Лабораторная работа

Часть 1. Создание простейшего приложения с использованием библиотеки Win32 API

Цель работы: познакомиться со структурой простейшего Windows-приложения с использованием интерфейса Win32 API, с типами данных и основными функциями Win API, научиться создавать простейшее однооконное приложение в интегрированной среде Visual C++ 6.0.

Задание

Создать с помощью мастера AppWizard полноценное приложение для операционной системы Windows на основе интерфейса Win32 API. Программа должна выводить на экран главное окно со строкой меню. Меню должно содержать команды вывода сообщений с помощью стандартных окон MessageBox. Варианты оформления окон должны отличаться друг от друга количеством кнопок и иконками.

Последовательность выполнения работы

1. Загрузите интегрированную среду Visual C++ 6.0.
2. Выполните команду File/New.
3. Выберите вариант приложения - Win32 Application, укажите имя проекта и место размещения проекта, нажмите кнопку ОК.
4. Выберите вариант приложения “A typical “Hello, World!” application” (приложение, выполняющее вывод строки текста “Привет Мир!” на экран) и нажмите на кнопку Finish.
5. Откомпилируйте проект в конфигурации DEBUG (отладка, устанавливается по умолчанию) и выполните его.
6. Изучите список файлов проекта, просмотрите их содержимое и определите их назначение в проекте. Используйте окно Workspace, вкладка File View.
7. Изучите список классов и ресурсов проекта. Используйте окно Workspace, вкладки Class View и Resource View.
8. Определите, сколько места на диске занимает приложение.
9. Удалите каталог DEBUG, установите конфигурацию проекта RELEASE (реализация) с помощью команды Build/Set Active Configuration… и перекомпилируйте проект. Определите, сколько места на диске занимает приложение, сравните с конфигурацией DEBUG.
10. Завершите работу над проектом с помощью команды File/Close Workspace.
11. Создайте в соответствии с п. 2-4 новое Windows-приложение “A Simple Win32 Application” (Простое Win32 приложение).
12. Изучите состав файлов, классов и ресурсов, включённых в проект мастером AppWizard.
13. Добавьте в файл, содержащий функцию WinMain, текст, так чтобы ваше приложение открывало окно, содержащее строку текста. Полный текст файла, содержащего WinMain, а также пояснения к листингу приведены в справочном материале.
14. Откомпилируйте и выполните приложение.
15. Добавьте в приложение меню, в которое включите не менее четырех команды, с помощью которых на экране будут появляться окна с сообщениями. Меню должно содержать команды непосредственного выполнения или всплывающие меню первого и второго уровня. Для создания всплывающего меню в свойствах элемента меню установите флажок Pop-up. Для создания окон с сообщениями используйте функцию MessageBox. Окна должны отличаться друг от друга количеством кнопок и иконками (см. Справочный материал). Последовательность создания меню приведена в справочном материале.
16. Откомпилируйте, выполните и протестируйте ваше приложение.
17. Результаты покажите преподавателю.

Справочный материал

Структура приложения на основе интерфейса Win32 API

Базовая структура приложения Win32 содержит:

- Функцию WinMain() точка входа в программу;
- Функцию обработки сообщений процедура окна (WndProc).
- Подключение заголовочного файла windows.h, содержащего заголовки функций Windows

В среде Visual C++ подключение заголовочного файла windows.h происходит через служебный модуль среды StdAfx в файле stdafx.h .

