Лабораторна робота №1. Знайомство із середовищем розробки програм Microsoft Visual Studio та складання модульних проєктів програм на C++

Мета: отримати перші навички створення програм для Windows на основі проєктів для Visual C++ з використанням Windows API і навчитися модульному програмуванню на C++

Завдання

1. Створити у середовищі MS Visual Studio C++ проєкт з ім'ям **Lab1**.
2. Написати вихідний текст програми згідно варіанту завдання.
3. Скомпілювати вихідний текст і отримати виконуваний файл програми.
4. Перевірити роботу програми. Налагодити програму.
5. Проаналізувати та прокоментувати результати та вихідний текст програми.

Теоретичні положення

Операційна система (ОС) Windows належіть до класу багатозадачних операційних середовищ, яка керує виконанням багатьох завантажених на виконання програмних застосунків з графічним інтерфейсом користувача (GUI – *graphical user interface*). Для взаємодії з програмними застосунками ця ОС активно використовує систему повідомлень.

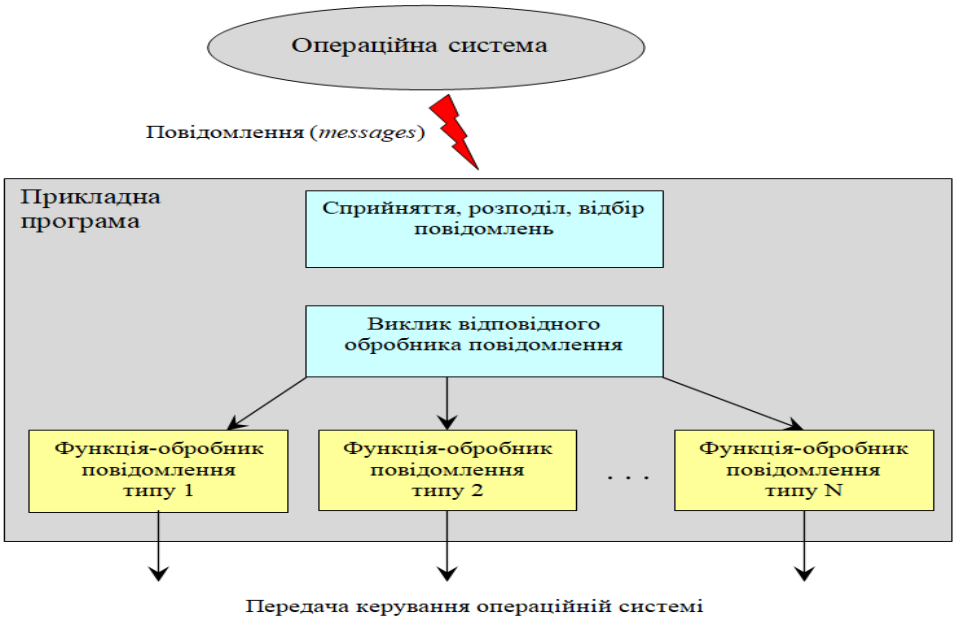

Рис. 1.1. Взаємодія ОС та програмного застосунку

Таким чином, програмний застосунок, керований повідомленнями (*message driven application*) повинен мати у себе програмний код обробників повідомлень.

Повідомлення від операційної системи можуть бути:

- про дії користувача;
- про події у системі;
- про завершення

Що таке повідомлення? В ОС Windows це безпосередній, або взятий з черги повідомлень виклик операційною системою функції вікна прикладної програми (CallBack-функції)

Функціональні можливості прикладної програми визначаються програмним кодом відповідних функцій-обробників повідомлень.

Розроблення програмних застосунків може бути на основі різноманітних бібліотек (фреймворків). Одним з різновидів є написання програмного коду на основі безпосереднього виклику функцій програмного інтерфейсу розробки застосунків операційної системи – API (*application program interface*).

Програмний код програмних застосунків зазвичай описується значним за обсягом вихідним текстом. Для того, щоб зменшити зусилля на розробку та виконати її якомога продуктивніше використовуються підходи щодо структурованості та модульності коду із забезпеченням можливостей повторного використання програмного коду. Конкретні рекомендації щодо цього наведені нижче.

Методичні рекомендації

Інсталяція Microsoft Visual Studio. Для виконання лабораторних робіт можна використовувати середовище розробки Microsoft Visual Studio починаючи з версії 2010. Але краще знайомитися з найсучаснішими версіями, якщо це дозволяють ресурси комп’ютера, який використовується для навчання.

Встановити на власному комп’ютері Microsoft Visual Studio можна з сайту корпорації Microsoft https://visualstudio.microsoft.com/

Створення в середовищі Microsoft Visual Studio нового проєкту C++

Далі розглянемо роботу із середовищем Microsoft Visual Studio на прикладі версії Visual Studio 2019 Community.

Для того, щоб розпочати створення нової програми, потрібно створити проєкт. Для цього на стартовій панелі обрати "Create a new project" (для англомовної версії Visual Studio)

Далі на панелі шаблонів проєктів C++ необхідно вибрати шаблон Windosw Desktop Application – Класичний застосунок Windows. Примітка. Класичним застосунком для операційної системи Windows вважається програмний додаток, який має головне вікно із заголовком та меню, які забезпечують графічний інтерфейс користувача.

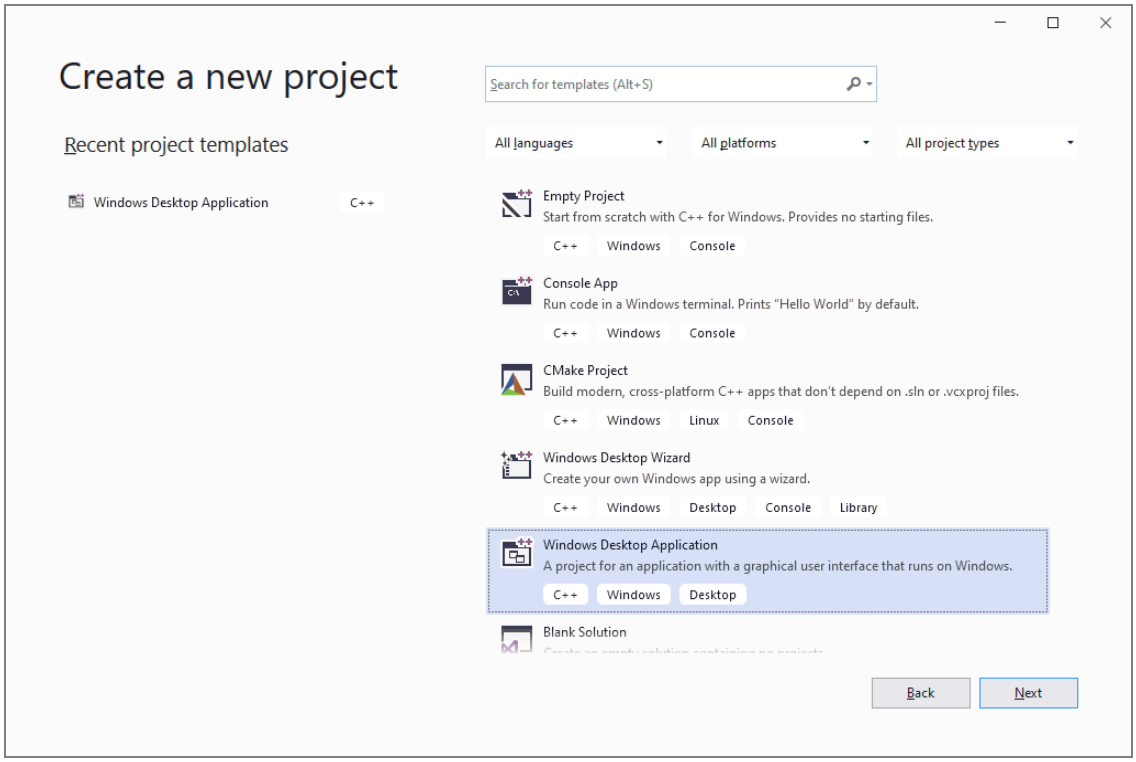

Рис. 1.1. Початок створення проєкту Visual Studio

Далі визначаємо назву проєкту, наприклад, MyProg і вказуємо його розташування на диску. Рекомендація: не записуйте проєкти у папки, які пропонуються за умовчанням (ProgramFiles), створіть для своїх проєктів окрему папку, наприклад, D:\Work.

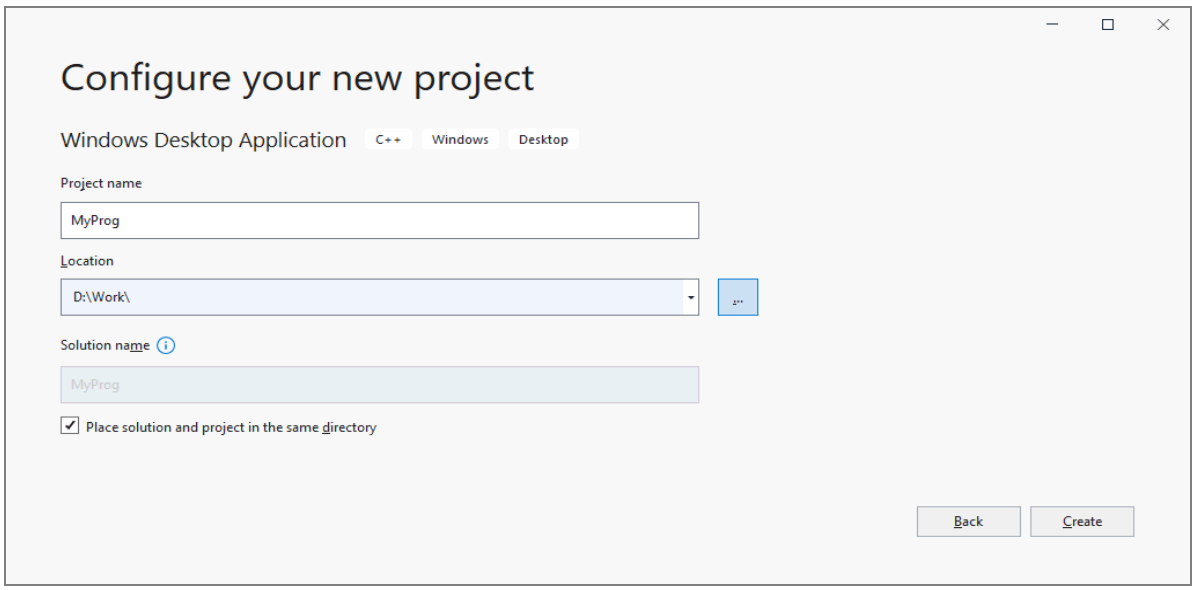

Рис. 1.2. Ім’я нового проєкту та його розташування

Натискуємо кнопку [Create]. Через декілька миттєвостей проєкт буде створений і середовище Visual Studio прийме наступний вигляд

