К лабораторной работе № 6

ПРОГРАММИРОВАНИЕ СТАТИЧЕСКИХ И ДИНАМИЧЕСКИХ БИБЛИОТЕК

ВОПРОС 1. РАЗРАБОТКА СТАТИЧЕСКИХ БИБЛИОТЕК

При разработке программ рекомендуется разбивать их на части, которые функционально ограничены и закончены. Например, некоторые функции можно расположить в отдельных ***.cpp** файлах. Такой подход обеспечивает ряд преимуществ:

• обычно сложная программа разбивается на несколько отдельных частей (модулей), которые отлаживаются отдельно и зачастую разными людьми; поэтому в завершении остается лишь собрать готовые модули в единый проект;

• при исправлении в одном модуле не надо снова транслировать (переводить в машинные коды) все остальные (это могут быть десятки тысяч строк);

• при компоновке во многих системах можно подключать модули, написанные на других языках, например, на Паскале (в машинных кодах).

Библиотека объектных файлов − несколько объектных файлов, которые используются для хранения функций и ресурсов отдельно от исполняемого файла. Библиотека содержит символьный индекс, который состоит из названий функций и переменных и т.д., которые содержатся в библиотеке. Это позволяет ускорить процесс компоновки программы, так как поиск функций и переменных в объектных файлах библиотеки происходит намного быстрее, чем поиск в наборе указанных объектных файлов.

Поэтому использование библиотеки позволяет компактно хранить все требуемые объектные файлы в одном месте, и при этом значительно повысить скорость компиляции. Таким образом, можно создавать большие проекты, которые больше не будут отнимать много времени на компиляцию и поиск ошибок. Однако нужно помнить, что не стоит также чересчур разбивать программу, иначе получится несколько десятков файлов, в которых рано или поздно можно запутаться. Рекомендуется в отдельные файлы помещать те функции или классы, с которыми приходится больше всего работать при отладке. После того, как функция будет окончательно отлажена, ее вполне можно перенести в более крупный файл.

Объектные библиотеки по способу использования разделяются на два вида:

- *Статические библиотеки*
- *Динамические библиотеки*

Статическая библиотека – это коллекция объектных файлов, которые присоединяются к программе во время компоновки. Таким образом, статические библиотеки используются только при создании программы. Потом же, при выполнении программы, они участия не принимают, в отличие от динамических библиотек.

Пример 1. Написать программу, вычисляющую площадь квадрата, прямоугольника, треугольника, круга и трапеции. Вид фигуры вводит пользователь с клавиатуры. Создать статическую библиотеку функций подсчета площадей каждой из фигур.

sl.cpp:

```
float kvadrat (float a)
{ return a*a;}
```

s2.cpp:

```
float pryamougolnik (float a, float b)
{ return a*b;}
```

s3.cpp:

```
float treugolnik (float a, float h)
{ return 0.5*a*h;}
```

s4.cpp:

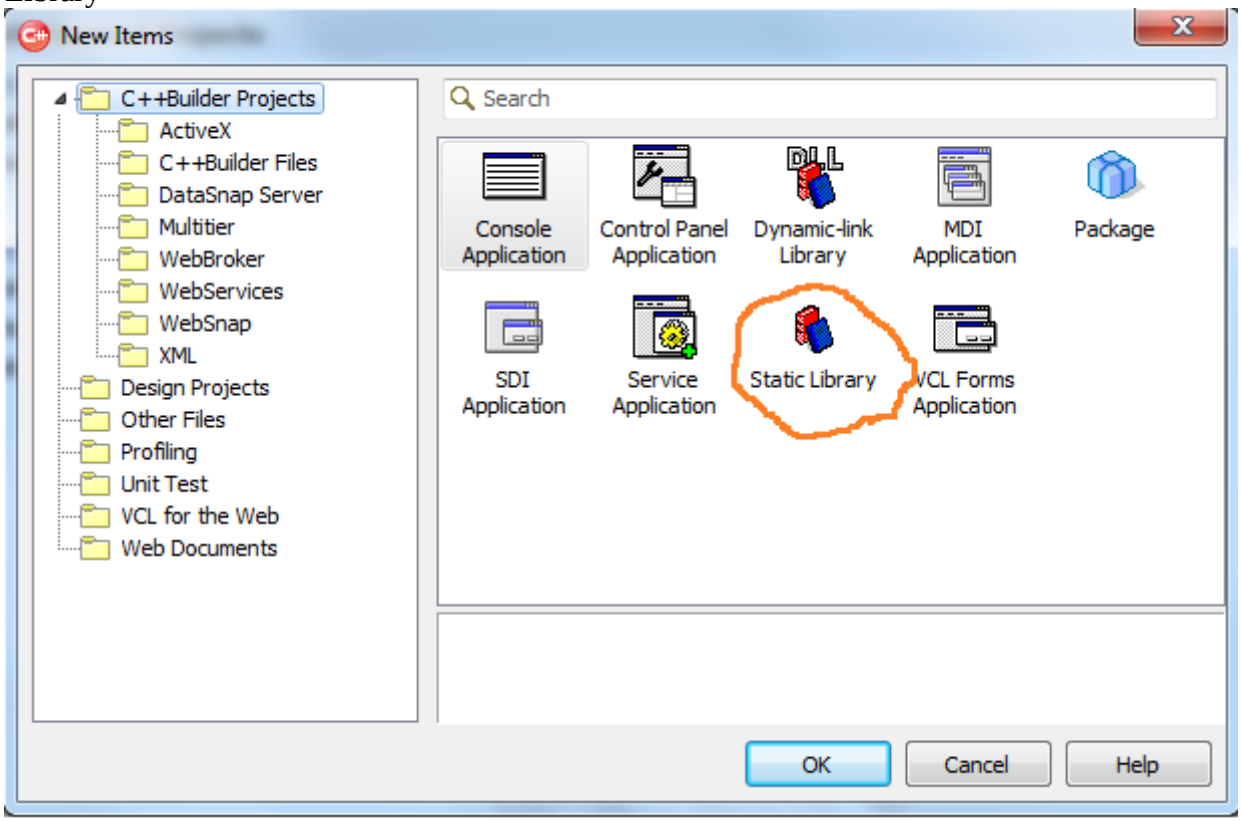
```
float krug (float r)
{ return 3.14*r*r;}
```

s5.cpp:

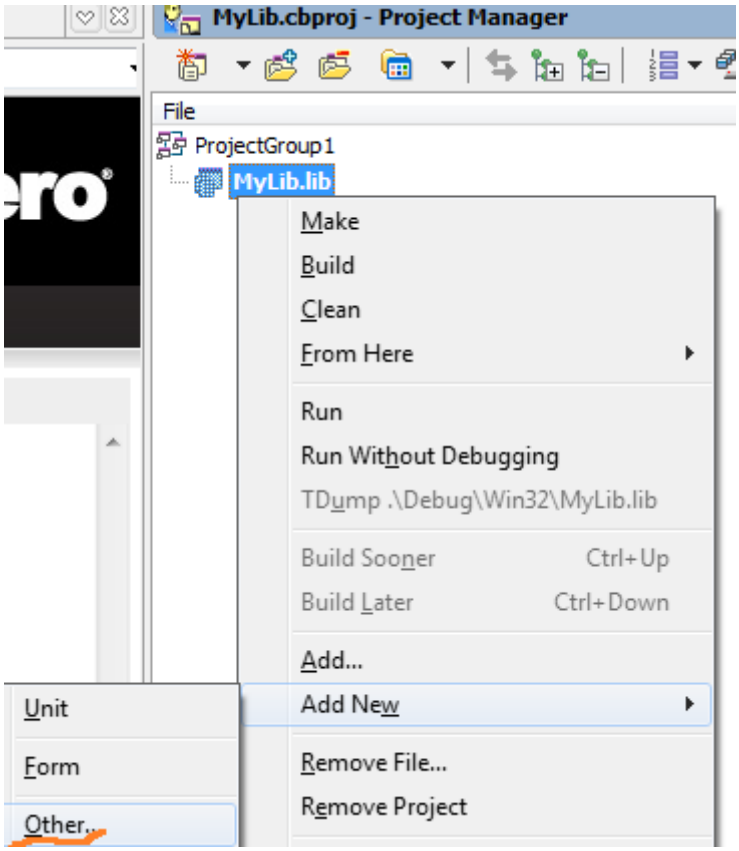
```
float trapeciya (float a, float b, float h)
{ return 0.5*(a+b)*h;}
```