Текст простейшей программы, выводящей в окно строку текста (главный файл, содержащий функцию WinMain):

```
#include <stdafx.h>

LRESULT CALLBACK WndProc (HWND, UINT, WPARAM, LPARAM);

int APIENTRY WinMain (HINSTANCE hInstance,
                      HINSTANCE hPrevInstance,
                      LPSTR lpCmdLine,
                      int nCmdShow)
{
  static char          szAppName[] = "Hello";
  HWND              hwnd;          // Дескриптор окна
  MSG                msg;          // Сообщение
  WNDCLASSEX  ws;                  // Окно

  ws.cbSize              = sizeof (ws);              // размер структуры окна
  ws.style               = CS_HREDRAW | CS_VREDRAW;  // стили класса
  ws.lpfnWndProc      = WndProc;                     // оконная процедура
  ws.cbClsExtra         = 0;                        // дополнительное поле для класса
  ws.cbWndExtra       = 0;                          //  дополнительное поле для окна
  ws.hInstance          = hInstance;                // дескриптор приложения
  ws.hIcon              = LoadIcon (NULL, IDI_APPLICATION);   //  иконка
  ws.hCursor            = LoadCursor (NULL, IDC_ARROW);       //   курсор
  ws.hbrBackground  = (HBRUSH)(COLOR_BTNFACE + 1);    //  кисть
  ws.lpszMenuName  = NULL;              // меню
  ws.lpszClassName   = szAppName;       // имя класса окна
  ws.hIconSm           = LoadIcon (NULL, IDI_APPLICATION);   // маленькая
                                                              //иконка

  RegisterClassEx(&ws);                 //регистрация класса

// Создание окна
  hwnd = CreateWindow (szAppName,                   // имя класса окна
                       "Hello World",               // загловок
                        WS_OVERLAPPEDWINDOW,        // тип окна
                        CW_USEDEFAULT,// x-координата левого верхнего угла окна
                        CW_USEDEFAULT,// y-координата левого верхнего угла окна
                        CW_USEDEFAULT,              //  ширина окна
                        CW_USEDEFAULT,              // высота окна
                        NULL,                       // указатель родительского окна
                        NULL,                       // указатель меню для окна
                        hInstance,                  // идентификатор приложения
                        NULL);                      //параметры создания

// Вывод окна на экран
  ShowWindow (hwnd, nCmdShow);
  UpdateWindow (hwnd);

// Цикл обмена сообщениями
  while (GetMessage (&msg, NULL, 0, 0))
```

```
 {
          TranslateMessage (&msg);
          DispatchMessage (&msg);
 }

 return msg.wParam;
}

LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg,
                                    WPARAM wParam, LPARAM lParam)
{
    HDC hdc;              // Дескриптор устройства
     PAINTSTRUCT ps;
     int  x = 100;
     int y = 100;
     char str[] = “Hello World”;

   switch (iMsg)
   {
        case WM_PAINT:                  // Сообщение, вызывающее перерисовку окна
              hdc = BeginPaint(hwnd, &ps);
              TextOut(hdc, x, y, (LPCTSTR)str, strlen(str));
               EndPaint (hwnd, &ps);
               break;
        case WM_DESTROY:              // Сообщение, вызвающее закрытие программы
              PostQuitMessage(0);
               break;
        default:
                 return DefWindowProc(hwnd, iMsg, wParam, lParam);
   }
   return 0;
}
```

1. **<u>Функция WinMain</u>**:

Параметры:

- hInstance – дескриптор (описатель) приложения
- hPrevInstance – дескриптор (описатель) предыдущего экземпляра приложения
- lpCmdLine – указатель на командную строку
- nCmdShow – переменная, влияющая на тип открытия окна (окно развёрнутое или свёрнутое в иконку)

В функции WinMain объявляется переменная ws класса WNDCLASSEX, предназначенная для хранения параметров главного окна программы, и определяются значения свойств создаваемого окна. Функция CreateWindow создаёт окно с заданными свойствами и возвращает его дескриптор. Дескриптор (описатель) представляет собой указатель на область памяти, где размещен объект Windows – окно, кисть, перо, меню и т.д.; дескриптор созданного окна присваивается переменной hwnd. Функция ShowWindow() отображает на экране окно в соответствии со значением параметра nCmdShow; UpdateWindow() перерисовывает клиентскую (внутреннюю) область окна. После этого выполняется главный цикл приложения – опрос очереди сообщений Windows с помощью функции GetMessage() – и выполнение действий, соответствующих сообщениям. Цикл работает до тех пор, пока не получит сообщение WM_QUIT (при этом GetMessage() вернёт значение 0, после обработки любых других сообщений GetMessage() возвращает значение, отличное от нуля). Внутри цикла функция TranslateMessage() преобразует сообщения от клавиатуры, вторая - DispatchMessage() - посылает сообщения в оконную процедуру.