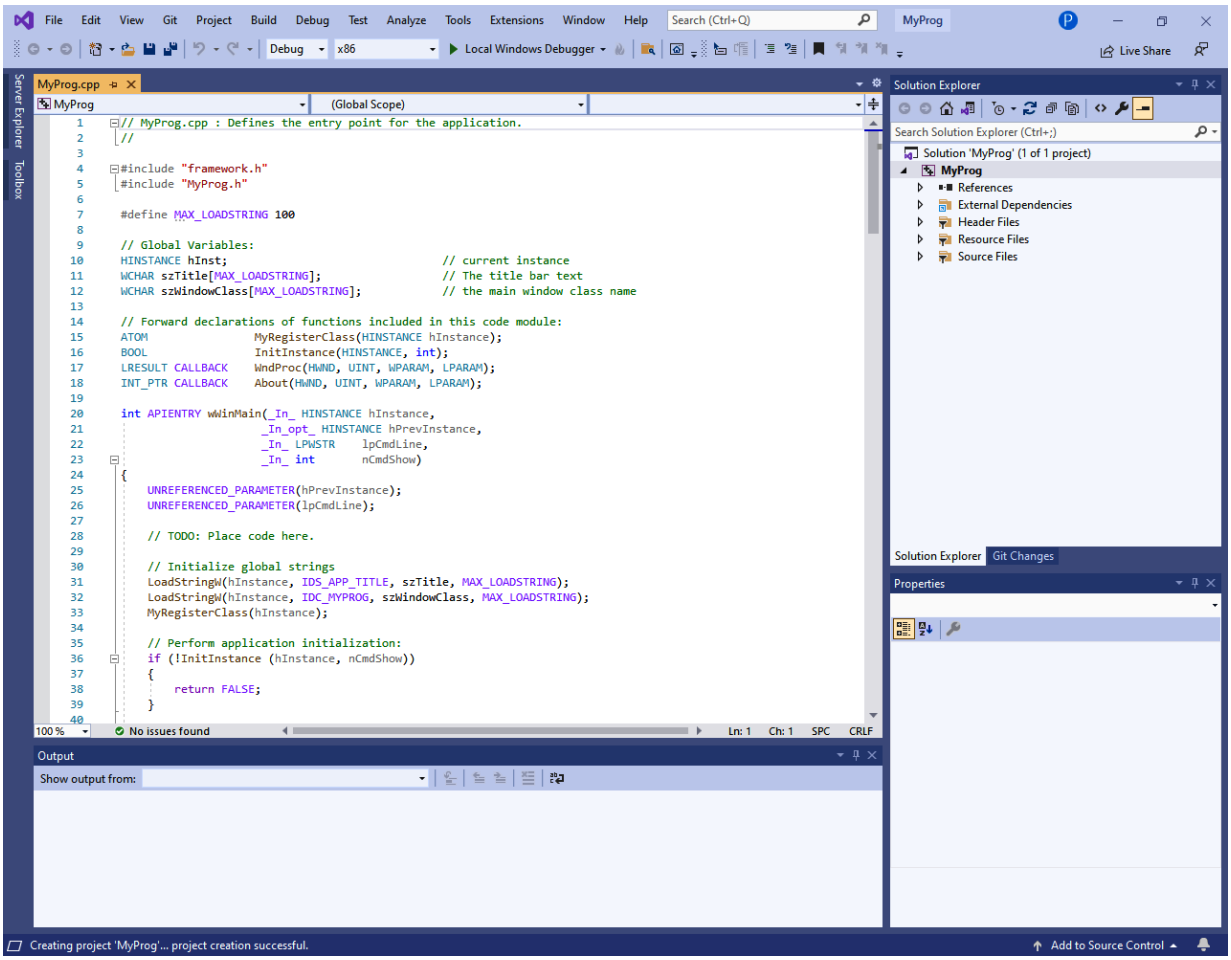

Рис. 1.3. Вигляд середовища розробки програми

У вікні праворуч "Solution Explorer" показуються назви файлів, які включені до проєкту. Головним файлом проєкту будемо вважати MyProg.cpp. Примітка. Якщо проєкт буде з ім’ям Lab1, то головний файл відповідно буде мати ім’я Lab1.cpp.

Тепер можна перевірити проєкт програми, який створюється від початку. Можна скомпілювати проєкт та викликати програму на виконання. Усе це можна зробити одним кроком. Для цього виберіть меню "Debug – Start Debugging" (або просто натиснути клавішу F5, або кнопку у вигляді трикутника). Розпочнеться компіляція, і далі, якщо помилок немає – лінкування. У разі успіху програма почне виконуватися – на екрані на тлі середовища Microsoft Visual Studio з’явиться вікно програми MyProg

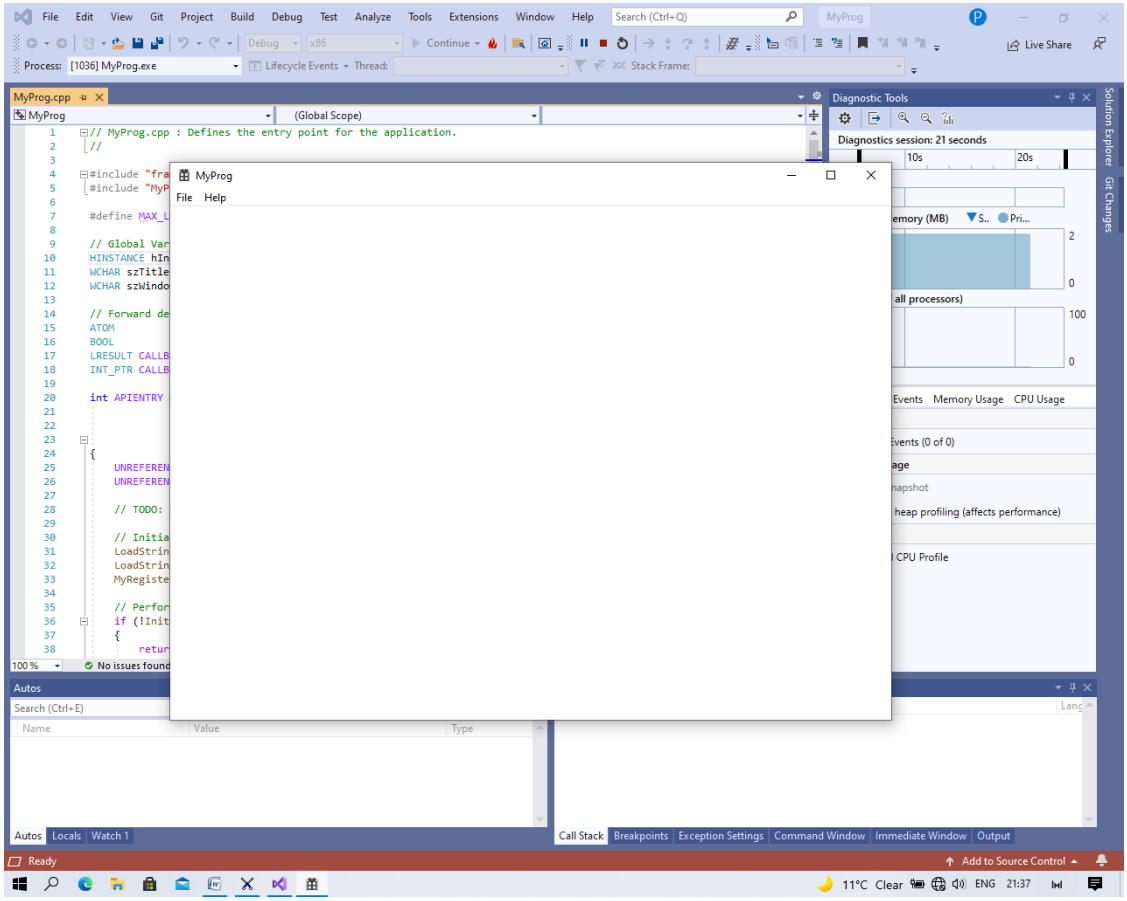

Рис. 1.4. Вигляд вікна програми MyProg на тлі середовища MS Visual Studio

У результаті отримуємо працюючу найпростішу програму, яка була автоматично запрограмована середовищем Visual Studio. Закриємо програму і перейдемо до наступного етапу роботи з проєктом.

Розгляд вихідного тексту програми. Вихідний текст складається з декількох файлів. Головний файл MyProg.cpp містить:

- перша частина – дві директиви #include та оголошення перемінних і функцій
- друга частина – визначення головної функції
- третя частина – Callback-функція головного вікна
- четверта частина – інші функції

Нижче наведені фрагменти тексту файлу MyProg.cpp

Перша частина головного файлу:

```
// MyProg.cpp : Defines the entry point for the application.
//

#include "framework.h"
#include "MyProg.h"

#define MAX_LOADSTRING 100

// Global Variables:
HINSTANCE hInst;                                // current instance
WCHAR szTitle[MAX_LOADSTRING];                  // The title bar text
WCHAR szWindowClass[MAX_LOADSTRING];            // the main window class name

// Forward declarations of functions included in this code module:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
```

Друга частина. Варто звернути увагу на те, що у даному проєкті C++ головна функція зветься не **main** а **_tWinMain**. Ця головна функція є точкою входу і початок роботи програми повністю описується кодом цієї функції. Для будь-якої Windows-програми зазвичай спочатку реєструється клас вікна, потім це вікно створюється і далі програма входить у цикл очікування повідомлень. Подальша робота залежіть вже від того, які повідомлення будуть надходити на адресу головного вікна програми.

```
int APIENTRY wWinMain(_In_ HINSTANCE hInstance,
                      _In_opt_ HINSTANCE hPrevInstance,
                      _In_ LPWSTR    lpCmdLine,
                      _In_ int       nCmdShow)
{
    UNREFERENCED_PARAMETER(hPrevInstance);
    UNREFERENCED_PARAMETER(lpCmdLine);

    // TODO: Place code here.

    // Initialize global strings
    LoadStringW(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
    LoadStringW(hInstance, IDC_MYPROG, szWindowClass, MAX_LOADSTRING);
    MyRegisterClass(hInstance);
```

```
    // Perform application initialization:
    if (!InitInstance (hInstance, nCmdShow))
    {
        return FALSE;
    }

    HACCEL hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_MYPROG));
    MSG msg;

    // Main message loop:
    while (GetMessage(&msg, nullptr, 0, 0))
    {
        if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
        {
            TranslateMessage(&msg);
            DispatchMessage(&msg);
        }
    }

    return (int) msg.wParam;
}
```