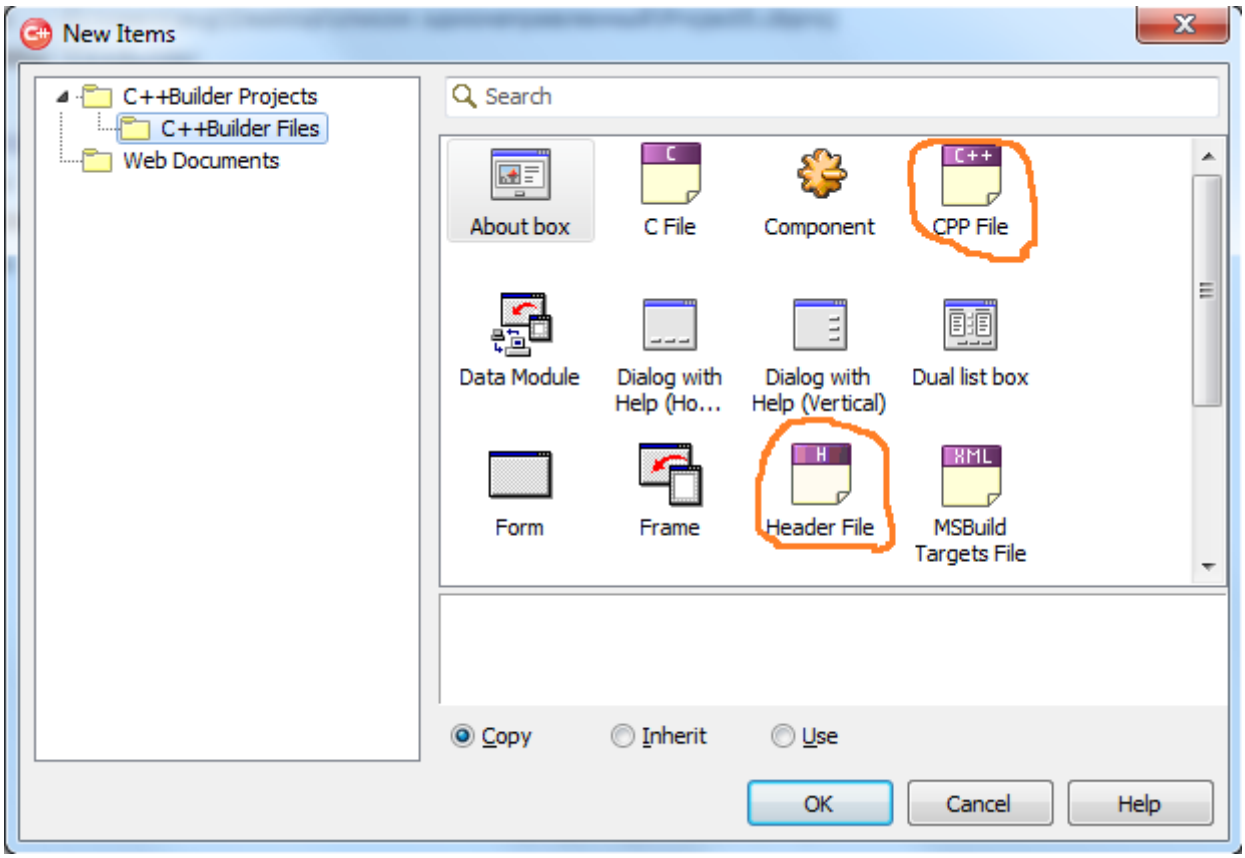
Решение:

Создать проект для разработки статической библиотеки: File/ New/ Other…/Static Library



В окне **Project Manager** изменить имя созданного проекта на MyLib.lib.
С помощью команды контекстного меню **Add New** добавить пять .cpp файлов (**sl.cpp, s2.cpp, s3.cpp, s4.cpp, s5.cpp**) и один .h файл (**squares.h**).