2. **<u>Оконная процедура</u>**.

Прототип:

```
LRESULT CALLBACK <Name>(HWND, UINT, WPARAM, LPARAM);
```

Заголовок:

```
LRESULT CALLBACK WndProc (HWND hwnd, UINT iMsg, WPARAM
wParam, LPARAM lParam)
```

hwnd - дескриптор окна

iMsg - идентификатор сообщения

wParam, lParam - дополнительные параметры.

Внутри процедуры оператор switch распознаёт сообщение, вызывает заданную процедуру обработки сообщения и возвращает 0; в противном случае вызывает процедуру обработки сообщения по умолчанию (DefWindowProc()). Каждому сообщению в Windows присвоен собственный уникальный идентификатор, например:WM_PAINT, WM_CREATE, WM_USER, WM_… и т. д. В принципе можно не обрабатывать ни одного сообщения, кроме WM_DESTROY, это сообщение не обрабатывается Windows по умолчанию. Для обработки сообщения WM_DESTROY требуется вызов ф-ции PostQuitMessage(), которая вставляет в очередь сообщений сообщение WM_QUIT, вызывающее завершение программы. Сообщение WM_PAINT возникает при любых изменениях окна (изменение размера, сворачивание, разворачивание, перекрытие другими окнами), при этом Windows помечает данную область как некорректную. При обработке сообщения WM_PAINT Windows перерисовывает рабочую область окна, при этом вызов функции BeginPaint() создаёт дескриптор устройства hdc, с помощью которого производится отображение в окне. Функция EndPaint() уничтожает дескриптор устройства и помечает область окна как корректную.

Типы данных в Win32 API

Идентификатор типа	Описание типа
Простые типы	
UINT, DWORD	32-разрядное беззнаковое целое
INT	32-разрядное знаковое целое
USHORT, WORD	16-разрядное беззнаковое целое
SHORT	16-разрядное знаковое целое
DWORD64, UINT64, ULONGLONG	64-разрядное беззнаковое целое
INT64, LONG64, LONGLONG	64-разрядное знаковое целое
CHAR	8-битный символ (ANSI Windows)
WCHAR	16-битный символ (Unicode)
TCHAR	CHAR или WCHAR, если используется Unicode
BOOL, BOOLEAN	логический (TRUE или FALSE)
Специальные типы	
LRESULT	то же, что и long
WPARAM	то же, что и long
LPARAM	то же, что и long
CALLBACK	ссылка на особую последовательность вызовов функций, которая имеет место между операционной системой Windows и приложением.
WINAPI	ссылка на особую последовательность вызовов функций, которая имеет место между операционной системой Windows и приложением.
Дескрипторы	
HINSTANCE	дескриптор экземпляра программы, 32-разрядное целое
HWND	дескриптор окна, 32-разрядное целое
HDC	дескриптор контекста устройства, 32-разрядное целое
HICON	дескриптор иконки, 32-разрядное целое
HCURSOR	дескриптор курсора мыши, 32-разрядное целое
HBRUSH	дескриптор кисти, 32-разрядное целое
HMENU	дескриптор меню

LPSTR	32-битный (длинный) указатель на строку символов (char*)
Указатели	
LPCSTR	32-битный (длинный) указатель на константную строку символов (char*)
LPTSTR	32-битный (длинный) указатель на строку символов, совместимую с Unicode
LPCTSTR	32-битный (длинный) указатель на константную строку символов, совместимую с Unicode
Структуры	
MSG	структура атрибутов сообщения
WNDCLASSEX	структура атрибутов класса окна
PAINTSTRUCT	структура атрибутов для рисования
RECT	структура атрибутов прямоугольника

Ресурсы приложения

К ресурсам приложения относятся диалоговые окна, акселераторы (горячие клавиши), меню, иконки, рисунки, строки. Все используемые программой ресурсы включаются в файл ресурсов *.rc . Каждый элемент ресурсов имеет имя или идентификатор. Имя задаётся в двойных кавычках, а идентификатору автоматически ставится в соответствие некоторое числовое значение, которое можно прочитать в заголовочном файле resource.h, подключённом к проекту.