Далі записаний текст функцій **MyRegisterClass** та **InitInstance**

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
. . .
    wcex.lpfnWndProc   = WndProc;     //зверніть на це увагу – так передаємо адресу
                                      //функції вікна, щоб потім Windows її викликала
. . .

    return RegisterClassExW(&wcex);
}

BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
. . .
   HWND hWnd = CreateWindowW(szWindowClass, szTitle, WS_OVERLAPPEDWINDOW,
      CW_USEDEFAULT, 0, CW_USEDEFAULT, 0, nullptr, nullptr, hInstance, nullptr);
. . .
   return TRUE;
}
```

Третя частина тексту – визначення Callback-функції головного вікна програми. Коментарі, які автоматично записуються самим Visual Studio при створенні проєкту вельми доречні:

```
//  FUNCTION: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  PURPOSE: Processes messages for the main window.
//
//  WM_COMMAND  - process the application menu
//  WM_PAINT    - Paint the main window
//  WM_DESTROY  - post a quit message and return
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
```

```
    case WM_COMMAND:
        {
            int wmId = LOWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
            case IDM_ABOUT:
                DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                break;
            case IDM_EXIT:
                DestroyWindow(hWnd);
                break;
            default:
                return DefWindowProc(hWnd, message, wParam, lParam);
            }
        }
        break;
    case WM_PAINT:
        {
            PAINTSTRUCT ps;
            HDC hdc = BeginPaint(hWnd, &ps);
            // TODO: Add any drawing code that uses hdc here...
            EndPaint(hWnd, &ps);
        }
        break;
    case WM_DESTROY:
        PostQuitMessage(0);
        break;
    default:
        return DefWindowProc(hWnd, message, wParam, lParam);
    }
    return 0;
}
```

Четверта частина тексту головного файлу – функція вікна ABOUT.

```
// Message handler for about box.
INT_PTR CALLBACK About(HWND hDlg, UINT message, WPARAM wParam, LPARAM lParam)
{
    UNREFERENCED_PARAMETER(lParam);
    switch (message)
    {
    case WM_INITDIALOG:
        return (INT_PTR)TRUE;

    case WM_COMMAND:
        if (LOWORD(wParam) == IDOK || LOWORD(wParam) == IDCANCEL)
        {
            EndDialog(hDlg, LOWORD(wParam));
            return (INT_PTR)TRUE;
        }
        break;
    }
    return (INT_PTR)FALSE;
}
```

Як і належіть функції вікна, вона має чотири аргументи і має модифікатор типу CALLBACK. Це означає, що цю функцію викликає Windows – надсилає повідомлення про події, які стосуються цього вікна. Повідомлення WM_INITDIALOG надсилається при створенні вікна – можна

запрограмувати обробник для ініціалізації елементів вікна (тут цього немає). Повідомлення WM_COMMAND для цього вікна означає реакцію на натискування кнопки ОК, а також закриття вікна. Закривається вікно діалогу після виклику функції EndDialog(hDlg, . . .);

Додавання пунктів меню. Меню є елементом графічного інтерфейсу користувача (GUI) програм. Меню програми описується у файлі ресурсів MyProg.rc наступним чином:

```
IDC_MYPROG MENU
BEGIN
    POPUP "&File"
    BEGIN
        MENUITEM "E&xit",                IDM_EXIT
    END
    POPUP "&Help"
    BEGIN
        MENUITEM "&About ...",           IDM_ABOUT
    END
END
```

Щоб редагувати меню, зокрема, додати пункт меню, можна викликати редактор ресурсів. Для цього треба двічи клікнути у вікні "Solution Explorer" на файлі MyProg.rc

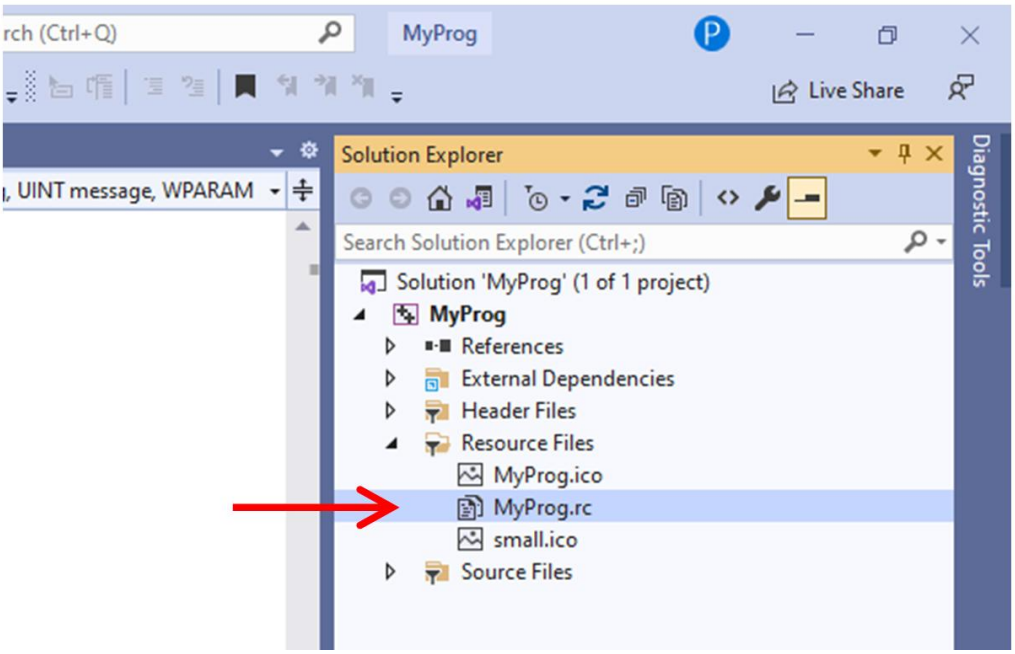

Рис. 1.5. Файл MyProg.rc у списку файлів проєкту

Після кліку на файлі MyProg.rc з'являється редактор ресурсів. У вікні списку ресурсів виберіть Menu і двічі клікніть – відкриється меню для редагування

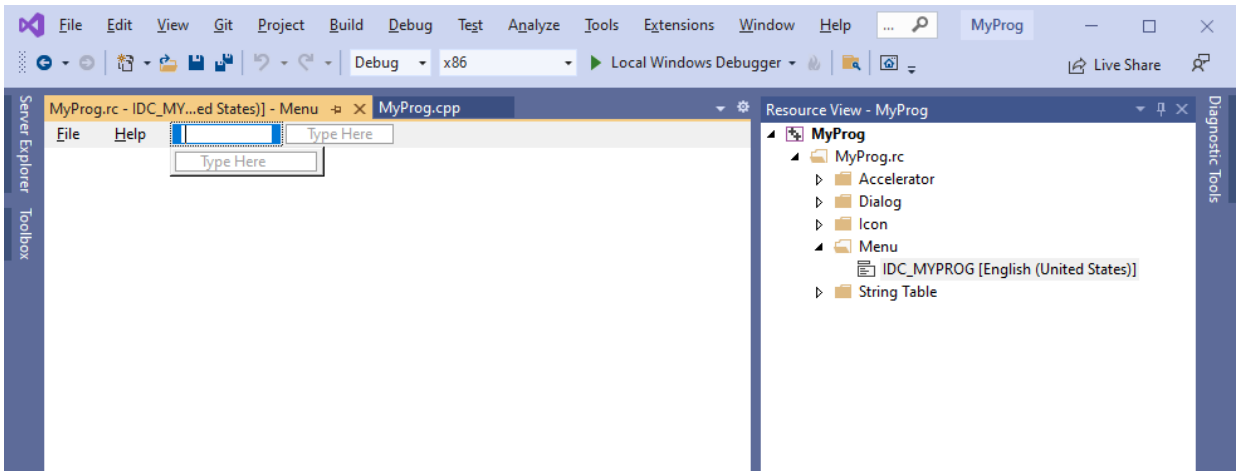

Рис. 1.6. Редактор ресурсів

Введіть назву нової групи пунктів меню (Popup menu), наприклад, "Actions", перетягніть її ліворуч пункту "Help". Далі в групі "Actions" можна внести пункти меню, наприклад, Work1 та Work2.

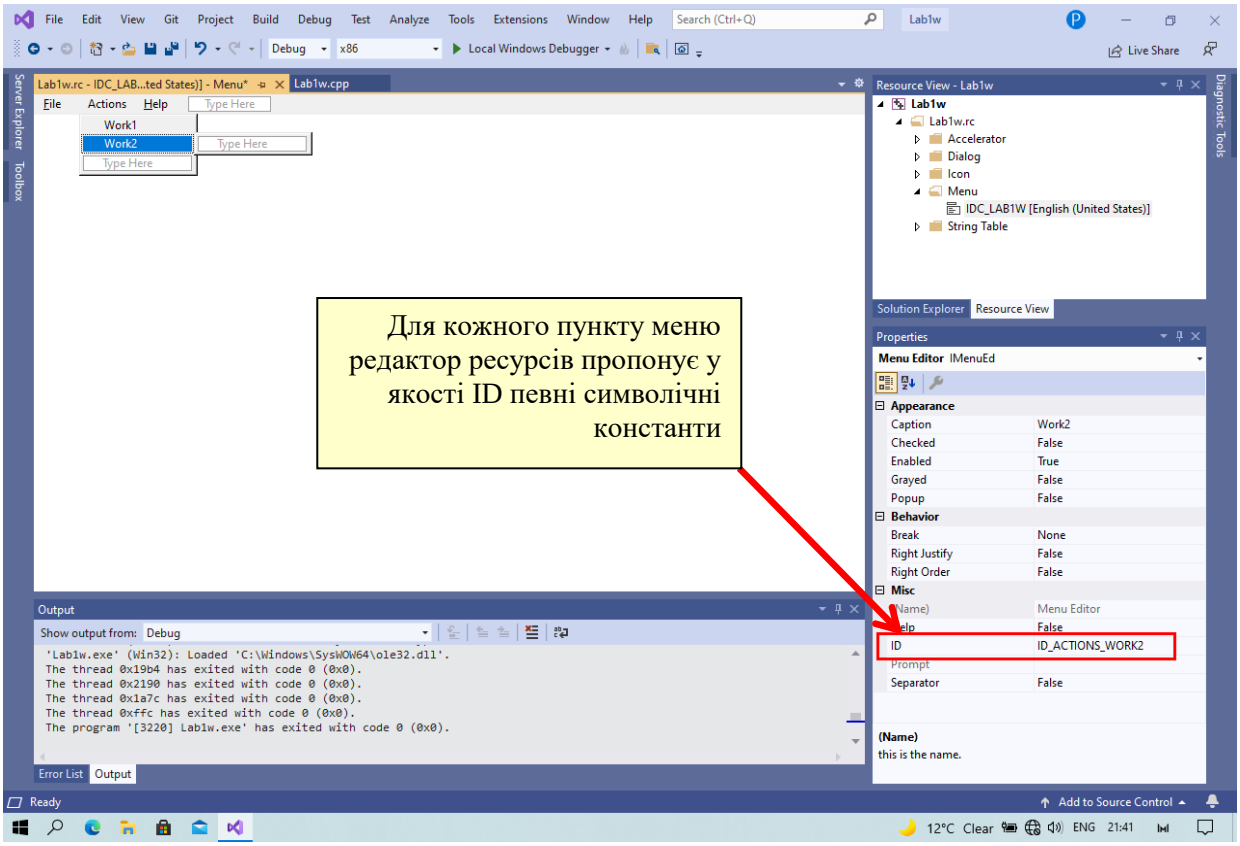

Рис. 1.7. Редактор ресурсів – створення пунктів меню

Символічні константи, які редактор ресурсів пропонує для кожного пункту меню, ми будемо використовувати у “case” обробників повідомлень у функції головного вікна. Визначення таких символічних констант можна

дозволити робити автоматично, але можна їхні імена та числові значення заповнювати власноруч. До відома – ці символічні константи Visual Studio автоматично записує у файл Resource.h.

Після того, як у редакторі ресурсів ми змінили меню далі перевіримо результати такої роботи – перекомпілюємо проєкт. Для цього достатньо натиснути F5. Після успішної компіляції, лінкування та запису exe-файлу наша програма почне виконуватися і у головному вікні програми мають відображатися потрібні пункти меню.

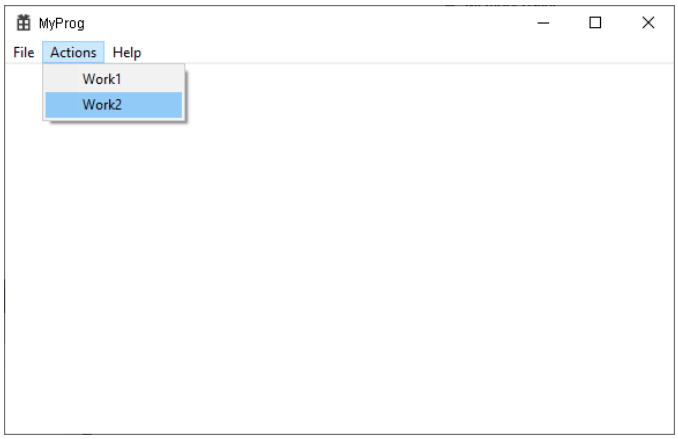

Рис. 1.8. Перевірка наявності нових пунктів меню

Новостворені пункти меню "Work1" або “Work2” можна клікнути, проте нічого не буде. Для того щоб при виборі меню щось виконувалося, потрібно це "щось" запрограмувати. Для цього треба додати у функцію WndProc для обробника повідомлення WM_COMMAND блоки case ID_ACTIONS_WORK1 та case ID_ACTIONS_WORK2:

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
    switch (message)
    {
    case WM_COMMAND:
        {
            int wmId = LOWORD(wParam);
            // Parse the menu selections:
            switch (wmId)
            {
            case ID_ACTIONS_WORK1:
                Work1(hWnd);
                break;
            case ID_ACTIONS_WORK2:
                Work2(hWnd);
                break;
            case IDM_ABOUT:
                DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
                break;
            case IDM_EXIT:
```

```
                DestroyWindow(hWnd);
                break;
```

