МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра вычислительной техники

УТВЕРЖДАЮ
Проректор по учебной работе
_____ О.Г. Локтионова
«15» _____ 2017г.

ПРОЕКТИРОВАНИЕ DLL

Методические указания к лабораторным и практическим занятиям
для студентов направления 09.03.01

Курск 2017

УДК 004.451.8

Составители: Е.Ю. Емельянова, И.Е. Чернецкая

Рецензент
Кандидат технических наук, доцент *Халин Ю.А.*

Проектирование DLL: методические указания к лабораторным и практическим занятиям для студентов направления 09.03.01/ Юго-Зап. гос. ун-т; сост.; Е.Ю. Емельянова, И.Е. Чернецкая. – Курск, 2017. - 15 с.: – Библиогр.: с. 15

Содержат сведения по вопросам создания исполняемых библиотек DLL и подключения их к программам на ЯВУ методами статического и динамического связывания DLL. Теоретические материалы рассмотрены на примерах. Излагаются указания по подготовке и выполнению задания.

Методические указания соответствуют рабочей программе дисциплины «Операционные системы».

Предназначены для студентов направления 09.03.01 очной и заочной форм обучения.

Текст печатается в авторской редакции

Подписано в печать 15.12.17. Формат 60*84 1/16.
Усл. печ. л. 0,87. Уч.-изд. л. 0,79. Тираж 50 экз. Заказ 474. Бесплатно.
Юго-Западный государственный университет.
305040 Курск, ул. 50 лет Октября, 94.

Оглавление

1 ЦЕЛЬ РАБОТЫ

Научиться создавать исполняемые библиотеки DLL и подключать их к программам на ЯВУ методами статического и динамического связывания DLL.

2 ТЕОРЕТИЧЕСКАЯ ЧАСТЬ

2.1 Создание и подключение DLL

Динамически подключаемые библиотеки (Dynamic Link Libraries – DLL) представляют собой программный модуль специального формата, используемый для хранения ресурсов, процедур и функций отдельно от исполняемых файлов. При этом в оперативной памяти хранится одна копия библиотеки DLL. Множество исполняемых файлов могут использовать объекты из DLL, спроецировав библиотеку на своё виртуальное адресное пространство. Таким образом, не нужно дублировать одни и те же коды в разных EXE-файлах – они собраны в общей библиотеке DLL. Получается экономия дискового пространства и оперативной памяти.

Библиотеки DLL дают возможность разрабатывать проект на разных языках программирования: например, DLL, написанная на Visual C++, с успехом может использоваться программами на Delphi, Visual Basic и пр.

Библиотеки DLL повышают гибкость программ. Например, русификация программ (или перевод на иной язык) заключается в том, что создается несколько библиотек DLL, содержащих все используемые программой тексты (надписи, подсказки и т.п.). Каждая из библиотек хранит тексты на своем языке: русском, английском, немецком. Выбор языка сводится к подключению к соответствующей DLL.

На механизме DLL построены все функции Windows API и большинство прикладных программ.

Существует два типа динамических библиотек – исполняемые и библиотеки ресурсов. **Исполняемые DLL** содержат процедуры и функции; **библиотеки ресурсов** – курсоры, значки, окна диалога, строки текста и пр. В принципе, можно создавать и смешанные DLL, но часто бывает удобнее разнести реализацию процедур и используемые приложением ресурсы в разные файлы.

Процедуры и функции, содержащиеся в библиотеке, можно разделить на две группы. Первые используются только внутри самой DLL как вспомогательные; вторые доступны прикладным программам извне. Такие функции называются *экспортируемыми* (exported). Экспортироваться могут также глобальные переменные. В Win32 каждый процесс получает свою копию глобальных переменных. Чтобы несколько процессов использовали совместно какой-либо участок памяти, надо либо создавать

файл, проецируемый на память [1], либо объявить *общий раздел данных* (shared data section) [1;4].