Создание меню

1. Добавить новый ресурс – меню – командой Insert/Resource -> Menu ->New. По умолчанию ресурсу будет дано имя (идентификатор ресурса) IDR_MENU1. Оставим имя без изменения.
2. Создать элементы меню в графическом редакторе. Для каждого пункта меню, кроме самых верхних, в окне Properties указать идентификатор (окно ID) и надпись (Caption). Для верхних пунктов меню указываются только надписи.
3. Сохранить файл ресурсов scrip1.rc, содержащий меню, в папку проекта. При сохранении можно изменить имя файла.
4. Добавить файл ресурсов в проект в окне Workspace, FileView, папка Source Files.
5. Добавить файл resource.h в проект в окне Workspace, FileView, папка Header Files.
6. Добавить подключение файла resource.h в главный файл проекта, содержащий WinMain.
7. Включить меню в класс главного окна через параметр lpszMenuName:

```
lpszMenuName = (LPCTSTR)IDR_MENU1;
```

8. Включить обработку пунктов меню в цикл обработки сообщений. Вызов пунктов меню создает сообщение WM_COMMAND, а выбор пункта меню связан с параметром оконной процедуры wParam: в его младшее слово помещается значение выбранного пункта меню:

```
switch (iMsg)

case WM_COMMAND:
      if (LOWORD(wParam) == <идентификатор пункта меню>)
           <выполняемые действия>;
      . . . . . .
}
```

Функция LOWORD возвращает младшее слово аргумента, представляющего собой длинное целое.

Использование окна сообщений Message Box.

Окно сообщений создаётся с помощью функции MessageBox():

```
int MessageBox(HWND hwnd, LPCTSTR lpText, LPCTSTR lpCaption,
UINT uType);
```

Параметры функции:

- hwnd – дескриптор окна;
- lpText – указатель на строку, содержащую текст в окне сообщения;
- lpCaption – указатель на строку, содержащую заголовок окна сообщения;
- uType – стиль окна сообщения (количество и вид кнопок, отображаемые иконки).

Кнопки:

Значения параметра uType	Количество кнопок	Назначение кнопок	Возвращаемые значения
MB_OK	1	Кнопка OK, значение по умолчанию	IDOK
MB_OKCANCEL	2	Кнопки OK и Cancel	IDOK, IDCANCEL
MB_YESNO	2	Кнопки Yes и No	IDYES, IDNO
MB_RETRYCANCEL	2	Кнопки Retry и Cancel	IDRETRY, IDCANCEL
MB_YESNOCANCEL	3	Кнопки Yes, No, Cancel	IDYES, IDNO, IDCANCEL
MB_ABORTRETRYIGNORE	3	Кнопки Abort, Retry, Ignore	IDABORT, IDRETRY, IDIGNORE

Иконки:

Значения параметра uType	Вид иконки
MB_ICONEXCLAMATION	Восклицательный знак
MB_ICONINFORMATION	Символ i в круге
MB_ICONQUESTION	Вопросительный знак
MB_ICONSTOP	Предупреждающий знак

Вызов в оконной процедуре при обработке соответствующего пункта меню:

```
MessageBox(hwnd, lpText, lpCaption, MB_YESNO|MB_ICONEXCLAMATION);
```

где `lpText` и `lpCaption` – указатели на строки текста выводимого сообщения и заголовка сообщения соответственно.

Часть 2. Создание диалогового окна средствами WinAPI

Цель работы: научиться создавать Windows-приложения с несколькими диалоговыми панелями в интегрированной среде Visual C++ 6.0, встраивать в окно изображение формата bmp средствами Win32 API, использовать элементы управления в диалоговых панелях.