Далі потрібно оголосити наші нові функції MyWork1 та MyWork2 у верхніх рядках тексту MyProg.cpp в списку оголошень функцій

```
ATOM   MyRegisterClass(HINSTANCE hInstance);
BOOL   InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);

void MyWork1(HWND hWnd);      // оголошення нашших функцій
void MyWork2(HWND hWnd);
```

Далі треба визначити функції MyWork1, MyWork2 – такі визначення найкраще записати наприкінці тексту MyProg.cpp

```
//функція-обробник пункту меню "Work1"
void MyWork1(HWND hWnd)
{
      //потрібний програмний код
}

//функція-обробник пункту меню "Work2"
void MyWork2(HWND hWnd)
{
      //потрібний програмний код робитися
}
```

У фігурних дужках для кожної функції запрограмуйте потрібний код згідно варіанту завдання.

Меню з пунктами у вигляді тексту можна вважати одним з найпростіших для програмістів елементів графічного інтерфейсу користувача. Значно складнішим для початківців може виглядати програмування вікон діалогу.

Як створити нове вікно діалогу? Для цього треба виконати наступні дії:

- у редакторі ресурсів додати нове вікно (Dialog) у файл ресурсів *.rc
- запрограмувати Callback-функцію вікна діалогу
- запрограмувати виклик вікна діалогу у функції MyWork1()

Візьміть до уваги, що у нашій програмі вже є одне вікно діалогу – ABOUT. Скористайтеся його текстом. Виклик вікна діалогу ABOUT робиться наступним чином

```
DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
```

Для нового вікна зробіть так само, проте замість IDD_ABOUTBOX запишіть відповідний ID вікна діалогу, а замість About – ім'я Callback-функції для цього вікна діалогу.

Рекомендації щодо розробки вікон діалогу. Якщо необхідно запрограмувати вікно діалогу у окремому модулі, то спочатку потрібно скласти новий файл ресурсів, наприклад, **module1.rc**. Оскільки це текстовий файл, то теоретично, можливо це зробити у текстовому редакторі. Проте, набагато зручніше це робити у редакторів ресурсів. Як вже вказувалося вище, цей редактор ресурсів автоматично викликається після клику на файлі зі списку файлів ресурсів у вікні Solution Explorer.

У редакторі ресурсів можна розпочати створення нового вікна діалогу, визначити розміри вікна, накласти на поверхню цього вікна елементи діалогу – кнопки, стрічки вводу тексту, списки тощо.

Потім потрібно запрограмувати функції підтримки вікна діалогу у відповідному файлі модуля, наприклад, **module1.cpp**. Зокрема, треба запрограмувати Callback-функцію – обробник повідомлень для цього вікна.

Далі розглянемо деякі варіанти вікон діалогу.

Вікно діалогу зі стрічкою вводу тексту Edit Control. Нехай у редакторі ресурсів ми створили вікно діалогу зі стрічкою вводу тексту. Ця стрічка – елемент Edit Control, може мати ідентифікатор, наприклад, IDC_EDIT1.

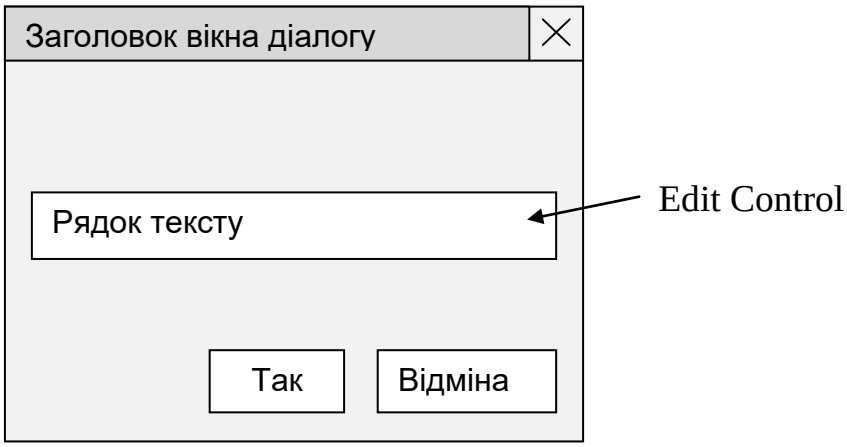

Рис. 1.9. Вікно діалогу зі стрічкою вводу тексту Edit Control

Потрібно запрограмувати, щоб після натискування кнопки Так (Ok) програма вихоплювала текст з цієї стрічки і записувала в якийсь буфер. Потім, після закриття вікна діалогу вміст цього буферу може бути переданий куди завгодно, у тому числі і зовні модуля **module1**.

Щоб вихопити текст з Edit Control можна скористатися функцією GetDlgItemText. Ця функція належіть до складу Windows API. Докладно про неї можна прочитати у довідці.

Фрагмент функції вікна діалогу:

```
//функція вікна діалогу
static BOOL CALLBACK Work1(HWND hDlg,UINT iMessage,WPARAM
wParam,LPARAM)
{
switch (iMessage)
  {
  case WM_COMMAND:
    if (LOWORD(wParam) == IDOK)      //якщо натиснуто кнопку Ok
       {
       //... можливо, щось ще
       //зчитуємо вміст елементу Edit Control у буфер
       GetDlgItemText(hDlg,IDC_EDIT1,буфер,кількість символів);
       //... можливо, щось ще
       EndDialog(hDlg, 1);
       break;
       }
    if (LOWORD(wParam) == IDCANCEL) EndDialog(hDlg, 0);
    break;
  default : break;
  }
return FALSE;
}
```

Вікно діалогу з окремим повзуном скролінгу – Horizontal Scroll Bar. У редакторі ресурсів на поверхню вікна додаємо такі елементи:

