7. Работа SY-107. Библиотеки

Цели:
- исследование статических и динамически подключаемых библиотек.

Задачи:
- построение и компоновка статической библиотеки;
- построение DLL;
- неявное связывание с DLL;
- явное связывание с DLL.

Опорные документы:
[1, «Библиотеки»]

7.1. Шаблон проекта

Получите шаблон работы SY-107. Извлеките из архива каталог library в корневой каталог диска C:. Откройте проект. Убедитесь, что целевой платформой является x86 или Win32. В решении четыре проекта: для статической библиотеки onelib, для ее тестирования testlib, для динамической библиотеки onedll, для ее тестирования testdll.

7.2. Статическая библиотека

Проект onelib, модуль onelib.h.

Описываем экспортируемую функцию:

```
// возвращает сумму
int Sum(int, int);
```

Модуль onelib.cpp. В начале модуля указываем информацию об организации, о себе, о проекте, дату начала работы:

```
// onelib.cpp
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Фамилия Имя Отчество
// Системное программирование
// SY-107. Статическая библиотека
// 01.01.2000
```

Ниже описываем собственно функцию:

```
// возвращает сумму
int Sum(int x, int y) {
    return (x + y);
}
```

Компилируем проект, убеждаемся, что в каталоге c:\onelib\Debug появился файл onelib.lib. Это и есть статическая библиотека.

Переходим в тестовый проект testlib.

В начале модуля testlib.cpp указываем информацию об организации, о себе, о проекте, дату начала работы:

```
// testlib.cpp
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Фамилия Имя Отчество
// Системное программирование
// SY-107. Тестирование статической библиотеки
// 01.01.2000
```

Добавляем в проект модуль onelib.lib, в каталог Source Files. Если возникнут какие-то диалоги, закрываем их. В конечном итоге в списке файлов проекта testlib должен появиться файл onelib.lib.

Вызываем Sum в функции main, и убеждаемся, что она суммирует.

Добавим в проект onelib какой-нибудь класс, например, класс point.

В модуле onelib.h описываем интерфейс класса, состоящий из конструктора по умолчанию, метода void move(int, int) и метода int getX(void). Элементами данных являются x и y в закрытой секции.

В модуле onelib.cpp описываем реализацию конструктора и методов.

Перекомпилируем проект onelib.

Переходим в проект testlib, функция main. Описываем точку и вызовы методов move и getX. Компилируем проект. Запускаем программу, убеждаемся, что класс работает.

7.3. Динамически подключаемая библиотека

Проект onedll, модуль onedll.h.

Описываем модификатор экспорта и импорта функций:

```
// модификатор экспорта или импорта
#ifdef ONEEXPORTS
#define ONEDLLAPI __declspec(dllexport)
#else
#define ONEDLLAPI __declspec(dllimport)
#endif
```

Если перед включением этого модуля в другой модуль определен символ ONEEXPORTS, константа ONEDLLAPI получит значение модификатора экспорта __seclspec(dllexport), иначе импорта __seclspec(dllimport).

Ниже описываем собственно импортируемую функцию:

```
// экспортируемая функция
extern "C" ONEDLLAPI int Sum(int x, int y);
```

Модуль onedll.cpp.

В начале модуля указываем информацию об организации, о себе, о проекте, дату начала работы:

```
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Фамилия Имя Отчество
// Системное программирование
// SY-107. DLL
// 01.01.2000
```

Ниже описываем константу ONEEXPORTS, так как библиотека экспортирует, после чего включение заголовочного модуля:

```
#define ONEEXPORTS
#include "onedll.h"
```

Ниже описываем собственно экспортируемую функцию:

```
// возвращает сумму
int Sum(int x, int y) {
    return (x + y);
}
```

Компилируем проект, убеждаемся, что появились файлы onedll.dll и onedll.lib.