В .cpp файлах разместить определения соответствующих функций. В заголовочном файле – прототипы этих функций:

```
float kvadrat (float a);
float pryamougolnik (float a, float b);
```

```
float treugolnik (float a, float h);
float krug (float r);
float trapeciya (float a, float b, float h);
```

Сохранить весь проект в папке **D:\412\GZ6**.

Создайте библиотеку с помощью команды: **Project/Build MyLib**. В папке проекта **D:\412\GZ6\Debug\Win32** должен появиться файл **MyLib.lib**. Это и есть статическая библиотека.

Пример *2.*

Использовать разработанную ранее статическую библиотеку **MyLib.lib** для вычисления площадей фигур.

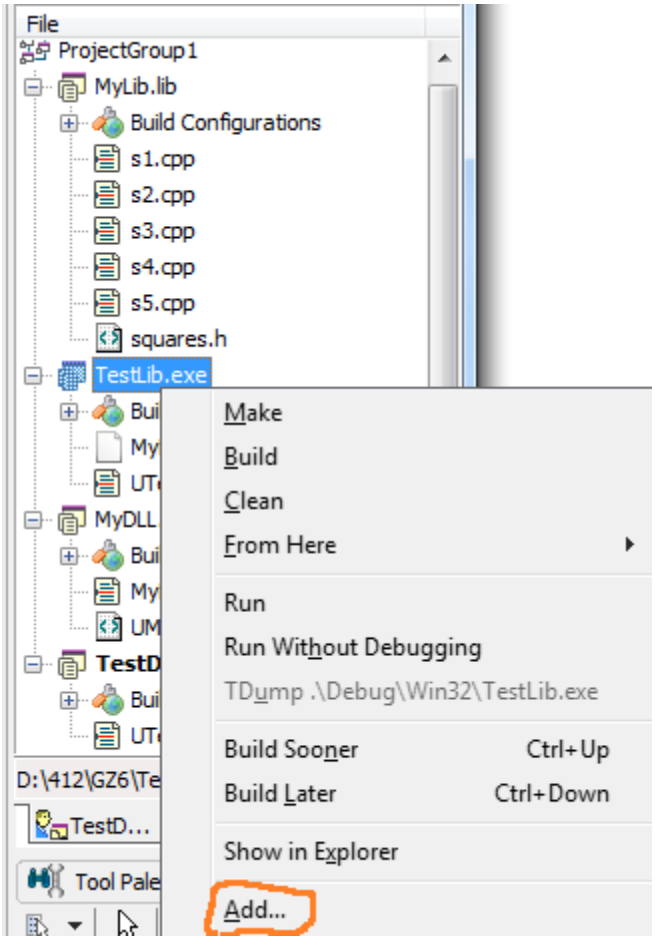
Решение:

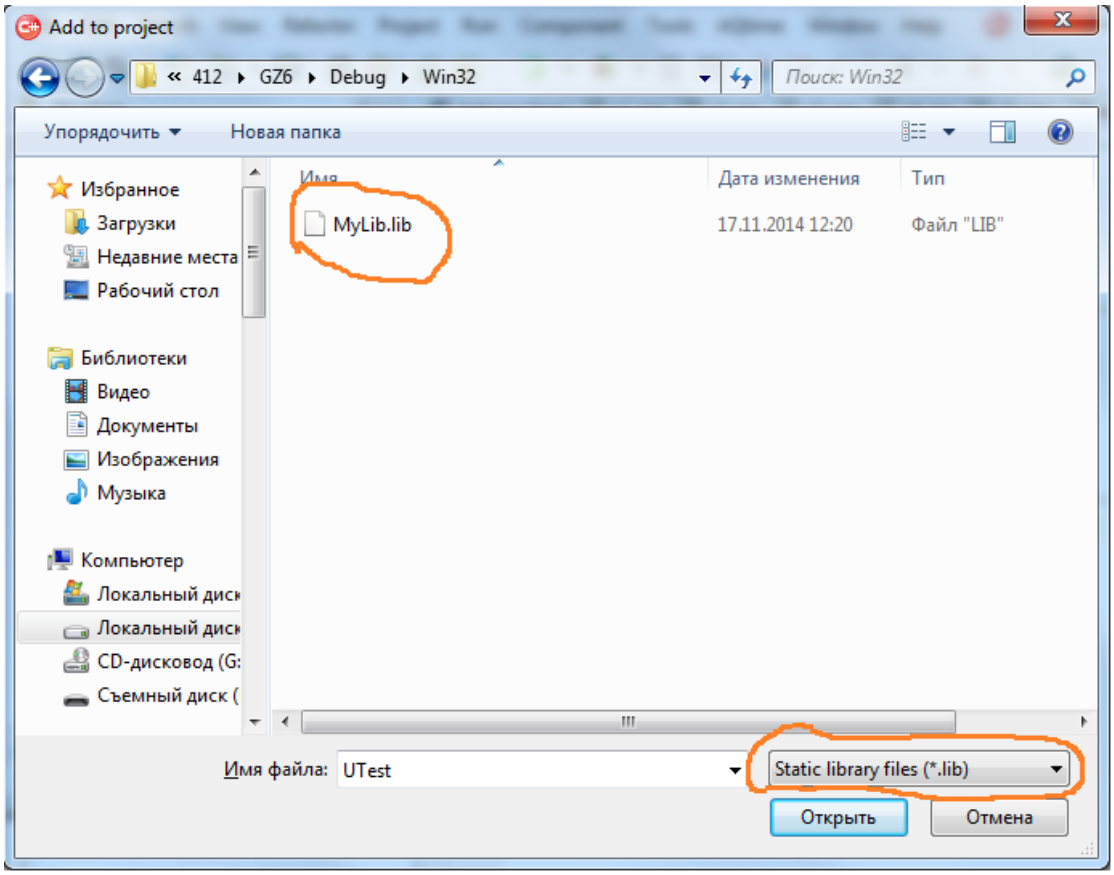
В группу проектов добавить новый проект **TestLib**, **.cpp** файл проекта переименовать в **Utest.cpp**, сохранить в папке **D:\412\GZ6**.

Запустить на выполнение.

Убедиться, что **TestLib.exe** находится в той же папке, что и **MyLib.lib**.

С помощью команды **Add…** контекстного меню к проекту **TestLib** добавить к проекту библиотеку **MyLib.lib**.





В файл **Utest.cpp** добавить директивы препроцессора

```
#include <conio.h>
#include <iostream.h>
//подключаем заголовочный файл созданной библиотеки
//MyLib.lib
#include "squares.h"
```

Использовать функции разработанной библиотеки **MyLib.lib** в функции **int _tmain()**

```
int _tmain(int argc, _TCHAR* argv[])
{
int i;
cout<<"select figure:\n1-kvadrat\n2-pryamougolnic\n3-
treugilnic\n4-krug\n5-trapeciya\n0-vyhod\n";
cin>>i;
switch(i)
{
case 1:cout<<kvadrat(5.5);break;
case 2:cout<<pryamougolnik(6.3,4);break;
case 3:cout<<treugolnik(4.3,8.1);break;
case 4:cout<<krug(5.9);break;
case 5:cout<<trapeciya(9.4,2.1,5);break;
case 0:return 0;
}
getch();
return 0;
}
```