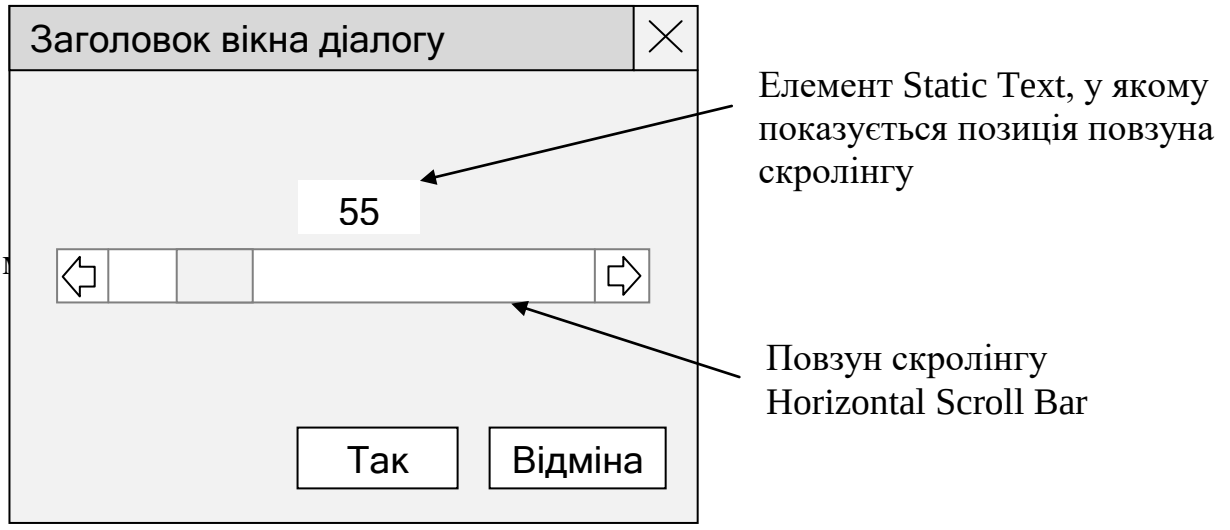

Рис. 1.9. Вікно діалогу з окремим повзуном скролінгу

Над повзуном скролінгу розташовуємо елемент для виводу тексту – Static Text, у який будемо записувати поточну позицію повзуна скролінгу.

Для того, щоб запрограмувати повзун скролінгу, у функції вікна потрібно передбачити обробник повідомлення WM_HSCROLL Спочатку потрібно задати потрібний діапазон значень. Для цього є функція SetScrollRange. Її краще викликати при обробці повідомлення WM_INITDIALOG, яке надсилається нашій програмі одразу після створення вікна діалогу. Щоб узнати поточну позицію повзуна, можна скористатися функцією GetScrollPos

```
pos = GetScrollPos(hWndScroll, SB_CTL);
```

Для того, щоб узнати handle повзуна скролінгу, можна використати функцію GetDlgItem

```
pos = GetScrollPos(GetDlgItem(hDlg, IDC_SCROLLBAR), SB_CTL);
```

Якщо користувач торкається повзуна скролінгу, то Windows надсилає на адресу функції вікна діалогу повідомлення WM_HSCROLL. Кожне повідомлення містить параметри. Для повідомлення WM_HSCROLL у параметр wParam записується код дії а також позиція повзуна. Нижче фрагмент функції вікна діалогу:

```
//Callback-функція вікна
static BOOL CALLBACK Work1(HWND hDlg,UINT iMessage,WPARAM
wParam,LPARAM)
{
switch (iMessage)
  {
  //...
  case WM_HSCROLL:
      pos = GetScrollPos(GetDlgItem(hDlg,IDC_SCROLLBAR),SB_CTL);
      switch (LOWORD(wParam))
        {
        case SB_LINELEFT:       //натиснуто кнопку ліворуч
            pos--;
            break;
        case SB_LINELEFT:       //натиснуто кнопку праворуч
            pos++;
            break;
        case SB_THUMBPOSITION: //фіксована позиція повзуна
        case SB_THUMBTRACK:     //поточна позиція повзуна
            pos = HIWORD(wParam);
            break;
        default : break;
        }
      //... потрібний код
```

```
      SetScrollPos(hWndScroll,SB_CTL,pos,TRUE);  //фіксація
                                                 // позиції повзуна
      //... потрібний код
      break;
  //...
  default : break;
  }
return FALSE;
}
```

Коли користувач рухає повзун скролінгу, або натискує кнопки ліворуч або праворуч, то Windows показує відповідні дії. Але потрібну позицію повзуна треба ще зафіксувати – для цього призначена функція **SetScrollPos**.

Вікно діалогу зі списком ListBox. У редакторі ресурсів на поверхню вікна додаємо такі елементи:

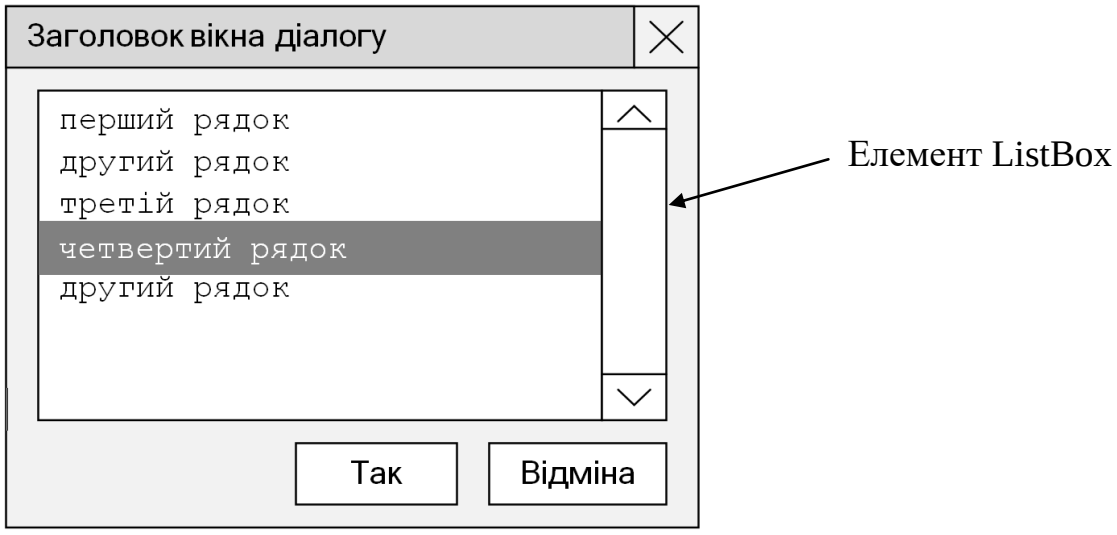

Рис. 1.10. Вікно діалогу зі списком ListBox

Вікно списку (ListBox) містить декілька рядків. Для того, щоб заповнити список потрібними рядками, можна викликати функцію

```
SendDlgItemMessage(hDlg, id, LB_ADDSTRING, 0, (LPARAM)text);
```

У параметр LPARAM записується вказівник на рядок тексту у вигляді або імені буфера тексту, або як символьна константа у подвійних лапках – "символи рядка" Ініціалізувати список, заповнювати його потрібними рядками зручно при обробці повідомлення WM_INITDIALOG, коли вікно діалогу починає активізуватися одразу після створення.

Для читання вибраного елемента списку у буфер можна використати функцію SendDlgItemMessage

```
SendDlgItemMessage(hDlg, id, LB_GETTEXT, indx, (long)buf);
```

Необхідно вказувати індекс потрібного елемента списку. Для того, щоб узнати індекс елемента, який вибрав користувач, можна викликати SendDlgItemMessage з параметром LB_GETCURSEL. Таким чином

```
indx = SendDlgItemMessage(hDlg, id, LB_GETCURSEL, 0, 0);
SendDlgItemMessage(hDlg, id, LB_GETTEXT, indx, (long)buf);
```

Щоб це виконувалося після натискування кнопки "Так", то необхідно запрограмувати відповідний обробник повідомлення WM_COMMAND у функції вікна. Нижче наведені фрагменти функції вікна діалогу, як приклад реалізації списку

```
//Callback-функція вікна діалогу
static BOOL CALLBACK Work1(HWND hDlg,UINT iMessage,WPARAM
wParam,LPARAM)
{
switch (iMessage)
  {
  case WM_INITDIALOG:
      SendDlgItemMessage(hDlg,id,LB_ADDSTRING,0, (LPARAM)str1);
      SendDlgItemMessage(hDlg,id,LB_ADDSTRING,0, (LPARAM)str2);
      SendDlgItemMessage(hDlg,id,LB_ADDSTRING,0, (LPARAM)str3);
      //...
      return (INT_PTR)TRUE;

  case WM_COMMAND:
      if (LOWORD(wParam) == IDOK)
         {
         //... можливо, щось ще
         //зчитуємо вибраний елементу списку у буфер
         indx = SendDlgItemMessage(hDlg, id, LB_GETCURSEL,0,0);
         SendDlgItemMessage(hDlg,id,LB_GETTEXT,indx,(long)buf);
         //... можливо, щось ще
         EndDialog(hDlg, 1);
         break;
         }
      if (LOWORD(wParam) == IDCANCEL) EndDialog(hDlg, 0);
      break;
  default : break;
  }
return FALSE;
}
```

Рекомендації щодо структурованості програмного коду

Навіть для достатньо простої програми вихідний текст може бути записаний занадто громіздким і заплутаним. Одним з факторів складносі сприйняття людиною вихідного тексту може стати текст Callback-функції вікна – як головного, так і вікна діалогу. Якщо вписувати програмний код усіх обробників повідомлень поміж case та break головного switch цієї функції, то це – поганий шлях. Для кожного з обробників можуть знадобитися локальні перемінні, масиви тощо. Функція вікна стає завеликою, буде мати сотні а то й тисячі рядків дещо хаотичного вихідного тексту.

```
//Callback-функція вікна
BOOL CALLBACK WndX(HWND hDlg,UINT iMessage,WPARAM wParam,LPARAM)
{
switch (iMessage)
  {
  case WM_INITDIALOG:
       //... програмний код з багатьох рядків
       return (INT_PTR)TRUE;
  case WM_HSCROLL:
       //... програмний код з багатьох рядків
       break;
  case WM_COMMAND:
       if (LOWORD(wParam) == IDOK)
          {
          //... програмний код з багатьох рядків
          break;
          }
       if (LOWORD(wParam) == IDCANCEL) EndDialog(hDlg, 0);
       break;
  default : break;
  }
return FALSE;
}
```

Приблизно такий самий вигляд має й функція головного вікна, де програмісти-початківці намагаються вписати багато рядків також, зокрема, в обробник повідомлення WM_PAINT.

