Лабораторна робота №1

Знайомство із середовищем розробки програм Microsoft Visual Studio та складання модульних проектів програм на C++

Мета: отримати перші навички створення програм для Windows на основі проектів API Win32 для Visual C++ і навчитися модульному програмуванню на C++

Завдання:

1. Створити у середовищі MS Visual Studio C++ проект Win32 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.

Порядок виконання роботи та методичні рекомендації

Частина 1. Знайомство із середовищем розробки програм Microsoft Visual Studio

Інсталяція Microsoft Visual Studio

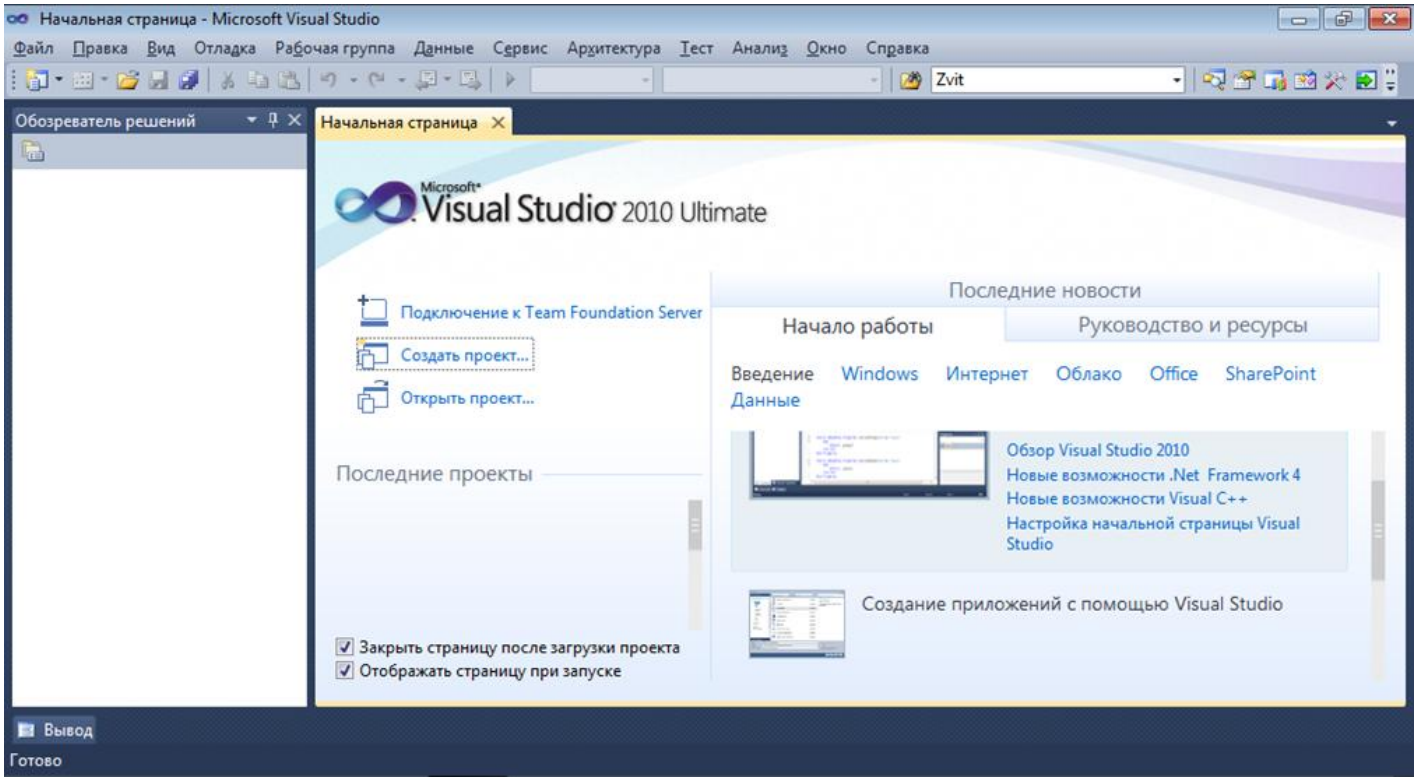
Для виконання лабораторних робіт можна використовувати середовище розробки Microsoft Visual Studio починаючи з версії 2010 і більш сучасних включно. У випадку використання старих версій, наприклад, 2010-ї, треба звернути увагу на те, що найпростіша версія – Express нам не згодиться, оскільки вона суттєво обмежена, зокрема, відсутній редактор ресурсів для проектів Win32 C++.

Встановити на власному комп'ютері середовище Microsoft Visual Studio можна, завантаживши необхідні інсталяційні файли з сайту корпорації Microsoft microsoft.com, або з інших джерел.

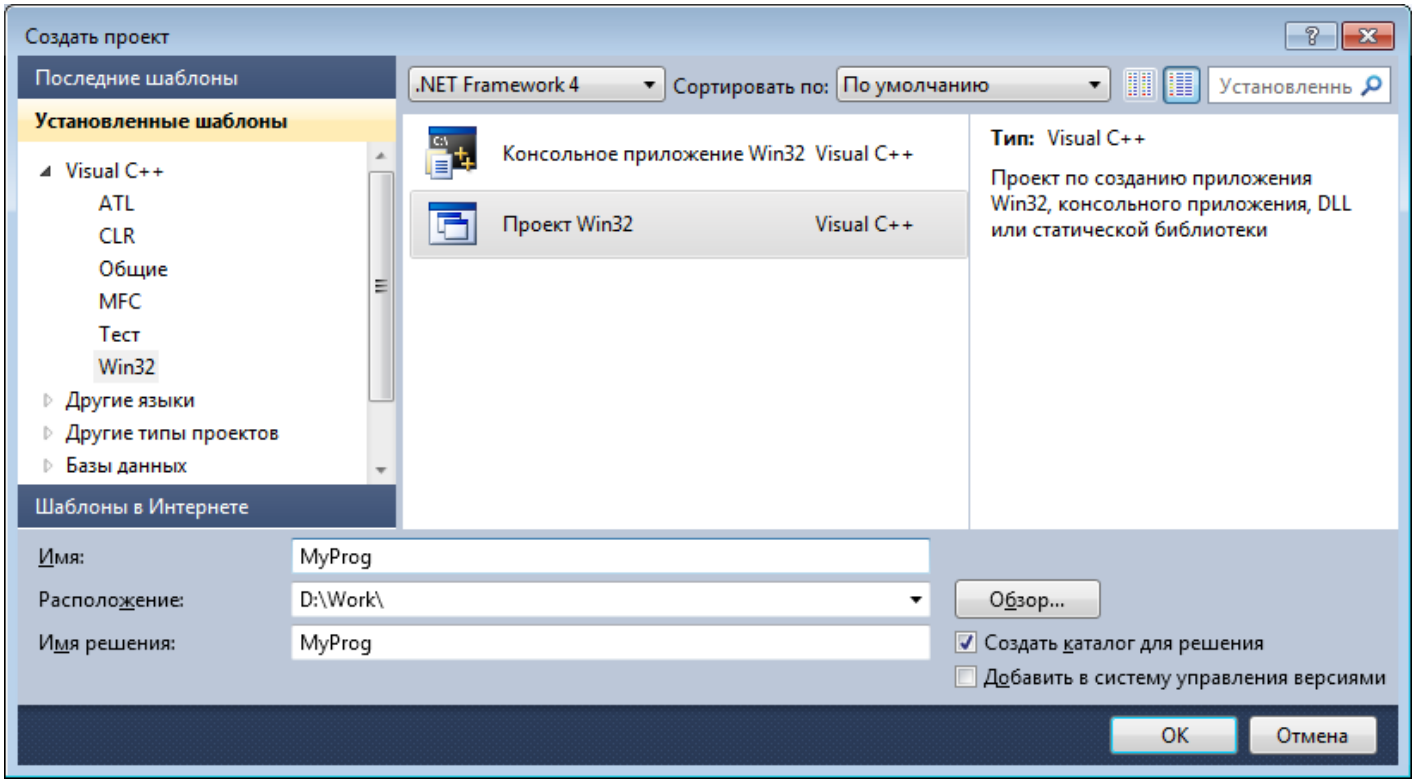
Створення в середовищі Microsoft Visual Studio нового проекту C++ Win32

Далі рощглянемо роботу із середовищем Microsoft Visual Studio на прикладі його російськомовної версії 2010.