DLL модуль компилируется так, чтобы его можно было загружать (проецировать) по определенному базовому адресу в виртуальном адресном пространстве процесса. Этот базовый адрес задается на этапе разработки DLL, по умолчанию он равен 10000000h. Если программа использует несколько DLL, и их базовые адреса совпадают, то загрузчик копирует и перемещает код DLL по свободному адресу, чтобы не было конфликта.

Список функций, содержащихся в динамической библиотеке, можно просмотреть с помощью утилиты *impdef.exe*, входящей в пакет TASM. Ей передается имя DLL, она формирует файл определений DEF, в котором перечислены имена всех функций библиотеки без описания передаваемых параметров и конвенций вызова.

Имена экспортируемых объектов находятся в *таблице экспортируемых имен* DLL. Экспортируемые объекты идентифицируются извне по их *символьному имени* или по целочисленному *порядковому номеру* (ordinal number)[1]. В таблице экспортируемых имен хранятся также адреса объектов внутри DLL. Впервые загружая DLL, программа (назовем ее клиентской) не знает адресов нужных ей переменных и функций, зато знает их имена или порядковые номера. При подключении к DLL создается таблица, связывающая вызовы из клиентской программы с адресами функций DLL[2].

Прежде чем начать использование какой-либо процедуры или функции, находящейся в динамической библиотеке, необходимо загрузить DLL в оперативную память и связать с процессом. Это может быть осуществлено одним из двух способов: *неявное связывание* (implicit linking) и *явное связывание* (explicit linking)[3]. Оба метода имеют как преимущества, так и недостатки.

Неявное связывание означает, что DLL загружается автоматически при запуске на выполнение использующего ее приложения, и остается связанной с приложением вплоть до окончания его работы. Преимущества: наиболее простой способ; любые экспортируемые из DLL объекты можно использовать так же, как если бы они были описаны внутри модулей программы. Недостатки метода: замедленная загрузка

---

[1] Windows рекомендует использовать символьные имена. Если программа вызывает очень много объектов из DLL, размер EXE-файла может значительно увеличиться за счет хранения строк с именами, в этой ситуации обращение к объектам по их номерам выгоднее.

[2] Можно совершенствовать DLL, не перекомпилируя клиентскую программу. Клиентскую программу нужно перекомпилировать, только если в DLL изменились имена экспортируемых объектов или порядок их следования.

[3] В другой терминологии неявное связывание называется *статическим присоединением*, а явное – *динамическим присоединением*.

программ за счет поиска DLL, их загрузки в память и подключения к ним; если DLL не найдена, программа не запустится; библиотека остается присоединенной, даже если программе больше не нужны объекты из этой библиотеки (а DLL по-прежнему занимает память).

Явное связывание подразумевает, что загрузка динамической библиотеки и отсоединение от нее происходит по мере надобности в процессе работы программы. Таким образом, если DLL не найдена, программа может продолжить работу с ограниченными функциональными возможностями или предложить пользователю найти библиотеку самостоятельно. Этот метод, хотя и обеспечивает наибольшую гибкость, довольно громоздкий.

При явном связывании работа с DLL организуется при помощи функций Windows API:

LoadLibrary, *LoadLibraryEx* – загрузка DLL в адресное пространство процесса;
GetProcAddress – получение адреса процедуры, функции или переменной;
FreeLibrary – отключение от DLL;
FreeLibraryAndExitThread – завершить поток с автоматическим отключением от DLL (часто применяется для окончания потоков, созданных внутри DLL).

Когда от DLL отключается последний процесс, операционная система выгружает библиотеку из памяти. При неявном связывании как подключение, так и отключение программы от динамической библиотеки происходит автоматически. При явном же связывании функцию *FreeLibrary* нужно вызывать самому. Если это забыть сделать, динамическая библиотека будет считаться используемой. Функцию *FreeLibrary* в приложении следует вызывать столько раз, сколько раз вызвалась *LoadLibrary(Ex)*.