Порада. Програмний код функції вікна треба розділити, структурувати шляхом оформлення обробника кожного повідомлення у вигляді окремої функції. Нижче наведений скелет структурованого програмного коду функції вікна діалогу

```
//Callback-функція вікна. Структурована версія коду
BOOL CALLBACK WndX(HWND hDlg,UINT iMessage,WPARAM wParam,LPARAM)
{
```

```
switch (iMessage)
  {
  case WM_INITDIALOG:
      return onInit(hDlg);

  case WM_HSCROLL:
      onHScroll(hDlg, wParam);
      break;

  case WM_COMMAND:
      if (LOWORD(wParam) == IDOK) onClickOk(hDlg);
      if (LOWORD(wParam) == IDCANCEL) EndDialog(hDlg, 0);
      break;
  default : break;
  }
return FALSE;
}
```

Тут кожний обробник повідомлення – це окрема функція **onXXX(..).** Програмний код визначення цих функцій міститься в окремій частині вихідного тексту

```
//Визначення функцій-обробників повідомлень
//Обробник повідомлення WM_INITDIALOG
BOOL onInit(HWND hDlg)
{
//... програмний код з багатьох рядків
return TRUE;
}

//Обробник повідомлення WM_HSCROLL
void onHScroll(HWND hDlg, WPARAM wParam)
{
//... програмний код з багатьох рядків
}

//Обробник натискування кнопки Ok
void onClickOk(HWND hDlg)
{
//... програмний код з багатьох рядків
EndDialog(hDlg, 1);
}
```

У подібний спосіб також рекомендується структурувати програмний код Callback-функції головного вікна – у тексті оператора switch для кожного “case“ повідомлень записувати виклик відповідної функції-обробника **onXXX(..).**

Потужним засобом структурованості є модульність, яка розглядається нижче.

Розробка модульного проєкту програми у середовищі Microsoft Visual Studio

Після того, як був створений проєкт C++ Visual Studio та перевірено роботу першої програми, можна продовжувати подальшу розробку. Далі розглянемо наступні теоретичні положення та методичні рекомендації.

Модульне програмування в C++

Загалом, весь вихідний текст програми може бути записаний в один файл *.cpp (за винятком файлу *.rc для проєктів Desktop Application). Таке можна уявити тільки для невеличких простих програм. При вдосконаленні, додаванні нових можливостей вихідний текст, як правило, зростатиме. Починаючи з деякої кількості рядків, у вихідному тексті буде важко розбиратися, аналізувати, робити виправлення.

Модульність є ефективним шляхом розробки складних програм. Вихідний текст розподіляється по модулям, наприклад, відповідно основним функціям. Засоби розробки програм на C++, такі як Visual Studio, дозволяють легко створювати програми з модулів, які можуть окремо компілюватися. Перевагою модульності є зменшення складності об’єктів компіляції, можливість прискорити розробку шляхом командної розробки – кожний модуль може розроблятися та налагоджуватися окремо різними програмістами.

Структура вихідного тексту модуля C++. Вихідний текст кожного модуля, наприклад, з ім’ям module, складається, як мінімум, з двох файлів:

1. Основний файл **module1.cpp**
2. Файл заголовку **module1.h**

Можна сказати, що це традиційно для мови C++

Для проєктів Visual Studio C++ Desktop Application додатково використовуються також й інші файли. Зокрема:

3. Файл **framework.h**
4. Файл ресурсів *.rc, наприклад, **module1.rc** (це необов’язково, тільки у випадку, коли модуль окремо репрезентує якийсь елемент інтерфейсу користувача, наприклад, вікно діалогу тощо)
5. Інші файли

1. Основний файл вихідного тексту модуля (**module1.cpp)** може містити оголошення-визначення перемінних, констант, визначення окремих функцій, визначення функцій-членів класів тощо. Приклад вмісту файлу **module1.cpp**:

```
#include "framework.h"
#include <math.h>
#include "module1.h"

//--оголошення-визначення глобальної перемінної--
int iResult_MOD1  = 0;

//--оголошення внутрішніх перемінних модуля--
static int i,n,size,counter;

//--оголошення внутрішніх функцій модуля—
static INT_PTR CALLBACK DlgProc_MOD1(HWND,UINT,WPARAM,LPARAM);
static void SomeFunc_MOD1(void);


//--визначення функції, яка експортується--
void MyFunc_MOD1(void)
{
. . .
}

//--визначення внутрішньої функції – функції вікна діалогу--
INT_PTR CALLBACK DlgProc_MOD1(HWND hDlg, UINT message,
                              WPARAM wParam, LPARAM lParam)
{
. . .
return 0;
}

//--визначення внутрішньої службової функції--
void SomeFunc_MOD1(void)
{
. . .
}
```

У наведеному вище прикладі перемінна **`iResult_MOD1`** та функція **`MyFunc_MOD1()`** будуть використовуватися в інших модулях. Це буде інтерфейс модуля. Крім того, у модулі є члени – перемінні та функції, які будуть використовуватися тільки для внутрішніх потреб модуля. Вони зовні невидимі, їхні імена недоступні для інших модулів програми – для цього вони оголошенні зі словом **`static`** попереду.

Файл module1.cpp потрібно включити у проєкт – він повинен бути у списку файлів Source Files вікна Solution Explorer. Тільки тоді цей файл буде компілюватися.

Примітка. У проєктах Visual Studio C++ Desktop Application на початку тексту файлів *.cpp повинен бути рядок

```
#include "framework.h"
```

2. Файл заголовку модуля (**module1.h)** містить оголошення елементів, які будуть загальнодоступними – для використання у інших модулях програми. Іншими словами, описують інтерфейс модуля. Такі елементи позначаються словом **extern**.

Приклад вмісту файлу **module1.h**

```
#ifndef _MODULE1_H_
#define _MODULE1_H_

#define STATUS_ERROR_MOD1 -1
#define STATUS_GOOD_MOD1 1

extern int iResult_MOD1;

extern void MyFunc_MOD1(void);

#endif
```

або

```
#pragma once

#define STATUS_ERROR_MOD1 -1
#define STATUS_GOOD_MOD1 1

extern int iResult_MOD1;

extern void MyFunc_MOD1(void);
```

Для того, щоб при включенні заголовочного файлу директивою **#include** уникнути помилок компіляції повторного визначення, зокрема символічних констант **STATUS_ERROR_MOD1** та **STATUS_GOOD_MOD1,** у файлі записано рядки:

```
#ifndef _MODULE1_H_
#define _MODULE1_H_
...

#endif
```

або

```
#pragma once

...
```

Рекомендується подібним чином оформлювати усі заголовочні файли *.h.

Файл module1.h включати у проєкт не обов'язково (він буде автоматично завантажуватися директивою **#include**), проте бажано – тоді він буде у списку файлів Header Files вікна Solution Explorer і його буде легко викликати для огляду.

Рекомендація. Щоб не заплутатися у іменах ідентифікаторів і не використовувати щось на кшалт **namespace**, рекомендується імена **extern** членів модулів робити унікальним. Наприклад, до кожного імені члена

модуля **module1** додавати наприкінці "_MOD1", для членів модуля **module2** додавати наприкінці "_MOD2" і так далі.

3. Файл ресурсів (**module1.rc)** містить опис елементів графічного інтерфейсу, наприклад, меню, вікон діалогу, та інших ресурсів. Структура файлу **module1.rc**:

```
#include "framework.h"
 . . .

IDD_MYDIALOG_MOD1 DIALOGEX . . .   //вікно діалогу
. . .  //опис елементів вікна діалогу

//інші ресурси - меню, курсори, іконки, рядки тексту тощо
```

Файл module1.rc потрібно включити у проєкт – він повинен бути у списку файлів Resource Files вікна Solution Explorer.

Примітка. Ідентифікатори елементів ресурсів, наприклад ід. вікон діалогу, є константами. Важливо те, що значення таких ід. повинні бути унікальними – не може співпадати з ід. інших відповідних ресурсів проєкту. Краще робити ід. у вигляді символічних констант, на кшалт **IDD_MYDIALOG_MOD1**, проте, тоді їх потрібно десь визначати. Визначення таких символічних констант можна зробити у окремому файлі *.h, а можливо, у файлі з іншим розширенням. Наприклад, якщо визначення **IDD_MYDIALOG_MOD1** зробити у файлі **module1.rh**

```
#pragma once

#define IDD_MYDIALOG_MOD1 1001
```

тоді у файлі **module1.rc** повинно бути #include "module1.rh"

```
#include "framework.h"
#include "module1.rh"

 . . .

IDD_MYDIALOG_MOD1 DIALOGEX . . .
. . .
```

а якщо у файлі **module1.cpp** також використовується ім’я **IDD_MYDIALOG_MOD1**, то у цьому файлі також повинно бути #include "module1.rh":

```
#include "framework.h"
#include <math.h>
#include "module1.h"
#include "module1.rh"
```

Як додати модуль в проєкт? Будемо вважати, що тексту модуля поки що немає і ми плануємо його писати безпосередньо у середовищі Visual Studio. Для того, щоб додати файл *.cpp модуля, вкажіть курсором розділ "Source Files", клацніть правою кнопкою миші, і потім – ”Add" ”New Item"