Для того, щоб розпочати створення нової програми, потрібно створити проект. Створити проект можна, як мінімум, двома способами: через меню "Файл-Создать-Проект", або безпосередньо на початковій сторінці вибрати пункт "Создать проект . . ." (для російськомовних версій Visual Studio)

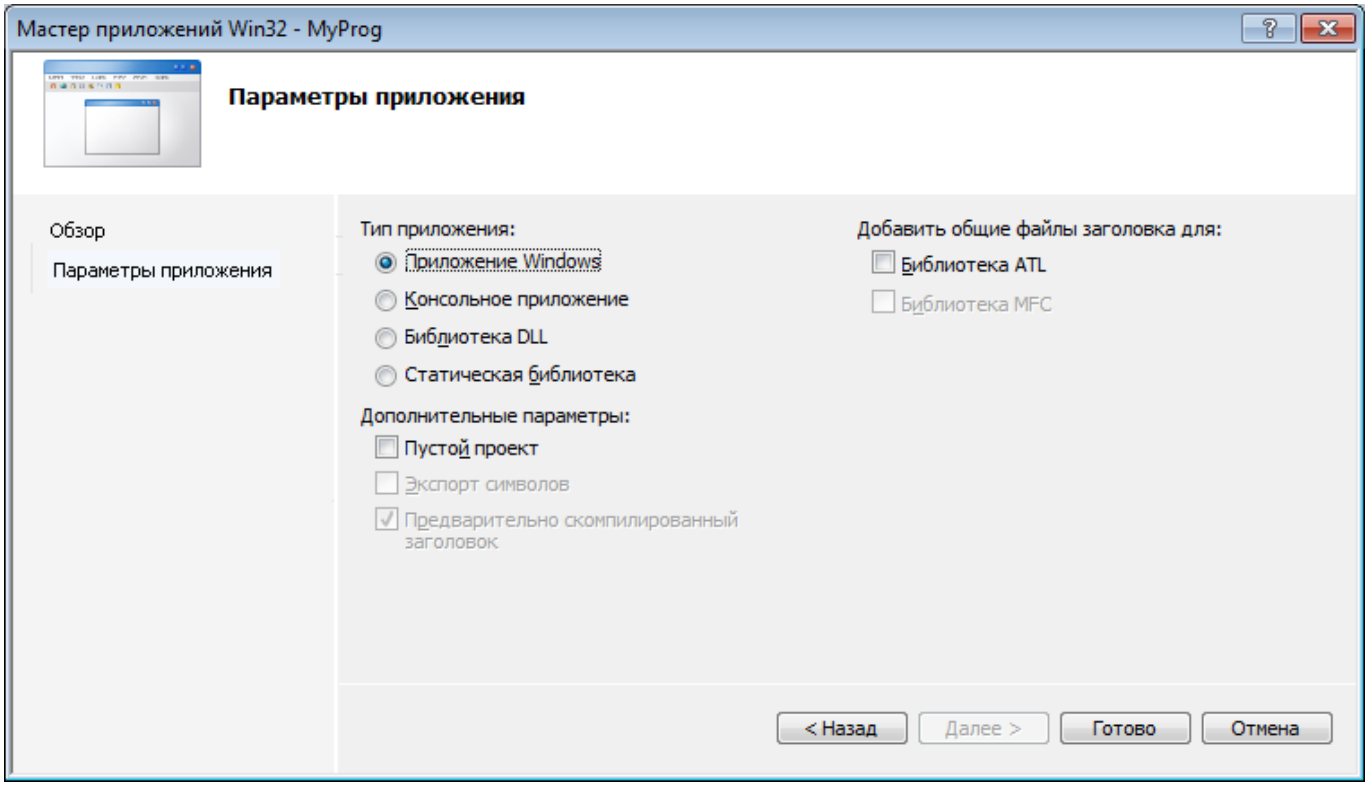


З’являється вікно, у якому необхідно вибрати тип проекту – Visual C++ Win32

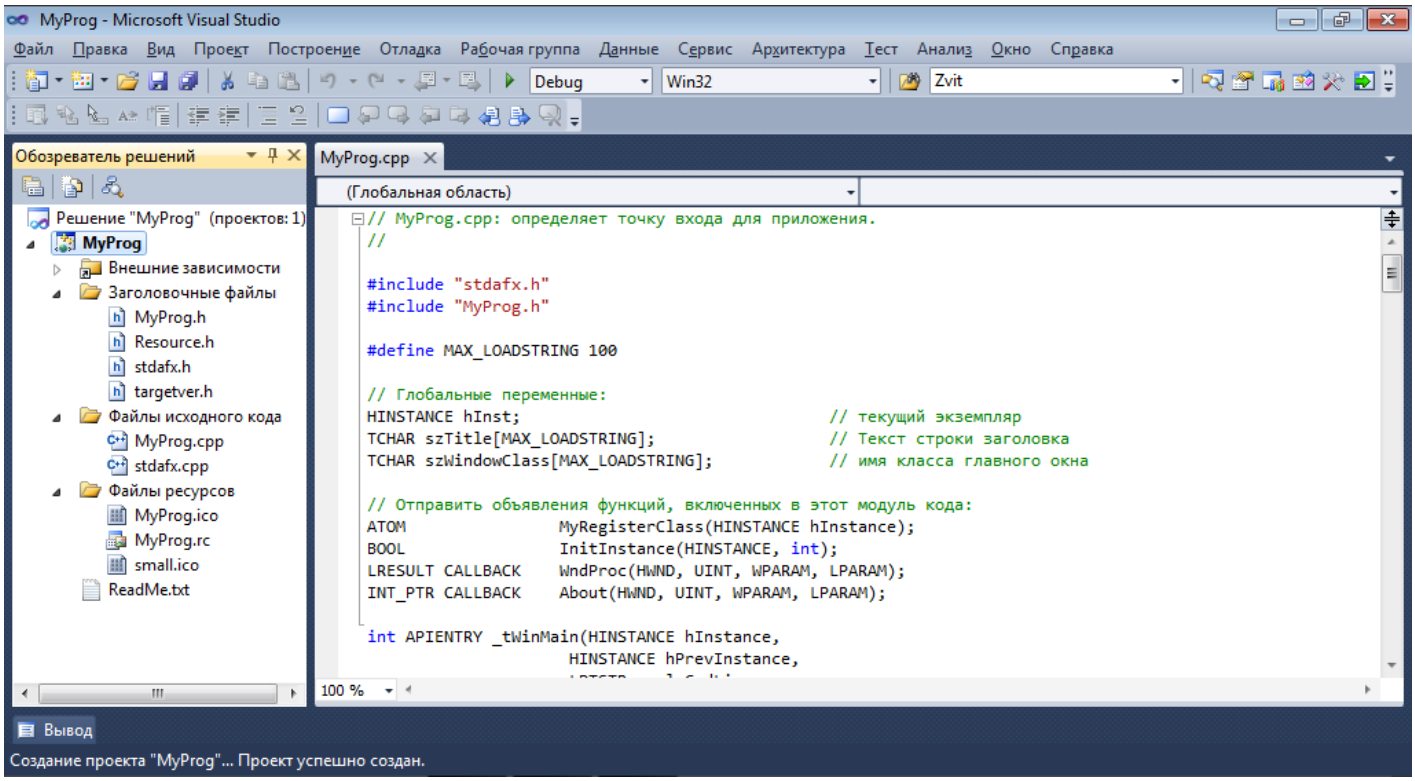


Далі визначаємо назву проекту, наприклад, MyProg і вказуємо його розташування на диску.

Рекомендація: не записуйте проекти у папки, які пропонуються за умовчанням (ProgramFiles та/або Visual Studio), створіть для своїх проектів окрему папку у корені диску, наприклад, D:\Work. Натискуємо [OK] і далі з’явиться наступне вікно:

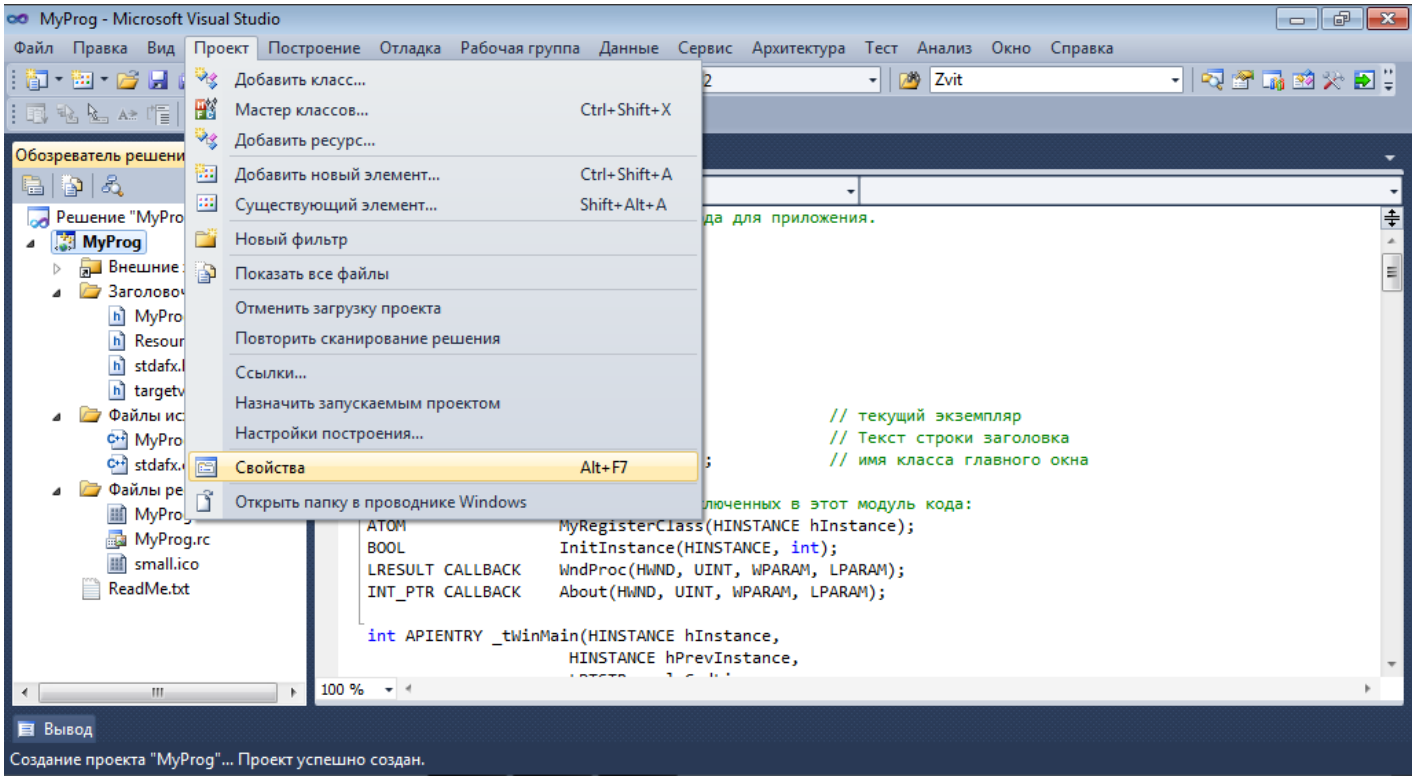


Просто натискуємо декілька разів [Далее] (або одразу [Готово]) – нічого змінювати не потрібно.
Розпочинається процес формування нового проекту. Visual Studio автоматично створює та записує потрібні файли проекту. Після цього отримаємо наступне:

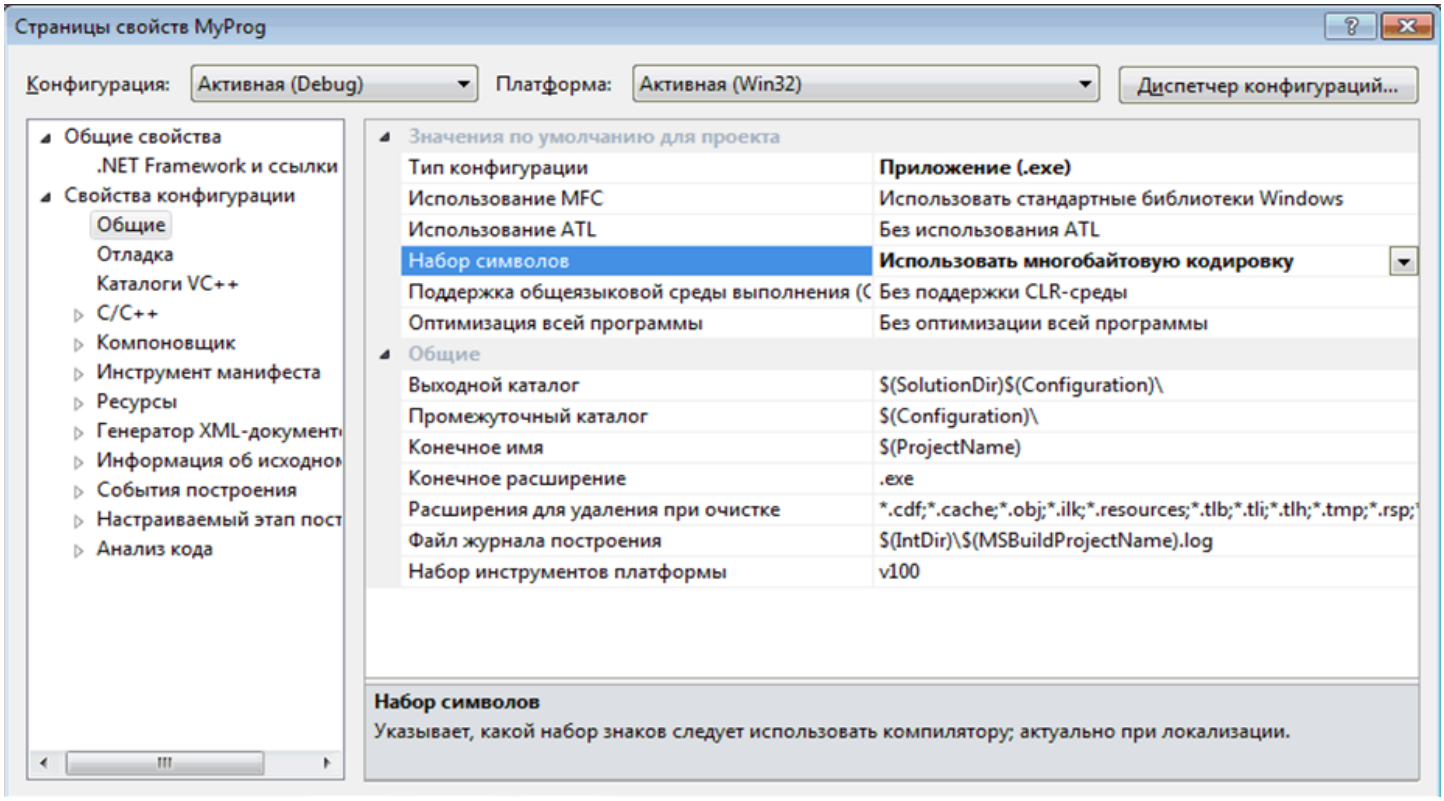


У вікні ліворуч "Обозреватель решений" показуються назви файлів, які включені до проекту. У вікні праворуч міститься текст головного файлу проекту (якщо проект буде з ім’ям Lab1, то головний файл буде мати ім’я Lab1.cpp).

Далі потрібно трохи налаштувати проект. Виберіть меню "Проект-Свойства"

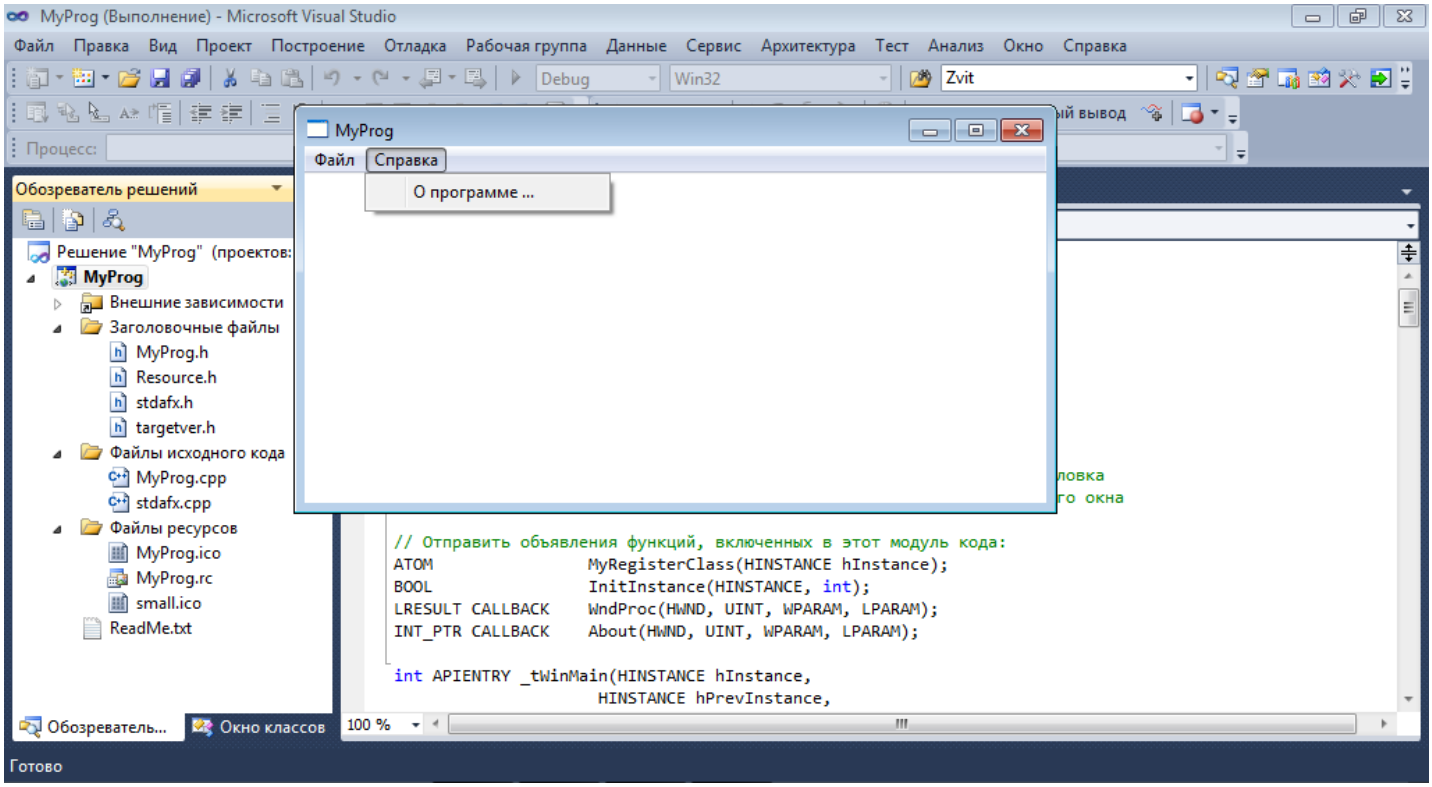


з’являється вікно властивостей проекту, у якому треба змінити набір символів з Unicode на "многобайтовую кодировку" (насправді це буде по одному байту на символ, тобто типу **char**) .