Тестовый проект testdll. В начале модуля testdll.cpp указываем информацию об организации, о себе, о проекте, дату начала работы:

```
// testlib/cpp
// ОТИ НИЯУ МИФИ
// 1ПО-00Д
// Фамилия Имя Отчество
// Системное программирование
// SY-107. Тестирование DLL
// 01.01.2000
```

Ниже включаем модуль onedll.h.

Указываем расположение файла onedll.lib (именно lib-файла, а не dll-файла). Для этого в открываем свойства проекта, раздел Linker, подраздел Input, поле Additional Dependencies, нажимаем кнопочку Edit (если нет, то кнопочка с троеточием), вводим строку ..\Debug\onedll.lib.

В принципе, все. После этого в функции main вызываем экспортируемую функцию Sum, убеждаемся, что она суммирует.

Копируем из предыдущего решения класс point и вставляем в это решение, в модуль onedll.h интерфейс, в модуль onedll.cpp реализацию. Для экспорта класса перед именем класса нужно вставить ONELIBAPI. Убеждаемся, что класс экспортируется. Создаем объект и вызываем только move, то есть только конструктор и move связываются.

Открываем FAR, каталог c:\onedll. Вводим команды:

```
dumpbin -exports onedll.dll > dll.txt
dumpbin -imports testdll.exe > testdll.txt
```

Записываем в отчет таблицу экспорта из файла dll.txt, таблицу импорта из файла test.txt (только для onedll). Убеждаемся, что экспортируются и импортируются декорированные имена методов. Вероятно, эту DLL нельзя будет использовать с другим компилятором для экспорта класса point. И без декорации методов тоже нельзя, как сказано в msdn.

В таблице экспорта ordinal предназначено для обратной совместимости с Win16, hint используется операционной системой, RVA показывает смещение функции (relative virtual address).

Теперь обратимся к функции DllMain.

Нужно исследовать, какие события возникают. Для этого в каждое событие вписываем вызов MessageBox, например:

```
case DLL_PROCESS_ATTACH:
    MessageBox(0, "PROCESS_ATTACH", 0, MB_OK);
    break;
```

Компилируем, запускаем, записываем в отчет возникшие сообщения.

Удаляем вызовы MessageBox.

7.4. Явное связывание

Модуль testdll.cpp. Описываем тип EXPFUN:

```
typedef int (*EXPFUN)(int, int);
```

Функция main. У нас есть неявное связывание, поэтому получим дескриптор DLL в переменную hModule типа HMODULE при помощи функции GetModuleHandle. Затем получим адрес функции Sum в переменную addr типа EXPFUN при помощи функции GetProcAddress. Затем вызываем функцию Sum, используя имя addr. Проверки всех значений обязательны!

7.5. Контрольные вопросы и упражнения

1. Поясните различие между статической библиотекой и DLL.
2. Назовите преимущества и недостатки тех и других библиотек.
3. Как к проекту подключается статическая библиотека?
4. Как к проекту подключается DLL?
5. Назовите 4 вида соглашений о вызове функций (методов). Какими модификаторами они задаются?
6. Что такое декорация имен? Как ее избежать для экспорта?
7. Как указывается экспортируемая (импортируемая) функция, класс?
8. В чем различие между явным и неявным связыванием?
9. Назовите порядок поиска DLL.
10. Для чего нужна функция DllMain?
11. Что называется переадресацией вызова?
12. Какие DLL называются известными?
13. Для чего нужна команда dumpbin?
14. Для чего нужна команда rebase?
15. Создайте статическую библиотеку regina.lib, скомпонуйте ее с законченным приложением regina. Единственная функция GetReginaRecordLib должна возвращать новую запись ReginaRecord. В приложении regina есть пункт меню для вызова этой функции.
16. Создайте DLL regina.dll с одной функцией GetReginaRecordDll, которая возвращает новую запись ReginaRecord. В приложении regina есть пункт меню для вызова этой функции. Используйте неявное связывание.