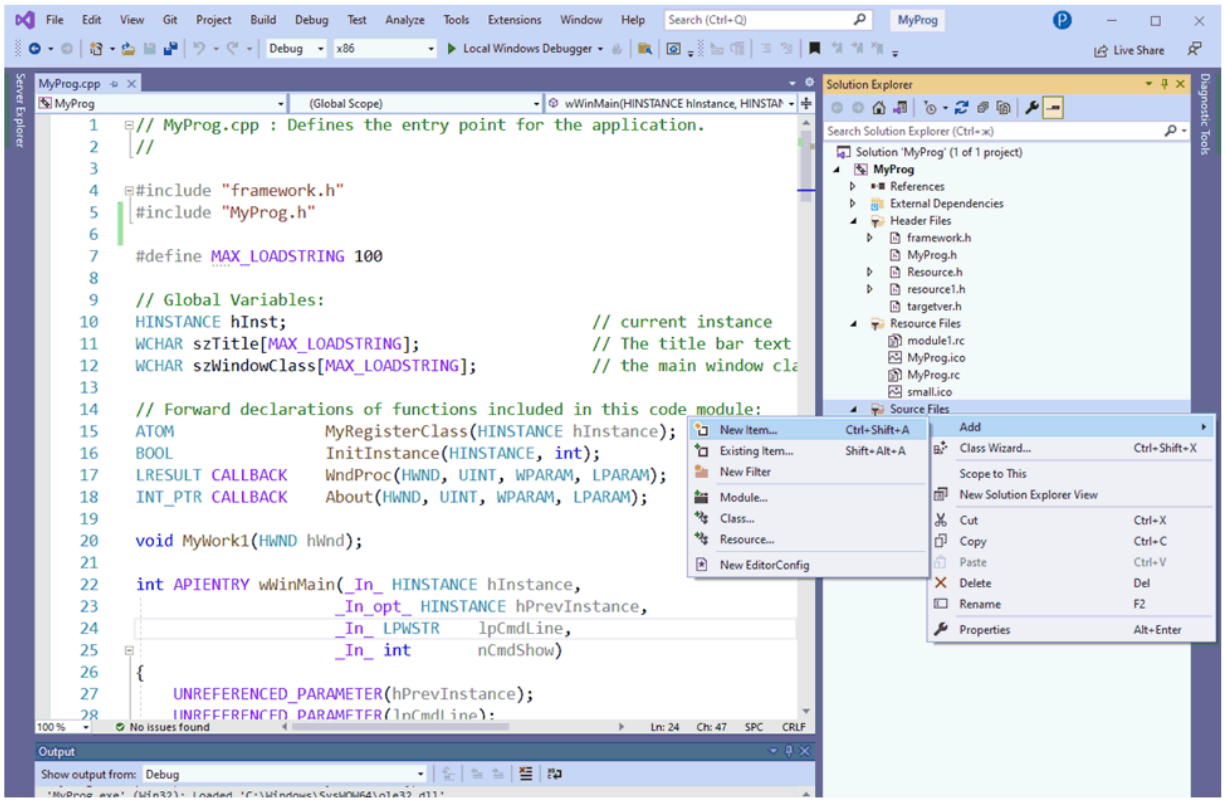

Рис. 1.11. Додавання файлу модуля у проєкт

Поняття #include – ієрархії файлів

Схема **#include – ієрархії файлів** описує відношення включення файлів проєкту директивами #include. У такій схемі вказуються файли, які містять директиви #include, а також файли, які включаються цими директивами.

На найнижчому рівні ієрархії розташовуються файли, які не містять #include. Наступний рівень посідають файли, у текстах яких записані #include файлів найнижчого рівня. І так далі. Така схема ілюструє залежності елементів проєкту (рис. 1.12).

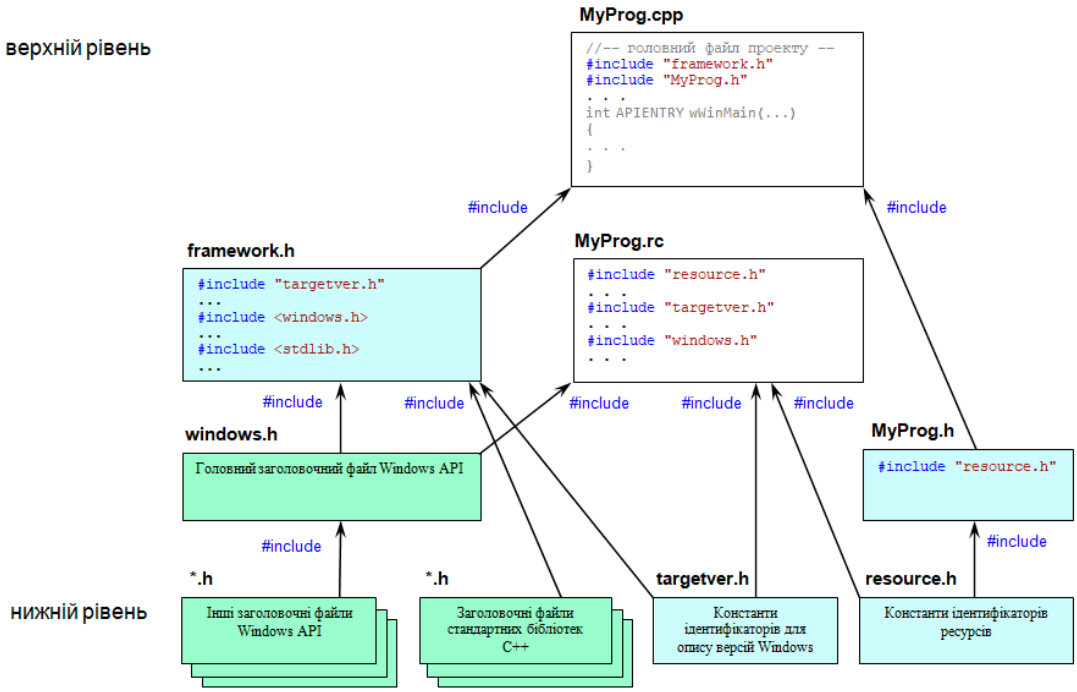

Рис. 1.12.Приклад схеми #include-ієрархії файлів проєкту

Поняття залежності модулів програми

Початковий проєкт C++, який автоматично створює Microsoft Visual Studio, не є модульним у контексті даної лабораторної роботи. І це незважаючи на те, що проєкт містить багато пов'язаних між собою файлів, як наведено вище. Тепер уявимо, що у проєкті є три модуля та головний файл програми. Відобразимо залежності модулів відповідно #include наступним чином:

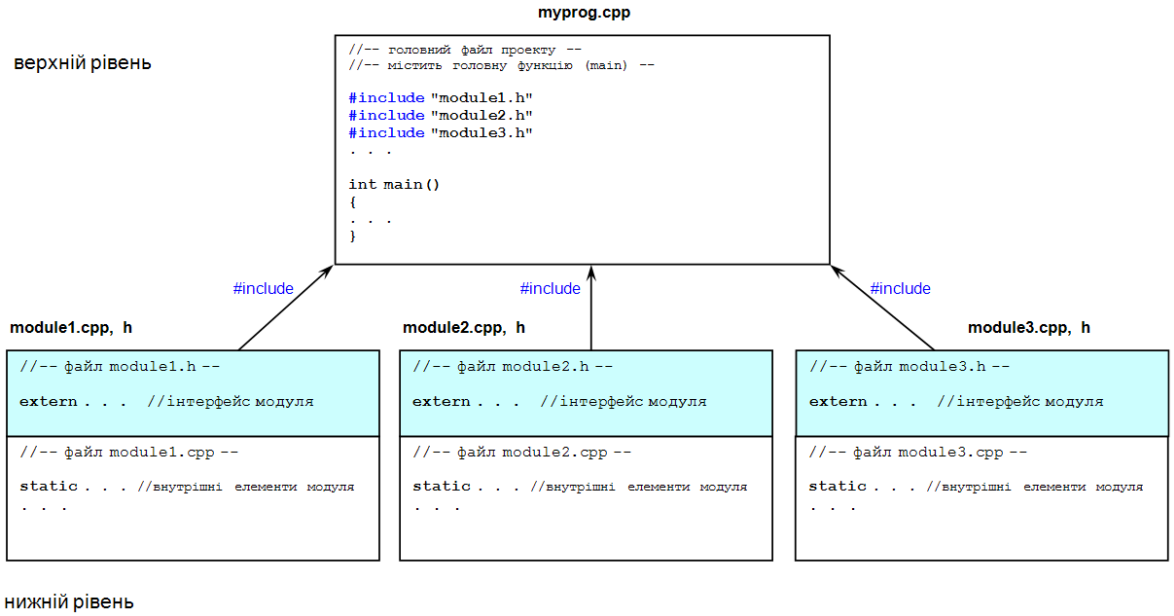

Рис. 1.13.Приклад схеми залежностей модулів

З цієї схеми видно, що три модуля – **module1, module2** та **module3** є незалежними один від одного. Елементи інтерфейсу кожного з модулів використовуються у головному файлі програми **myprog.cpp**. У якості елементів інтерфейсу зазвичай записуються функції – вони оперують схованими у модулях функціями та даними. Інтерфейсні функції призначені для виклику у інших модулях. Таким чином, у даному проєкті головний файл залежіть від вказаних модулів. Модулі можуть роздільно компілюватися – відповідно створюються об'єктні файли **module1.obj, module2.obj** та **module3.obj.**

У лабораторній роботі потрібно розробити декілька модулів, врахувавши специфіку програмування на основі Windows API.

Рекомендації щодо модуля, який показує вікно діалогу

Інтерфейс модуля (описується у файлі ***.h**) потрібно зробити у вигляді єдиної функції, наприклад

```
extern int Func_MOD1(HINSTANCE hInst, HWND hWnd);
```

Згідно завдань нашої лабораторної роботи, ця функція буде викликати єдине вікно діалогу, створення та підтримка якого буде запрограмоване у файлі ***.cpp** модуля. Наприклад, для першого модуля у файлі **module1.cpp** можна так:

```
//функція-оболонка створення вікна діалогу
int Func_MOD1(HINSTANCE hInst, HWND hWnd)
{
return DialogBox(hInst, MAKEINTRESOURCE(IDD_DIALOG1), hWnd, Work1);
}

//Callback-функція вікна
static BOOL CALLBACK Work1(HWND hDlg,UINT iMessage,WPARAM
wParam,LPARAM)
{
switch (iMessage)
  {
  case WM_INITDIALOG:
      //. . . ініціалізуємо елементи вікна діалогу (якщо потрібно)
      break;
  case WM_COMMAND:
      if (LOWORD(wParam) == IDOK)
         {
         //. . . зчитуємо вміст елеменітів вікна (якщо потрібно)
         EndDialog(hDlg, 1);
         break;
         }
```

```
      if (LOWORD(wParam) == IDCANCEL) EndDialog(hDlg, 0);
      break;
  default : break;
  }
return FALSE;
}
```

Функція **DialogBox** повертає значення, яке вказується другим параметром функції **EndDialog.** Наприклад, для того, щоб результат роботи **Func_MOD1** при натискуванні кнопки Cancel був 0, то відповідно у функції вікна записується **EndDialog(hDlg, 0)**.

Де взяти константу **IDD_DIALOG1**? Її можна визначити, наприклад, так

```
#define IDD_DIALOG1 1301
```

і записати це у окремому файлі **module1.rh**. Потім посилання на цей файл записати у файлах **module1.cpp** та **module1.rc** директивою #uinclude

```
#include "module1.rh"
```

Для того, щоб в інтерфейсній функції модуля здихатися параметра **hInst**, який потрібен для DialogBox, можна скористатися функцією **GetWindowLongPtr( )**, яка при параметрі GWLP_HINSTANCE повертає значення HINSTANCE.