Після цього можна налагоджувати програму. Для цього виберіть меню "Отладка – Начать отладку" (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 з’явиться вікно програми.



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

Розгляд вихідного тексту програми

Вихідний текст складається з декількох файлів. Головний файл MyProg.cpp містить:

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

Нижче наведений текст файлу MyProg.cpp

Перша частина.

```
// MyProg.cpp: определяет точку входа для приложения.
//
#include "stdafx.h"
#include "MyProg.h"

#define MAX_LOADSTRING 100

// Глобальные переменные:
HINSTANCE hInst;                                // текущий экземпляр
TCHAR szTitle[MAX_LOADSTRING];                  // Текст строки заголовка
TCHAR szWindowClass[MAX_LOADSTRING];            // имя класса главного окна
```

```
// Отправить объявления функций, включенных в этот модуль кода:
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. Зверніть увагу, що хоча ми створили проект C++, проте головна функція зветься не **main** а **_tWinMain**. Як і належіть, головна функція є точкою входу програми, початок роботи програми повністю описується кодом цієї функції. Загалом для будь-якої Windows-програми 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 _tWinMain(HINSTANCE hInstance,
                     HINSTANCE hPrevInstance,
                     LPTSTR    lpCmdLine,
                     int       nCmdShow)
{
       UNREFERENCED_PARAMETER(hPrevInstance);
       UNREFERENCED_PARAMETER(lpCmdLine);

       // TODO: разместите код здесь.
       MSG msg;
       HACCEL hAccelTable;

       // Инициализация глобальных строк
       LoadString(hInstance, IDS_APP_TITLE, szTitle, MAX_LOADSTRING);
       LoadString(hInstance, IDC_MYPROG, szWindowClass, MAX_LOADSTRING);
       MyRegisterClass(hInstance);

       // Выполнить инициализацию приложения:
       if (!InitInstance (hInstance, nCmdShow))
       {
              return FALSE;
       }

       hAccelTable = LoadAccelerators(hInstance, MAKEINTRESOURCE(IDC_MYPROG));

       // Цикл основного сообщения:
       while (GetMessage(&msg, NULL, 0, 0))
       {
              if (!TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
              {
                     TranslateMessage(&msg);
                     DispatchMessage(&msg);
              }
       }

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

далі записаний текст функцій

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
. . .
       wcex.lpfnWndProc = WndProc;   //зверніть на це увагу – так передаємо адресу віконної функції
. . .

       return RegisterClassEx(&wcex);
}
```

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

Третя частина тексту – визначення віконної функції головного вікна програми. Коментарі, які записуються самим Visual Studio при створенні проекту (тут наведені для російськомовної версії) вельми доречні:

```
//  ФУНКЦИЯ: WndProc(HWND, UINT, WPARAM, LPARAM)
//
//  НАЗНАЧЕНИЕ:  обрабатывает сообщения в главном окне.
//
//  WM_COMMAND - обработка меню приложения
//  WM_PAINT  -Закрасить главное окно
//  WM_DESTROY - ввести сообщение о выходе и вернуться.
//
//
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM lParam)
{
       int wmId, wmEvent;
       PAINTSTRUCT ps;
       HDC hdc;

       switch (message)
       {
       case WM_COMMAND:
              wmId    = LOWORD(wParam);
              wmEvent = HIWORD(wParam);
              // Разобрать выбор в меню:
              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:
              hdc = BeginPaint(hWnd, &ps);
              // TODO: добавьте любой код отрисовки...
              EndPaint(hWnd, &ps);
              break;
       case WM_DESTROY:
              PostQuitMessage(0);
              break;
       default:
              return DefWindowProc(hWnd, message, wParam, lParam);
       }
       return 0;
}
```

Четверта частина тексту головного файлу – віконна функція діалогового вікно ABOUT.

```
// Обработчик сообщений для окна "О программе".
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;
}
```

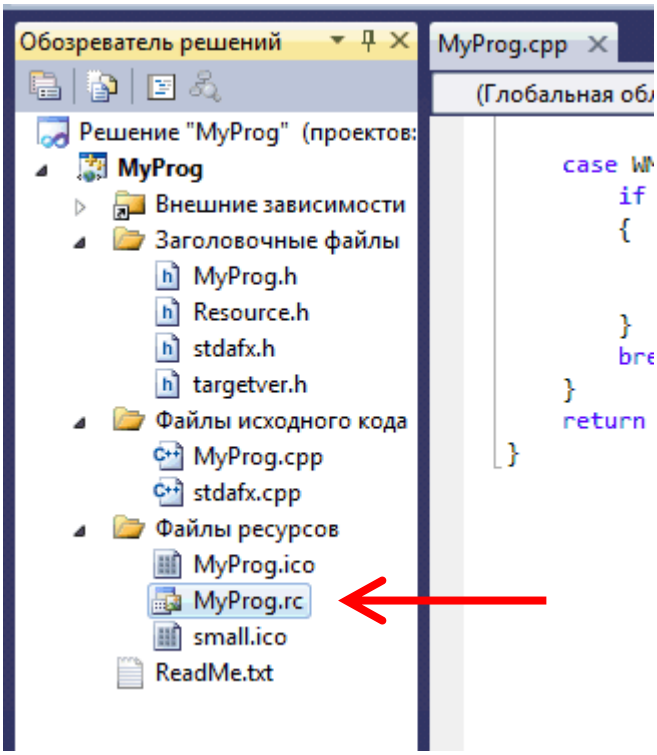
Як і належіть віконній функції, вона має чотири аргументи і має модифікатор типу CALLBACK. Це означає, що цю функцію викликає Windows – надсилає повідомлення про події, які стосуються цього вікна. Повідомлення WM_INITDIALOG надсилається при створенні вікна – можна запрограмувати обробник для ініціалізації елементів вікна (тут цього немає). Повідомлення WM_COMMAND для цього вікна означає реакцію на натискування кнопки ОК, а також закриття вікна. Закривається діалогове вікно 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:

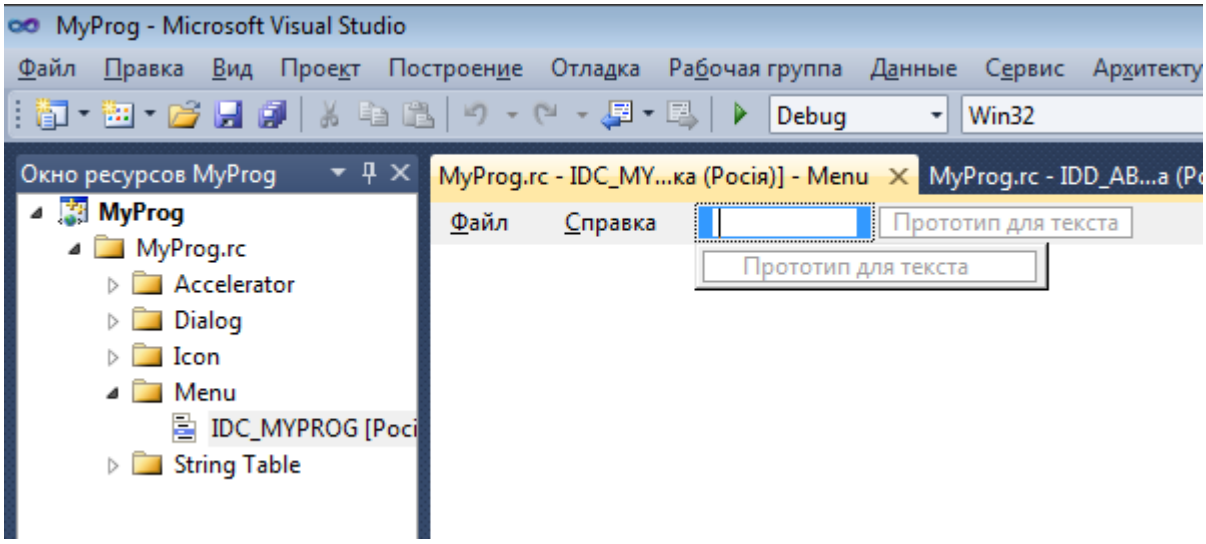
```
IDC_MYPROG MENU
BEGIN
    POPUP "&Файл"
    BEGIN
        MENUITEM "В&ыход",                IDM_EXIT
    END
    POPUP "&Справка"
    BEGIN
        MENUITEM "&О программе ...",           IDM_ABOUT
    END
END
```