Задание

Создать приложение на основе каркаса приложения Win32 API, содержащее меню и выполняющее следующие действия:

9. Вывод информации о программе в отдельное окно (окно About);
10. Взаимодействие пользователя и программы с помощью диалогового окна (ввод данных пользователем, вычисления и вывод результатов).

Действия вызываются через меню.

- Окно с информацией о программе About – модальное, содержит кнопку закрытия окна, рисунок в формате bmp и статическую информацию;
- Диалоговая панель содержит окно(а) редактирования для ввода данных и две кнопки - закрытия окна и выполнения простейших вычислений. Вычисления выполняются в соответствии с заданием 1 лабораторной работы № 1. Результат вычислений выводится в дополнительное окно редактирования.

Справочный материал.

Создание диалоговой панели со статической информацией (типа окна About)

1. Добавить новый ресурс – диалоговую панель типа IDD_ABOUT;
2. Настроить свойства диалоговой панели :
   a. установить идентификатор панели IDD_MY_DIALOG,
   b. установить стиль (style) – всплывающий (popup),
   c. добавить заголовок окна (Title bar),
   d. добавить системное меню (System menu),
   e. выбрать рамку для окна (Border) – Dialog Frame,
   f. установить для ресурса русский язык (в окне Resource View).
3. Используя редактор ресурсов, добавить в панель статические строки и кнопку закрытия окна IDOK;
4. Создать оконную процедуру для данного окна, обрабатывающую нажатие кнопки окна About:

```
BOOL CALLBACK AboutDialogProc(HWND hDlg,UINT Message, WPARAM wParam,
                                                        LPARAM lParam)
{
      switch(Message)
      {
      case WM_COMMAND:
            if (LOWORD(wParam) == IDOK)
                  EndDialog(hDlg,0); //Закрытие окна, если была нажата кнопка OK
            return TRUE;
      case WM_SYSCOMMAND:
            if (LOWORD(wParam) == SC_CLOSE)
                  EndDialog(hDlg,0);
                  //Закрытие окна, если была нажата кнопка системного меню
            return TRUE;
      }
      return FALSE;
}
```

5. Вызвать диалоговое окно с помощью команды меню главного окна:

```
DialogBox(hInst, MAKEINTRESOURCE(IDD_ABOUT), hwnd, AboutDialogProc);
```

где hInst – дескриптор приложения (глобальная переменная программы, используемая в различных функциях), IDD_ABOUT – идентификатор диалогового окна, MAKEINTRESOURCE – макрокоманда, преобразующая идентификатор диалогового окна в шаблон диалогового окна, hwnd – дескриптор окна-владельца, AboutDlgProc – оконная процедура диалогового окна.

Вставка рисунка в диалоговую панель

- Скопировать рисунок в формате bmp в папку проекта.

- Импортировать файл с рисунком в проект как элемент ресурса (Выполнить команду Import… из контекстного меню диаграммы ResourceView. Файл станет элементом ресурса с идентификатором IDB_BITMAP1.
- Объявить переменные в соответствующей оконной процедуре (дескриптор окна – hDlg):

PAINTSTRUCT ps; // структура для хранения атрибутов рисования

HDC hDC, BitmapDC; // дескрипторы контекста окна и совместимого с ним контекста в памяти

HBITMAP hBitmap, hOldBitmap; // дескриптор нового и старого растрового изображения