Таким чином, визначення функції **Func_MOD1** може бути таким:

```
int Func_MOD1(HWND hWnd)
{
return DialogBox(GetWindowLongPtr(hWnd, GWLP_HINSTANCE),
                 MAKEINTRESOURCE(IDD_DIALOG1),
                 hWnd,
                 Work1);
}
```

Можна уявити собі, що кожний модуль, який реалізує якесь вікно діалогу, буде запрограмований вже чотирма файлами: ***.cpp**, ***.h**, ***.rc**, ***.rh**. Тоді загальна структура проєкту, який складатиметься з двох модулів, буде такою:

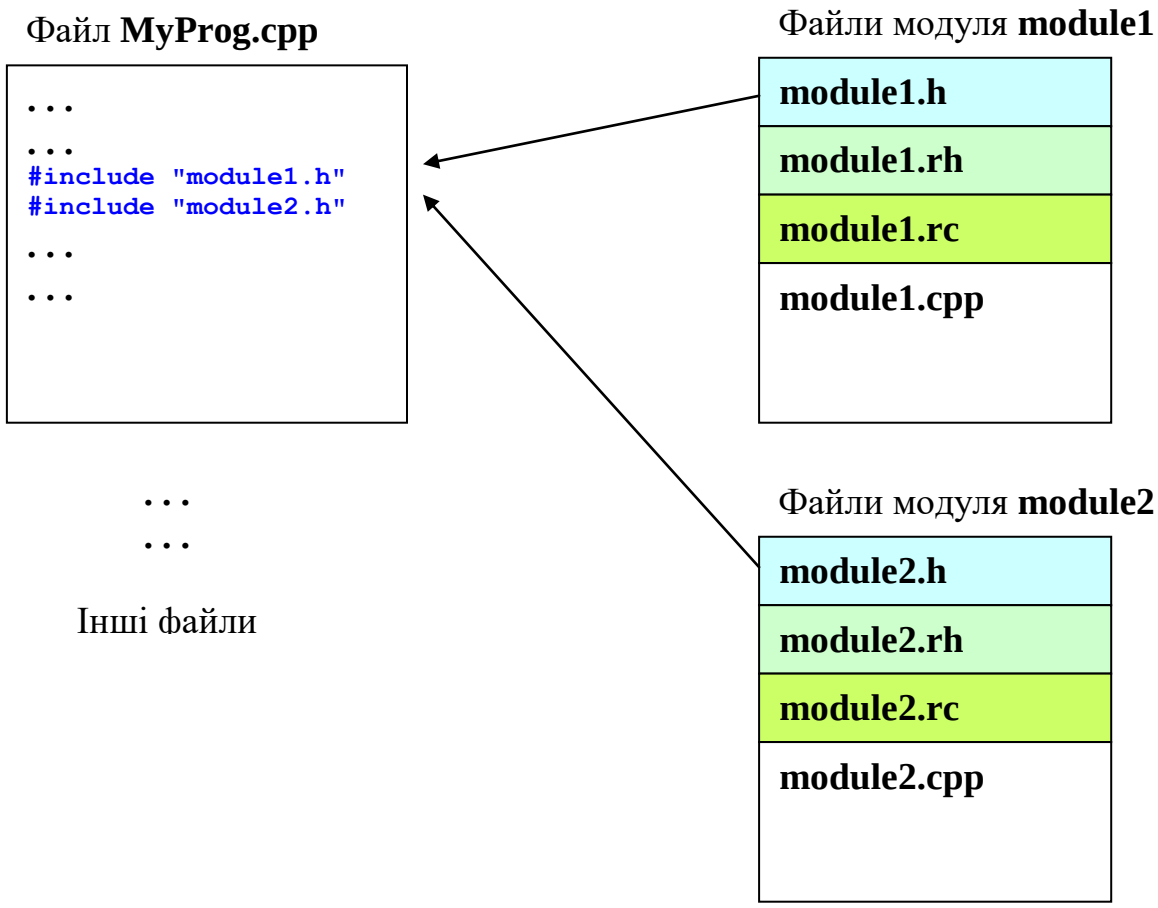

Рис. 1.14. Структура проєкту з двох модулів

Рекомендації щодо джерел інформації

Найкраще джерело по розробці програм на основі інструментів Microsoft – довідкова служба Microsoft. Знайти її можна за посиланням:
https://msdn.microsoft.com
або
https://docs.microsoft.com/en-us/

Ця довідкова служба також викликається з середовища Visual Studio або через Internet, або може бути сформована як локальний ресурс

Тематично інформацію, яка стосується даної роботи, можна оглянути у розділі:
https://developer/microsoft.com/ru-ru/windows/desktop/develop
та інших.

Варіанти завдань та основні вимоги

Для усіх варіантів необхідно створити **два пункти меню** – "Робота1" та "Робота2".

1. Потрібно, щоб при виборі пункту меню "Робота1" виконувалося щось згідно варіанту B_1, причому B_1 обчислюється за формулою

$$B_1 = Ж \bmod 4,$$

де Ж – номер студента у журналі, mod – залишок від ділення

2. Запрограмувати також, щоб при виборі пункту меню "Робота2" виконувалося щось згідно варіанту B_2:

$$B_2 = (Ж+1) \bmod 4$$

3. Вихідний текст функції B_1 запрограмувати в окремому модулі, а вихідний текст для функції B_2 – у іншому окремому модулі. Таким чином, проєкт повинен складатися з головного файлу Lab2.cpp та ще, як мінімум, двох файлів модулів з іменами module1.cpp та module2.cpp. Примітка: **кожне вікно діалогу повинно бути в окремому незалежному модулі**, тому якщо $B_1 = 2$, або $B_2 = 2$ то має бути, як мінімум, три модуля.

4. **Callback-функції вікон діалогу не мають бути видимими зовні модулів** – це внутрішні подробиці модулів, треба сховати їх, оголошуючи словом **static**. Також не можна виносити за межі модулів ресурси – опис вікон діалогу. Це забезпечується тим, що для кожного модуля **moduleX.cpp** має бути власний файл ресурсів **moduleX.rc**.

5. Інтерфейс модулів. Інтерфейс кожного модуля, наприклад, модуля **module1.cpp**, записується у відповідний файл заголовку **module1.h**. Інтерфейс модуля повинен бути у вигляді тільки одної функції

```
extern int Func_MOD1(HWND hWnd, . . . );
```

Ця функція має, як мінімум, один аргумент, через який у модуль буде передаватися handle головного вікна. Можуть бути ще інші аргументи – за необхідності. Інтерфейсна функція повинна повертати 0, якщо користувач натиснув у вікні діалогу кнопку "Відміна" ("Cancel"), або "×" у правому верхньому куті вікна. **Вимога: інших функції, перемінних у інтерфейсі модуля (у файлі .h) не повинно бути.**

6. Якщо виводиться наступне вікно діалогу, то попереднє вікно повинно закриватися

7. Намалювати повну #include-ієрархію усіх наявних файлів проєкту (враховувати тільки ті файли, які відображаються у вікні Solution Explorer). Неприпустимі перехресні #include-зв'язки між модулями.

8. Варіанти В1 та В2 вибираються згідно таблиці 1.1

Таблиця 1.1

Варіант	Що треба зробити	Підказка
0	Вікно діалогу для вводу тексту, яке має стрічку вводу (Edit Control) та дві кнопки: [Так] і [Відміна]. Якщо ввести рядок тексту і натиснути [Так], то у головному вікні повинен відображатися текст, що був введений.	Взяти від стрічки вводу рядок тексту – функція **GetDlgItemText** Вивести текст у головному вікні – функція **TextOut** в обробнику повідомлення WM_PAINT
1	Вікно діалогу з повзуном горизонтального скролінгу (Horizontal scroll Bar) та дві кнопки: [Так] і [Відміна]. Рухаючи повзунок скролінгу користувач вводить число у діапазоні від 1 до 100. Після натискування кнопки [Так] вибране число буде відображатися у головному вікні.	Обробка руху повзуна скролінгу – обробка повідомлення **WM_HSCROLL** у Callback-функції вікна діалогу. Вивести число у головному вікні – функція **TextOut** в обробнику повідомлення WM_PAINT. Щоб число вивести як текст – функція **itoa** або **ltoa** або інша функція, яка перетворює ціле число у рядок тексту.
2	Два вікна діалогу. Спочатку з'являється перше, яке має дві кнопки: [Далі >] і [Відміна]. Якщо натиснути кнопку [Далі >], то воно закриється і з'явиться друге длг вікно, яке має кнопки: [< Назад], [Так] і [Відміна]. Якщо натиснути кнопку [<Назад], вікно закриється і перехід до першого вікна.	Аналізувати результат роботи длг вікна можна по значенню, яке повертає DialogBox. Значення визначається другим аргументом функції EndDialog, яка розташовується у функції вікна. Можна, наприклад: - для кнопки [Далі >] робити виклик функції EndDialog(hDlg, 1) - для кнопки [Відміна] – EndDialog(hDlg, 0) - для кнопки [< Назад] – EndDialog(hDlg, -1)
3	Вікно діалогу з елементом списку (List Box) та двома кнопками: [Так] і [Відміна]. У список автоматично записуються назви груп нашого факультету. Якщо вибрати потрібний рядок списку і натиснути [Так], то у головному вікні повинен відображатися текст вибраного рядка списку.	Занесення рядка у список: **SendDlgItemMessage**(hDlg, id, LB_ADDSTRING, 0, (LPARAM)text); Читання елементу списку. Спочатку індекс рядка, що вибрано: indx = **SendDlgItemMessage**(hDlg, id, LB_GETCURSEL, 0, 0); а потім взяття indx-го рядка тексту: **SendDlgItemMessage**(hDlg, id, LB_GETTEXT, indx, (long)buf); Вивести текст у головному вікні – функцією **TextOut** в обробнику для WM_PAINT

Примітка. Незважаючі на однакові для деяких студентів варіанти завдань, їхні рішення повинні бути індивідуальними.

Зміст звіту

1. Титульний аркуш
2. Варіант завдання
3. Вихідний текст головного файлу .cpp (фрагменти, що ілюструють власний код),
4. та вихідні тексти модулів – файли .cpp, .h.
5. Ілюстрації (скріншоти)
6. Висновки

Контрольні запитання

1. Що виконує головна функція програми?
2. Чим відрізняється реєстрація вікна від його створення?
3. Що робить у Callback-функції вікна оператор switch?
4. Як створюється новий пункт меню?
5. Як викликається вікно діалогу?
6. Як створити нове вікно діалогу?
7. Де запрограмований цикл очікування повідомлень головного вікна?
8. Що таке WM_COMMAND?
9. Які файли потрібні для проєкту Visual Studio Desktop Application?
10. Що таке #include – ієрархія?
11. Що робить директива #include?
12. Чим відрізняється Callback-функція головного вікна від функції вікна діалогу?
13. Що означає роздільна компіляція модулів і як її досягти?