Щоб редагувати меню – додати пункт меню можна викликати редактор ресурсів. Для цього треба двічи клікнути у вікні "Обозреватель решения" (Solution Explorer) на файлі MyProg.rc

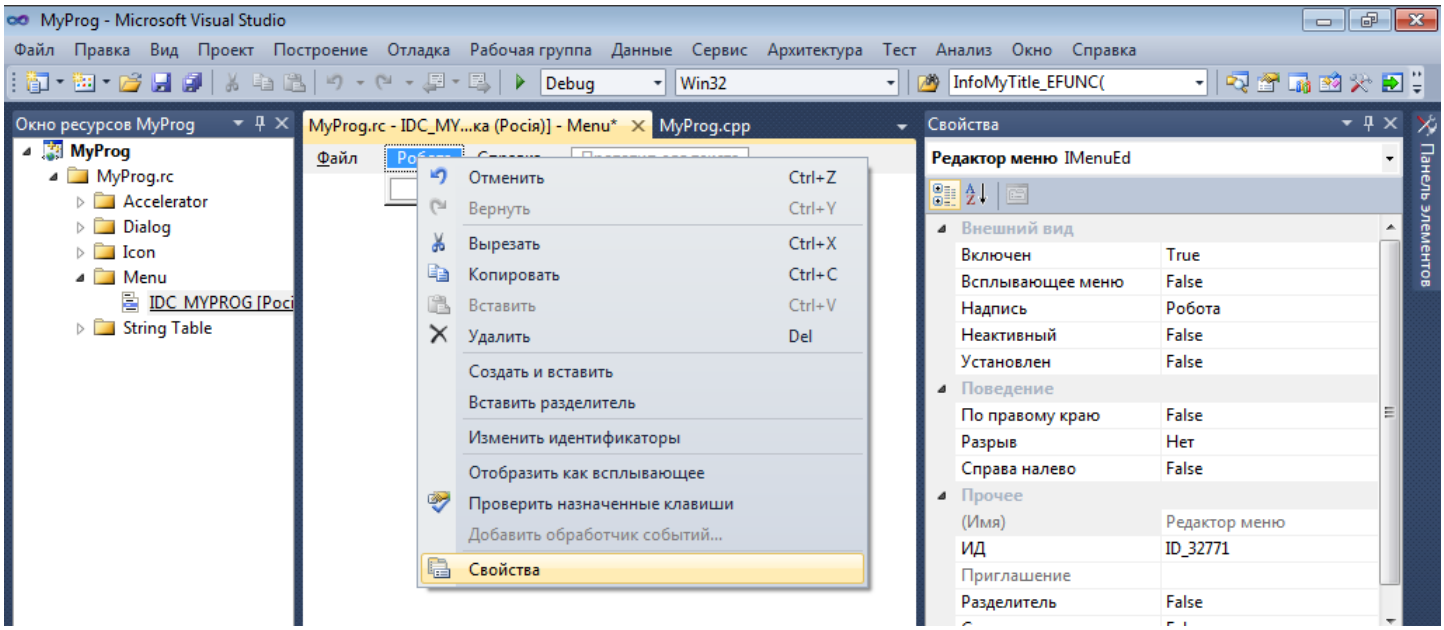


Файл MyProg.rc у списку файлів проекту

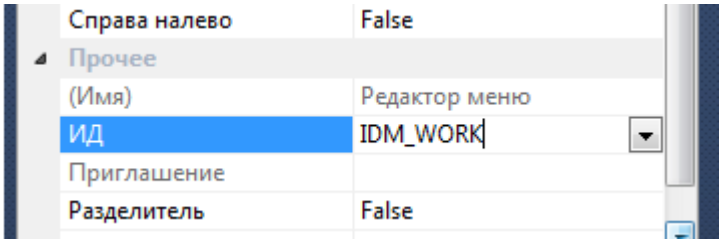
Після кліку на файлі MyProg.rc з'являється редактор ресурсів. У вікні списку ресурсів виберіть меню і двічі клікніть – відкриється меню для редагування



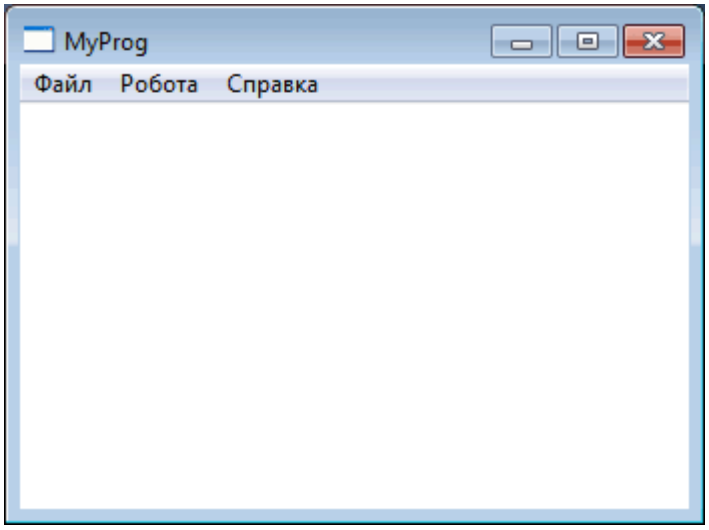
Введіть назву нового пункту меню, наприклад, "Робота", перетягніть його ліворуч пункту "Справка" (Help) і визначте його ID. За умовчанням редактор ресурсів присвоїть новому пункту меню ідентифікатор ID_32771. Це не дуже наочне імя символічної константи можна змінити. Для цього клікнути правою кнопкою на пункті меню і відкрити вікно властивостей:



Замість ID_32771 впишемо IDM_WORK



Перевіримо – перекомпілюємо проект. Для цього достатньо натиснути F5. Після успішної компіляції, лінкування та запису exe-файлу наша програма почне виконуватися



Пункт меню "Робота" можна клікнути, проте нічого не буде. Для того щоб при виборі меню щось виконувалося, потрібно це "щось" запрограмувати. Для цього треба додати у віконну функцію WndProc для обробника повідомлення WM_COMMAND рядки, які розпочинаються case IDM_WORK:

```
      case WM_COMMAND:
        wmId    = LOWORD(wParam);
        wmEvent = HIWORD(wParam);
        // Разобрать выбор в меню:
        switch (wmId)
          {
          case IDM_WORK:
            MyWork(hWnd);   // Буде нашим обробником вибіру пункту Робота
            break;
          case IDM_ABOUT:
            DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUTBOX), hWnd, About);
            break;
          case IDM_EXIT:
            DestroyWindow(hWnd);
            break;
```

Далі потрібно оголосити нашу нову функцію MyWork у верхніх рядках тексту MyProg.cpp в списку оголошень функцій

```
// Отправить объявления функций, включенных в этот модуль кода:
ATOM                MyRegisterClass(HINSTANCE hInstance);
BOOL                InitInstance(HINSTANCE, int);
LRESULT CALLBACK    WndProc(HWND, UINT, WPARAM, LPARAM);
INT_PTR CALLBACK    About(HWND, UINT, WPARAM, LPARAM);
void MyWork(HWND hWnd);      // оголошення нашшої функції
```

Далі треба визначити функцію MyWork – найкраще записати визначення наприкінці тексту MyProg.cpp

```
//функція-обробник пункту меню "Робота"
void MyWork(HWND hWnd)
{

      //Що ми тут запрограмуємо, те й буде робитися

}
```

У фігурних дужках запрограмуйте потрібний код згідно варіанту завдання.

Як створити нове діалогове вікно?