ВОПРОС 2. РАЗРАБОТКА ДИНАМИЧЕСКИХ БИБЛИОТЕК

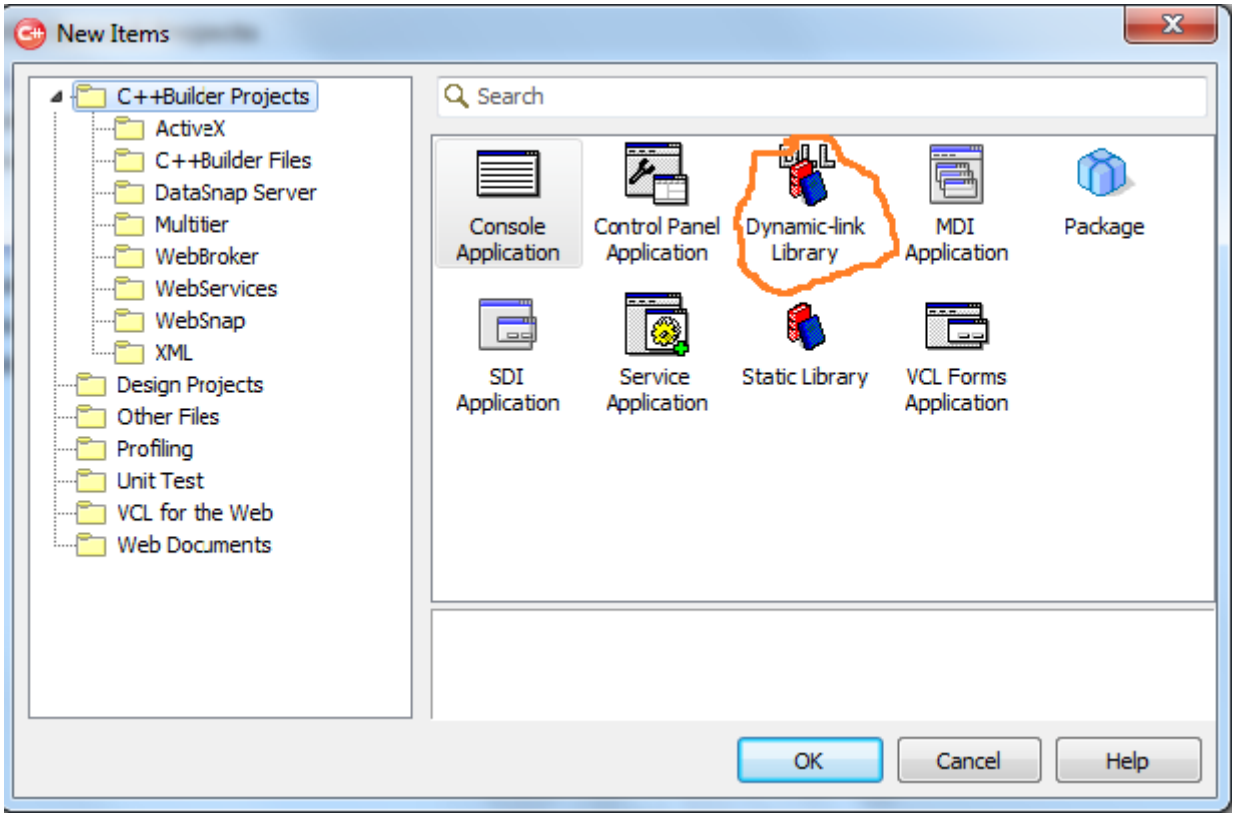
Динамическая библиотека (DLL), с точки зрения программиста, представляет собой библиотеку функций (ресурсов), которыми может пользоваться любой процесс, загрузивший эту библиотеку. Сама загрузка, кстати, отнимает время и увеличивает расход потребляемой приложением памяти; поэтому бездумное дробление одного приложения на множество DLL не рекомендуется.

Однако, если какие-то функции используются несколькими приложениями, то, поместив их в одну DLL, мы избавимся от дублирования кода и сократим общий объем приложений − и на диске, и в оперативной памяти. Можно выносить в DLL и редко используемые функции отдельного приложения.

Загрузившему DLL процессу доступны не все ее функции, а лишь явно предоставляемые самой DLL для "внешнего мира" − т. н. ***экспортируемые***. Функции, предназначенные сугубо для "внутреннего" пользования, экспортировать бессмысленно (хотя и не запрещено). Чем больше функций экспортирует DLL − тем медленнее она загружается; поэтому к проектированию ***интерфейса*** (способа взаимодействия DLL с вызывающим кодом) следует отнестись повнимательнее.

