосхеми технологічного процесу(рисунок блок-схеми алгоритму на аркушу формату А3).....	5
3. Структура віконних <i>Forms</i> , які використовуються у С++ програмі та алгоритми їх відкриття і закриття (рисунок блок-схеми алгоритму на аркушу формату А3).....	8
4. Лістинги програмних модулів прикладної С++ програми до МКР - № 2.....	12
5. Література	20
6. Додаток:	
6.1 Виконуючий файл і файли проекту на диску CD-R до прикладної програми «Обробка подій мишки у меню команд прикладної С++ програми до мнемосхеми технологічного процесу (назва хімічного виробництва)»	21

Підп. и дата									
Взам. ине.									
Инв. №									
Підп. и дата									
						ТРПЗ-МКР2.ЛА-ХХ.ХХХХ.000.04.Пз			
	Ли	Изм.	№ докум.	Подп.	Дат				
Инв. № подл	Разраб.	п. і. б.					Лім.	Лист	Листів
	Пров.	Ковалевський В.М.						2	19
	Т.					Мнемосхема технологічного процесу виробництва хлорметанів			
	Н.					НТУУ «КПІ»			
	Утв.					ІХФ, ЛА-ХХ			



На рис. 4.7 Зображено роботу команди «Апарати».

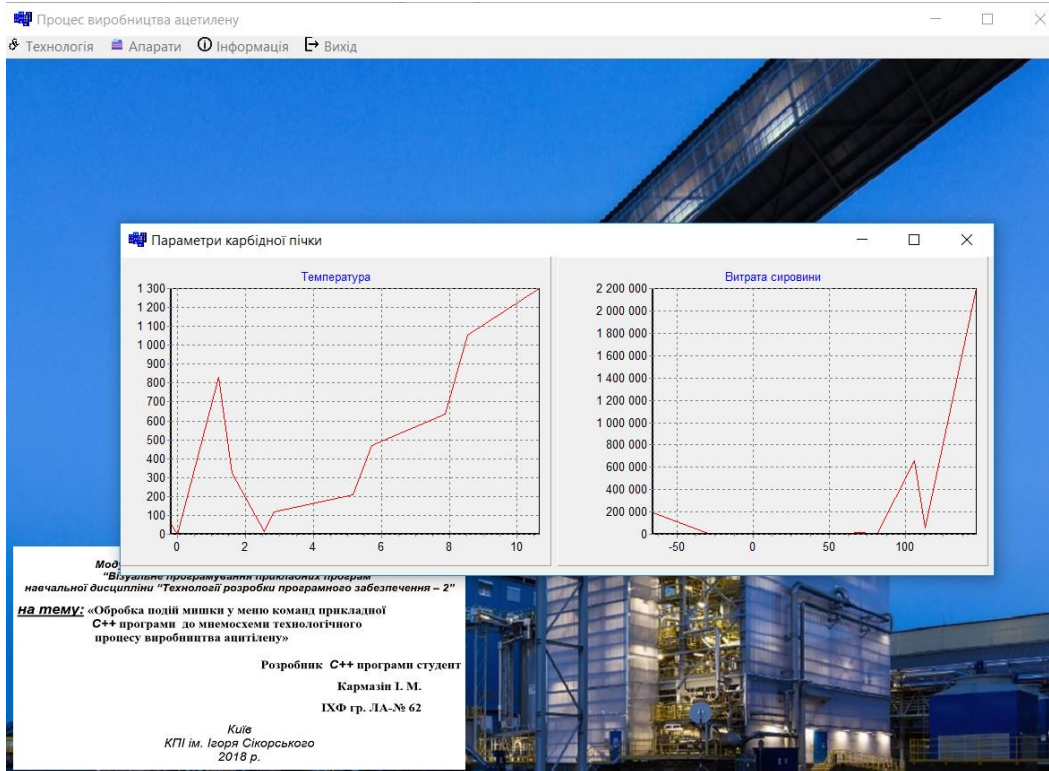


Рис.4.8.Робота команди «Параметри».

						Арк.
Змн.	Арк.	№ докум.	Підпис	Дата	ТРПЗ-МКР2. ТРПЗ.ЛА-62.6212	24