Для цього треба виконати наступні дії:

- у редакторі ресурсів додати нове діалогове вікно (Dialog) у файл ресурсів *.rc
- запрограмувати віконну функцію діалогового вікна
- запрограмувати виклик діалогового вікна у функції MyWork()

Візьміть до уваги, що у нашій програмі вже є одне діалогове вікно – ABOUT. Скористайтеся його текстом. Виклик діалогового вікна ABOUT робиться так:

```
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 віконної Callback-функції для цього вікна діалогу.

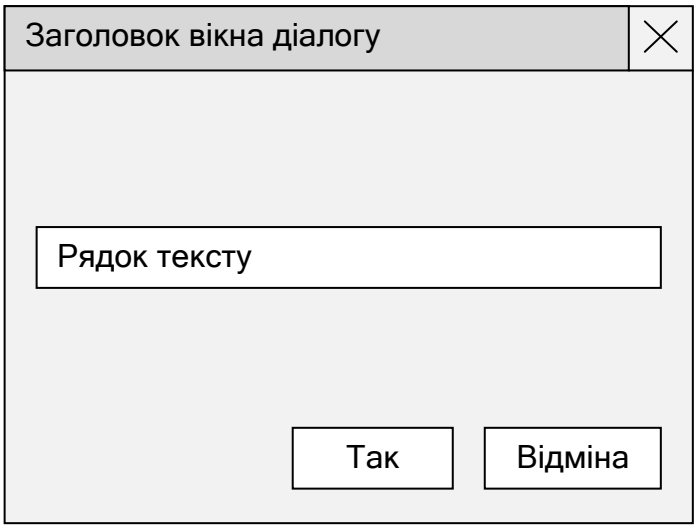
Рекомендації щодо розробки вікон діалогу

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 (Обозреватель решений для російськомовних Visual Studio)
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 віконну функцію – обробник повідомлень для цього вікна. Далі розглянемо деякі варіанти

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 отримати ідентифікатор, наприклад, IDC_EDIT1.



Потрібно запрограмувати, щоб після натискування кнопки Так (ОК) 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 MSDN.

Фрагмент функції вікна діалогу:

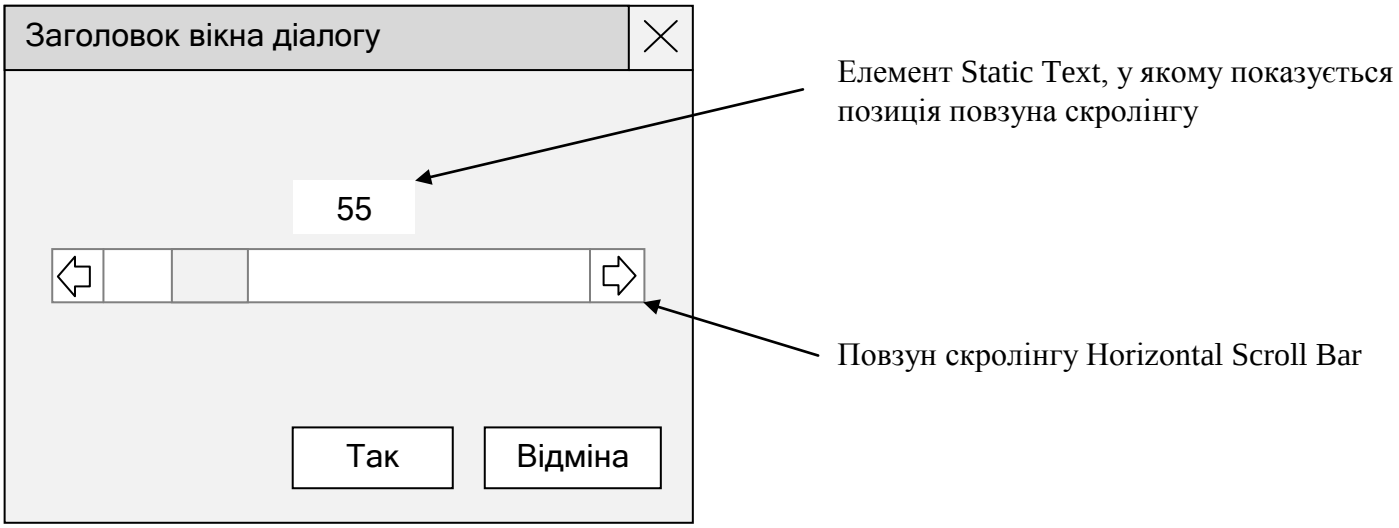
```
//Callback-функція вікна
static BOOL CALLBACK Work1(HWND hDlg,UINT iMessage,WPARAM wParam,LPARAM)
{
switch (iMessage)
  {
  case WM_COMMAND:
      if (LOWORD(wParam) == IDOK)
         {
         //... можливо, щось ще

         //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:



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);
```

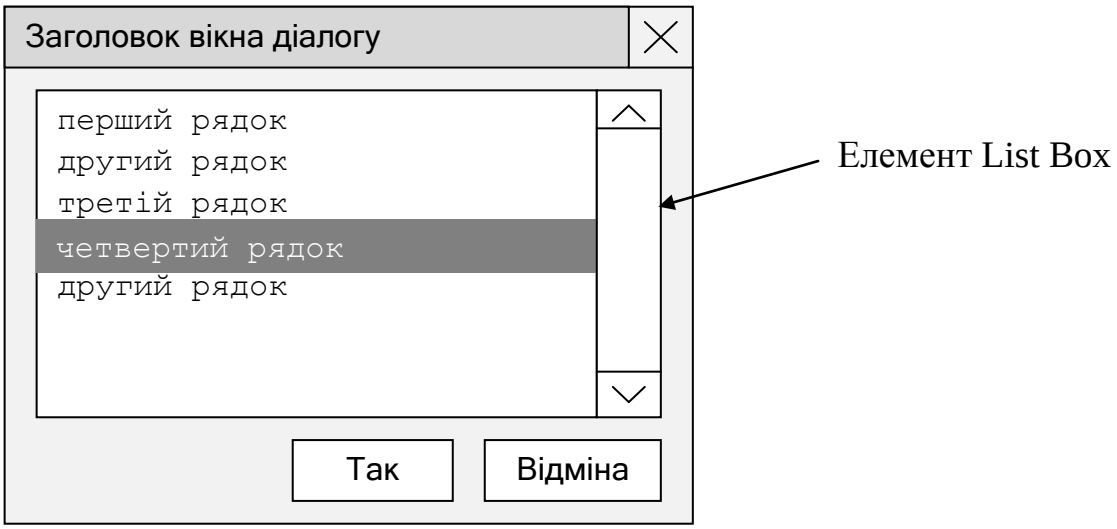
Якщо користувач торкається повзуна скролінгу, то Windows надсилає на адресу функції вікна діалогу повідомлення WM_HSCROLL. Кожне повідомлення містить параметри. Для повідомлення WM_HSCROLL у параметр wParam записується код дії а також позиція повзуна. Нижче фрагмент функції вікна діалогу:

```
//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 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 List Box

У редакторі ресурсів на поверхню вікна додаємо такі елементи:



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

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

Тут 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 діалогове вікно тільки починає зявлятися

Для читання вибраного елемента списку у буфер можна використати функцію

```
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 програмуємо відповідний обробник повідомлення WM_COMMAND у функції вікна.

Нижче наведені фрагменти функції вікна діалогу, як приклад реалізації списку

Фрагменти програмного коду функції вікна діалогу зі списком

```
//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 зроблений занадто громіздким і заплутаним. Одним з факторів заплутаності може стати текст Callback-функції вікна – як головного, так і діалогового. 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 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 розділити, структурувати шляхом оформлення обробника кожного повідомлення у вигляді окремої функції. Нижче наведений скелет струкурованого програмного коду

```
//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(hDlg, . . .)**
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.

Потужним засобом структурованості є модульність, яка розглядається нижче.

Частина 2. Розробка модульного проекту програми у середовищі Microsoft Visual Studio

Після того, як був створений проект C++ Win32 та перевірено роботу першої програми, можна продовжувати подальшу розробку.

Далі розглянемо наступні 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 проектів Win32). 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 загальноприйняте для мови C++

Для проектів Visual C++ Win32 додатково використовуються також й інші файли. Зокрема:

3. Файли **stdafx.cpp** та **stdafx.h**
4. Файл ресурсів *.rc, наприклад, **module1.rc** (це необов'язково, тільки у випадку, коли модуль окремо репрезентує якийсь елемент інтерфейсу користувача, наприклад, діалогове вікно 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 "stdafx.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 діалогового вікна--
INT_PTR CALLBACK DlgProc_MOD1(HWND hDlg,UINT message,WPARAM wParam,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 проект – він повинен бути у списку файлів Source Files вікна Solution Explorer. Тільки тоді цей файл буде компілюватися.
Примітка. У проектах Win32 на початку тексту файлів *.cpp повинен бути рядок:

```
#include "stdafx.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
```

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
```

Рекомендується подібним чином оформлювати усі заголовочні файли *.h.

Файл module1.h включати у проект 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 у проекті 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, діалогових вікон, та інших ресурсів. Структура файлу **module1.rc**:

```
#include "stdafx.h"
 . . .

IDD_MYDIALOG_MOD1 DIALOGEX . . .
. . .
```

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

Примітка. Ідентифікатори елементів ресурсів, наприклад ід. діалогових вікон, є константами. Важливо те, що значення таких ід. повинні бути унікальними – не може співпадати з ід. інших відповідних ресурсів проекту. 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, *.rh, *.rdh тощо.
Наприклад, якщо визначення **IDD_MYDIALOG_MOD1** зробити у файлі **module1.rh:**

```
#ifndef _MODULE1_RH_
#define _MODULE1_RH_

#define IDD_MYDIALOG_MOD1 1001
. . .

#endif
```

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

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

 . . .