Для экспортирования функции из DLL - перед ее описанием следует указать ключевое слово ***__declspec(dllexport)***, как показано в следующем примере

Пример 3. Разработать динамическую библиотеку следующих функций:
$f1(n,m) = n + m;$
$f1(n,m,k) = (n + m)k;$
$f1(n,m) = n - m;$
В имеющуюся группу проектов добавить проект для разработки динамической библиотеки: File/ New/ Other…/Dynamic-linc Library



Так же как и в предыдущем задании добавить с помощью команды контекстного меню **Add New .h** файл(**UMyDLL.h**). Переименовать файл **.cpp** в **MyDLL. cpp.** Сохранить проект под именем **MyDLL.dll** в папке **D:\412\GZ6**. В **.cpp** файле разместить определения функций. В заголовочном файле – их прототипы.

Определения функций в динамической библиотеке нужно дописать в конце созданного **.cpp** файла. Они имеют следующий вид:

```
extern "C" double __declspec(dllexport) __stdcall f1 (double n, double m)
{return n+m;}

extern "C" double __declspec(dllexport) __stdcall f2 (double n, double m,
double k)
{return (n+m)*k;}

 extern "C" double __declspec(dllexport) __stdcall f3 (double n, double m)
{return n-m;}
```

Прототипы функций в заголовочном файле:

```
extern "C" double __declspec(dllexport) __stdcall f1 (double n, double m) ;

extern "C" double __declspec(dllexport) __stdcall f2 (double n, double m,
double k);

extern "C" double __declspec(dllexport) __stdcall f3 (double n, double m);

//Здесь конструкция __declspec(dllexport)  означает, что функция может
//экспортироваться из библиотеки, то есть может вызываться внешними
//приложениями
//stdcall  – соглашение о вызовах, применяемое в ОС Windows для
//вызова функций WinAPI. Аргументы функций передаются через стек,
//справа налево. Очистку стека производит вызываемая подпрограмма.
```

Создайте библиотеку с помощью команды: **Project/Build MyDLL**. В папке проекта **D:\412\GZ6\Debug\Win32** должен появиться файл **MyDLL.lib**, который можно использовать как статическую библиотеку, а также файл **MyDLL.dll**, применяемый для динамического связывания .

Задание 4.

Использовать разработанную ранее динамическую библиотеку **MyDLL.dll** для определения значения выражения:

$$y = \begin{cases} f1(n,m), \text{ если } m > n \\ f2(n,m,k), \text{если } m = n \\ f3(n,m), \text{ если } m < n \end{cases}$$

Решение:

В группу проектов добавить новый проект **TestDLL**.
.cpp файл проекта переименовать в **Utest1.cpp**.
Все сохранить в папке **D:\412\GZ6**.
Запустить на выполнение.

Убедиться, что **TestDLL.exe** находится в той же папке, что и **MyLib.lib**.
Изменить содержимое **Utest1.cpp**

```
#include <conio.h>
#include <iostream.h>

int _tmain(int argc, _TCHAR* argv[])
{
//загрузка DLL
 HINSTANCE load;
 load=LoadLibrary(L"MyDLL.dll");

//получение указателя на функцию
// pfsum – произвольное имя
typedef double (__stdcall *pfsum)(double,double);
pfsum  f1,f3;

typedef double (__stdcall *pfsum1)(double,double,double);
pfsum1 f2;

// функция API Windows GetProcAddress используется для
//получения указателя на функцию, где
//load – указатель на загруженный модуль DLL
//f1 – имя функции
 f1=(pfsum)GetProcAddress(load,"f1");
 f2=(pfsum1)GetProcAddress(load,"f2");
 f3=(pfsum)GetProcAddress(load,"f3");

 double m,n,k;
 cin>>m>>n>>k;
 if (m>n)   cout<<f1(n,m);
 else  if (m==n)  cout<<f2(n,m,k);
 else  cout<<f3(n,m);


//освобождение DLL
 FreeLibrary(load);
getch();
    return 0;
}
```