- Выполнить обработку сообщения WM_PAINT (перерисовка окна диалога) – в соответствующей оконной процедуре

```
case WM_PAINT:
      // прорисовка окна диалога
      hDC = BeginPaint(hDlg,&ps); // начало перерисовки
      //создать контекст устройства для вывода картинки:
      BitmapDC = CreateCompatibleDC(hDC);
      // загрузить битовое изображение в память:
      hBitmap = LoadBitmap(hInst,MAKEINTRESOURCE(IDB_BITMAP1));
      // выбрать объект hBitmap в контекст устройства BitmapDC:
      hOldBitmap = (HBITMAP)SelectObject(BitmapDC, hBitmap);
      // вывод- копирует картинку из памяти
      // непосредственно в окно диалога:
      BitBlt(hDC, 0, 0, 141, 200, BitmapDC, 0, 0, SRCCOPY);

      // восстановить прежний объект в контекст BitmapDC:
      SelectObject(BitmapDC, hOldBitmap);
      DeleteDC(BitmapDC); // удалить контекст BitmapDC
      EndPaint(hDlg,&ps);
      DeleteObject(hBitmap);//удалить битовое изображение из памяти
      break;
```

Использование окна редактирования.

1. Создать диалоговую панель IDD_DIALOG1 для работы с окном редактирования;
2. Отредактировать диалоговую панель с помощью редактора ресурсов – добавить окно редактирования и кнопки;
3. Создать оконную процедуру MyDialogProc для диалоговой панели:

Объявить дескриптор для окна редактирования

```
static HWND hEdit;
```

Объявить строковую переменную (буфер) для хранения текста, помещённого в окно редактирования

```
char Buffer[80];
```

Включить в процедуру диалогового окна обработку события WM_INITDIALOG – инициализацию диалога:

```
.............
case WM_INITDIALOG:
   hEdit = GetDlgItem(<дескриптор окна диалога>,
               <идентификатор элемента управления>); // возвращает
                                                     //дескриптор окна диалога
   SetDlgItemText(<дескриптор окна диалога>,
               <идентификатор элемента управления>,
               <указатель на строку-буфер>); //- выводит в окно
                                  //редактирования текст из буфера
   return TRUE;
...............
```

Включить в процедуру диалогового окна обработку события нажатия кнопки, выполняющей вычисления – WM_COMMAND:

```
   case WM_COMMAND:
   ..................
```

```
if (LOWORD(wParam) == <идентификатор кнопки>)
{
//читает содержимое элемента управления (окна редактирования)
// в буфер
      GetDlgItemText(<дескриптор окна диалога>,
                  <идентификатор элемента управления>,
                  <указатель на строку-буфер>,
                  <максимальная длина строки>);

      <действия над полученными данными и помещение их в буфер>
// Вывод результата в окно редактирования
      SetDlgItemText(<дескриптор окна диалога>,
                  <идентификатор элемента управления>,
                  <указатель на строку-буфер>);
.     .     .
return TRUE;
}
```

Завершить процедуру командой:

```
return FALSE;
```

Вызвать диалоговое окно с помощью команды меню:

```
DialogBox(hInst,MAKEINTRESOURCE(IDD_DIALOG1), hwnd, MyDialogProc);
```

где hInst – дескриптор приложения, IDD_DIALOG1 – идентификатор диалогового окна, MAKEINTRESOURCE – макрокоманда, преобразующая идентификатор диалогового окна в шаблон диалогового окна, hwnd – дескриптор окна-владельца, MyDlgProc – оконная процедура диалогового окна. Тот же результат можно получить, если выполнить преобразование идентификатора ресурса в строку:

```
DialogBox(hInst,(LPCSTR)IDD_DIALOG1, hwnd, MyDialogProc);
```

Функция BitBlt

Функция BitBlt копирует изображение (bitmap) из одного контекста в другой без масштабирования. Вызов функции:

```
BitBlt(hdcDest, xDest, yDest, xWidth, yHeight, hdcSrc, xSrc, ySrc, dwROP);
```

где

hdcDest – контекст устройства – приемник изображения,

xDest, yDest, xWidth, yHeight – определяют модифицируемую прямоугольную область в контексте-приемнике, параметры задаются в логических единицах,

hdcSrc - контекст устройства – источник изображения,

xSrc, ySrc – координаты начала прямоугольника с изображением в контексте – источнике в логических единицах,

dwROP – код графической операции, значение параметра SRCCOPY соответствует копированию изображения без каких-либо преобразований.