IDD_MYDIALOG_MOD1 DIALOGEX . . .
. . .
```

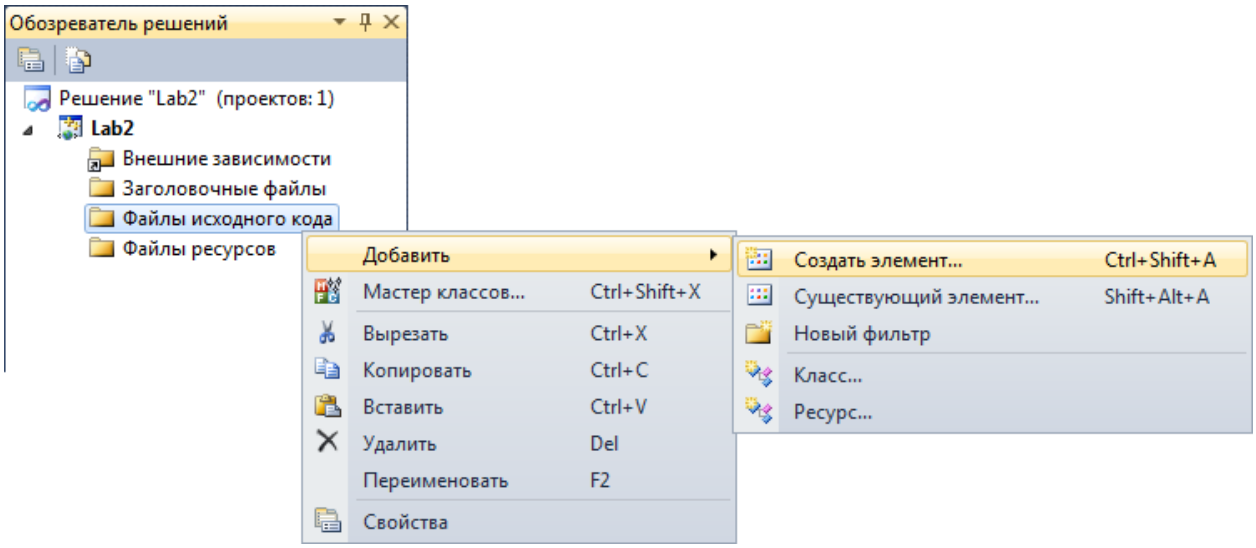
а якщо у файлі **module1.cpp** також використовується ім'я **IDD_MYDIALOG_MOD1**, то у цьому файлі повинно бути #include "module1.rh":

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

. . .
```

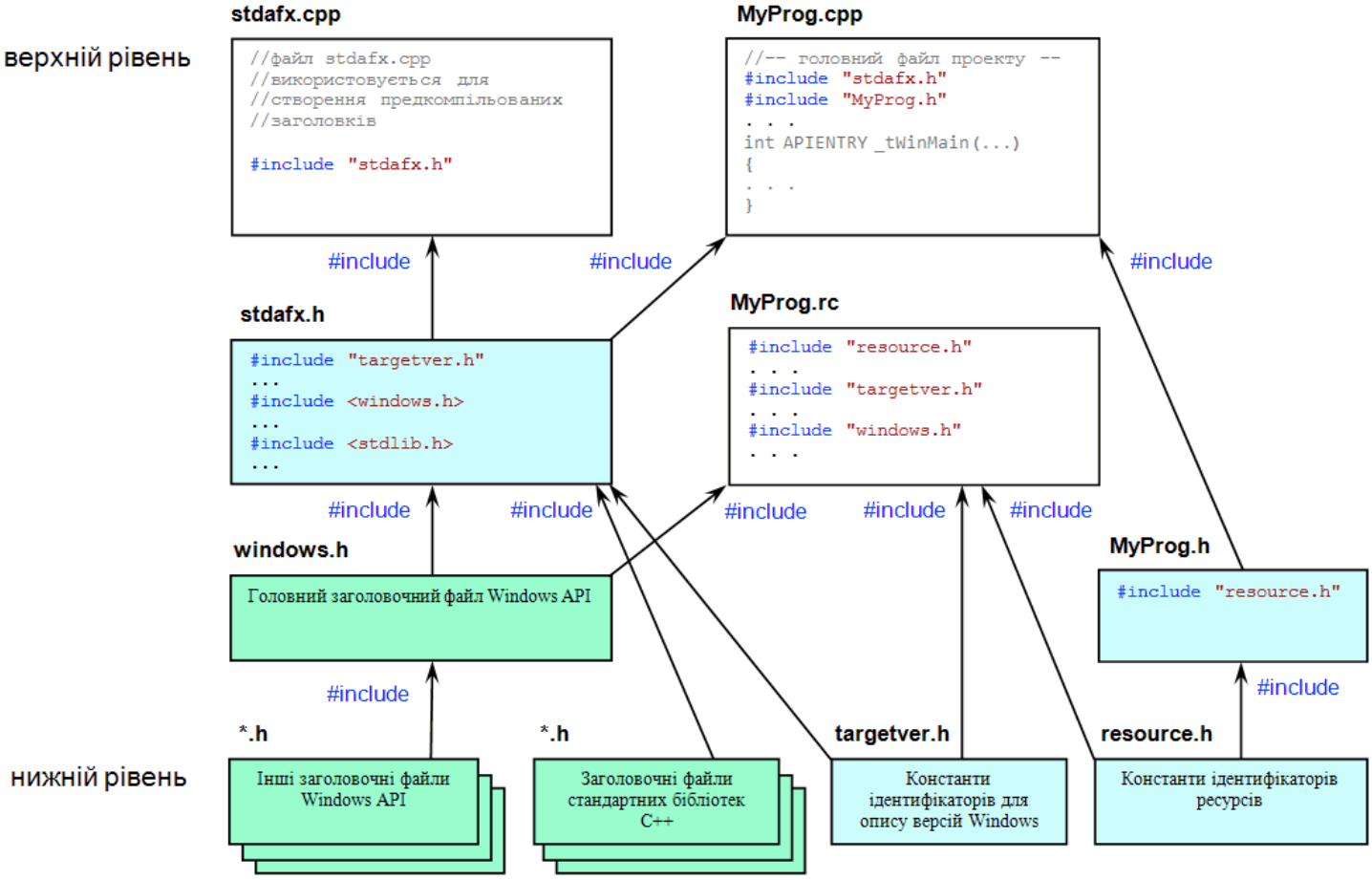
Як додати модуль в проект?

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 "Файлы исходного кода", клацніть правою кнопкою миші, і потім - "Создать элемент" (для російськомовної Visual Studio 2010):



#include – ієрархія файлів

Описує відношення включення файлів проекту 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.



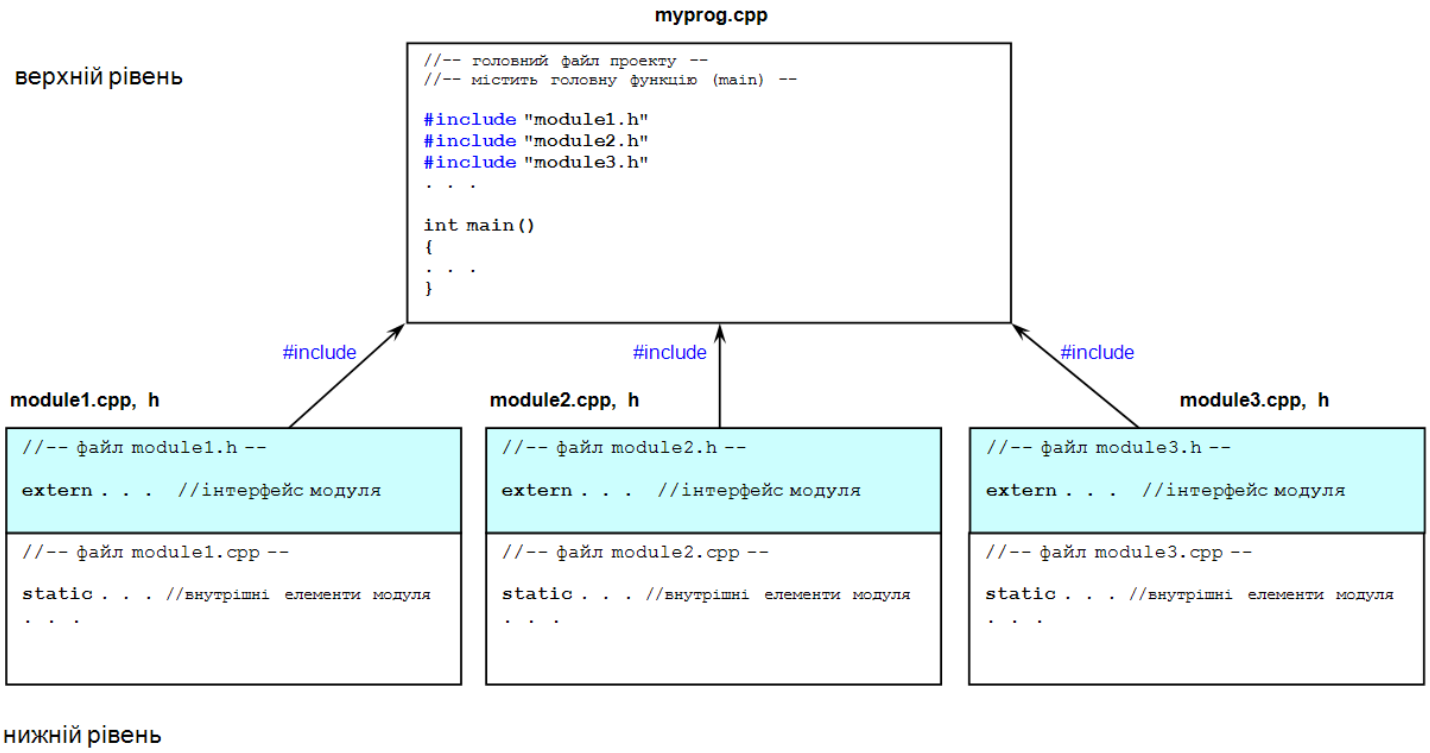
Приклад #include-ієрархії файлів найпростішого проекту Visual C++ Win32

#include – ієрархія модулів програми

Необхідно відзначити, що поняття #include-ієрархії **модулів** дещо відрізняється від #include-ієрархії **файлів** програм.

Початковий проект C++ Win32, який автоматично створює Microsoft Visual Studio, не є модульним у контексті даної лабораторної роботи. І це незважаючи на те, що проект містить багато пов'язаних між собою файлів, як наведено вище.

Тепер уявимо, що у проекті є три модуля та головний файл програми. Спростимо розгляд, відкинувши особливості проектів Win32. Відобразимо #include-ієрархію модулів наступним чином:



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.

Модулі можуть роздільно компілюватися – відповідно створюються об'єктні файли **module1.obj, module2.obj** та **module3.obj.**

У лабораторній роботі належіть розробити саме такі модулі, врахувавши специфіку програмування на основі Windows API.

Рекомендації щодо модуля, який показує діалогове вікно

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 діалогове вікно, 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"
```

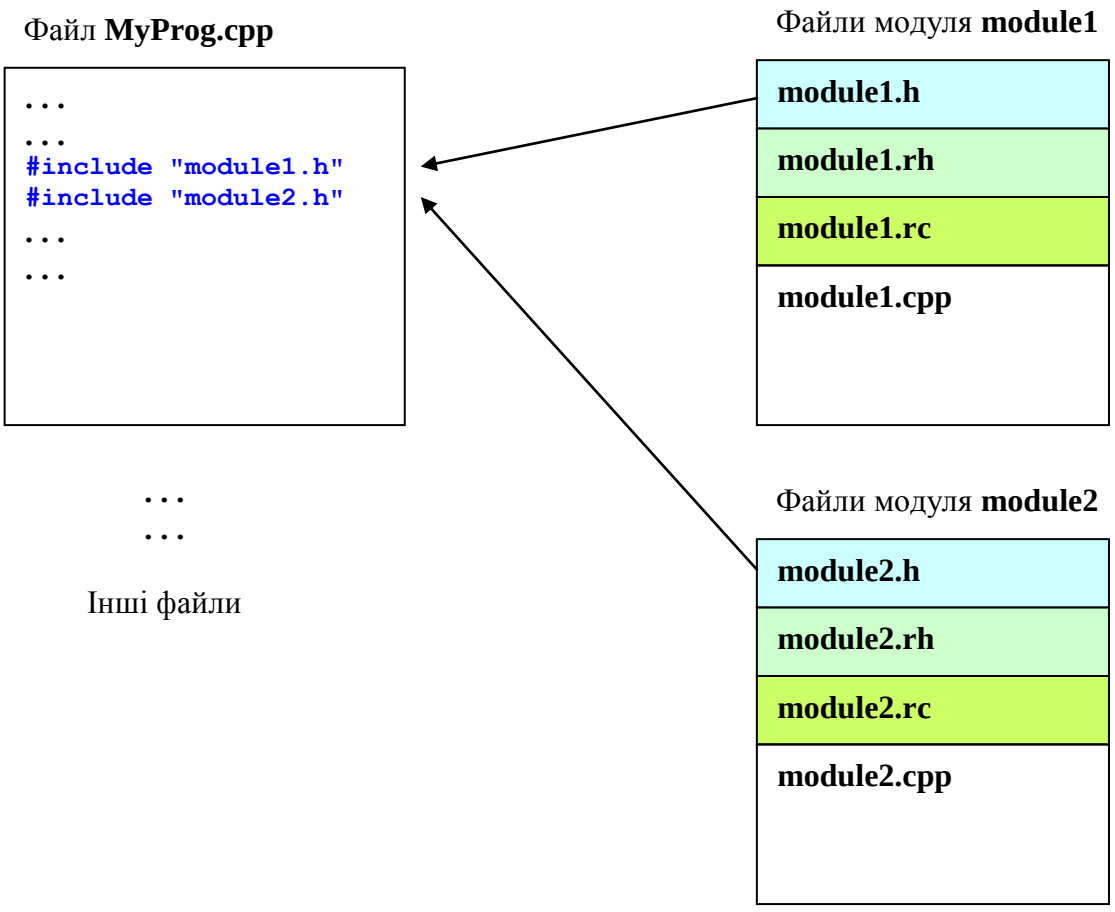
Для того, щоб в інтерфейсній функції модуля здихатися параметра **hInst**, який потрібен для DialogBox, можна скористатися функцією **GetWindowLongPtr( )**, яка при параметрі GWLP_HINSTANCE повертає значення HINSTANCE.

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 діалогове вікно, буде запрограмований вже чотирма файлами: ***.cpp**, ***.h**, ***.rc**, ***.rh**.

Тоді загальна структура проекту, який складатиметься з двох модулів, буде такою:



Як розібратися в усьому цьому?

Найкраще джерело по розробці програм на основі інструментів Microsoft – довідкова служба у вигляді біліотеки MSDN. Знайти її можна на сайті Microsoft за адресою:
https://msdn.microsoft.com
Ця довідкова служба також викликається з середовища Visual Studio або через Internet, або може бути сформована як локальний ресурс

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

та інших.

Існує багато інших сайтів, зокрема для початківців можна порекомендувати перші лекції з курсу "Прикладное программное обеспечение. 1. Visual C/C++ (лекции)" Кравцов Г.М., Львов М.С.
http://dls.kherson.ua/dls/Library/LibdocView.aspx?id=3656141d-c966-41af-a825-32b1e1f48f4c

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 mod 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) mod 4

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

4. **Callback-функції діалогових вікон 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 діалогових вікон. Це забезпечується тим, що для кожного модуля **moduleX** має бути власний файл ресурсів **moduleX.rc**.

5. Інтерфейс модулів. Інтерфейс кожного модуля, наприклад, модуля **module1**, записується у відповідний файл заголовку **module1.h**. Інтерфейс модуля повинен бути у вигляді єдиної 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 діалоговому вікні кнопку "Відміна" ("Cancel"), або "×" у правому верхньому куті вікна.
Інших функції, перемінних у інтерфейсі модуля не повинно бути.

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

7. Намалювати повну #include-ієрархію усіх наявних файлів проекту (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, наведеній нижче

Таблиця варіантів

Варіант	Що треба зробити	Підказка
0	Діалогове вікно 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	Діалогове вікно 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 віконній функції діалогового вікна. Вивести число у головному вікні – функція **TextOut** в обробнику повідомлення WM_PAINT. Щоб число вивести як текст – функція **itoa** або **ltoa** перетворює ціле число у рядок тексту.
2	Два діалогових вікна. Спочатку з'являється перше, яке має дві кнопки: [Далі >] і [Відміна]. Якщо натиснути кнопку [Далі >], то з'явиться друге длг вікно, яке має три кнопки: [< Назад], [Так] і [Відміна]. Якщо натиснути кнопку [< Назад], то перехід до першого длг вікна.	Аналізувати результат роботи длг вікна можна по значенню, яке повертає DialogBox. Значення визначається другим аргументом функції EndDialog, яка розташовується у віконнній функції. Можна, наприклад: - для кнопки [Далі >] робити виклик функції EndDialog(hDlg, 1) - для кнопки [Відміна] – EndDialog(hDlg, 0) - для кнопки [< Назад] – EndDialog(hDlg, -1)
3	Діалогове вікно 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 рядку, 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 – функція **TextOut** в обробнику повідомлення WM_PAINT

Примітка. Незважаючі на однакові варіанти завдань, рішення повинні бути індивідуальними. Однаковість рішень буде штрафуватися.

Контрольні запитання

1. Що робить головна функція програми?
2. Чим відрізняється реєстрація вікна від його створення?
3. Що робить у віконній функції оператор switch?
4. Як створюється новий пункт меню?
5. Як викликається діалогове вікно?
6. Як створити нове діалогове вікно?
7. Де запрограмований цикл очікування повідомлень головного вікна?
8. Що таке WM_COMMAND?
9. Які файли потрібні для модулів проекту Win32?
10. Що таке #include – ієрархія?
11. Що робить директива #include?
12. Чим відрізняється віконна функція головного вікна від діалогового?
13. Що означає роздільна компіляція модулів і як її досягти?

У ході захисту-прийняття роботи викладач може також запитувати інше, що стосується виконання роботи.

Зміст звіту

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