Функции загрузки динамической библиотеки

```
HINSTANCE LoadLibrary(PCTSTR pszDLLPathName);
```

и

```
HINSTANCE LoadLibraryEx(
   PCTSTR pszDLLPathName,
   HANDLE hFile,
   DWORD dwFlags);
```

ищут образ DLL среди уже загруженных библиотек и в каталогах (список ниже), и пытаются спроецировать его на адресное пространство вызывающего процесса. Значение типа HINSTANCE, возвращаемое этими функциями[4], содержит адрес виртуальной памяти, по которому спроецирован образ файла. При ошибке функции возвращают NULL. В параметре *pszDLLPathName* передается строка с именем файла DLL. Параметр *hFile* зарезервирован и должен быть равен NULL. Параметр *dwFlags* содержит 0 или комбинацию флагов загрузки:

DONT_RESOLVE_DLL_REFERENCES – не вызывать функцию *DllMain* (о ней позже);
LOAD_LIBRARY_AS_DATAFILE – для загрузки библиотек с ресурсами;
LOAD_WITH_ALTERED_SEARCH_PATH – изменить последовательность поиска DLL.

Стандартно файл библиотеки DLL ищется в следующей последовательности:

1. каталог, содержащий EXE-файл;
2. текущий каталог процесса;
3. системный каталог Windows (например, *windows\system*)
4. основной каталог Windows (обычно *windows* или *winnt*)

---

[4] это значение называется *описателем экземпляров*

5. каталоги, указанные в переменной окружения PATH.

Флаг LOAD_WITH_ALTERED_SEARCH_PATH на 1-м этапе ищет DLL не в каталоге с EXE-файлом, а по пути, заданному в параметре *pszDLLPathName*.

Выгрузка динамической библиотеки происходит функцией

```
BOOL FreeLibrary(HINSTANCE hInstDll);
```

ей передается значение, возвращаемое функциями загрузки.

Адрес экспортируемого объекта из явно загруженной DLL возвращает функция

```
FARPROC GetProcAddress(
   HINSTANCE hInstDll,
   PCSTR pszSymbolName);
```

Параметр *hInstDll* содержит описатель, возвращенный функциями загрузки. Параметр *pszSymbolName* разрешается указывать в форме ANSI-строки с именем объекта, а можно указывать порядковый номер объекта в таблице экспортируемых имен DLL.

Другие функции, которые могут оказаться полезны при работе с DLL:
GetModuleHandle – для проверки, подключена DLL или нет.
GetModuleFileName – получение имени DLL по значению её описателя HINSTANCE.

В каждой динамической библиотеке существует специальная функция *DllMain*, которую вызывает Windows при загрузке, выгрузке библиотеки и подключении или отключении от нее пользовательского процесса. Системы программирования Delphi, C++ Builder, VC++ позволяют программировать функцию *DllMain*, но часто помещают ее в функцию–"обертку" с похожим по смыслу именем.

```
BOOL WINAPI DllMain(
   HINSTANCE hInstDll,
   DWORD fdwReason,
   PVOID fImpLoad);
```

Функция *DllMain* обычно используется для инициализации и очистки ресурсов в конкретных процессах или потоках, для освобождении блоков оперативной памяти. Параметр *fdwReason* указывает причину вызова функции *DllMain*, возможные значения (подробности в [1]):
DLL_PROCESS_ATTACH – библиотека загружается в память и подключается к процессу;
DLL_PROCESS_DETACH – процесс отключается от библиотеки;
DLL_THREAD_ATTACH – в процессе, подключенном к DLL, создается новый поток;
DLL_THREAD_DETACH – уничтожается поток процесса.

Через параметр *hInstDll* передается описатель экземпляра DLL. Параметр *flmpLoad* равен 0, если библиотека загружена явно, и не равен нулю, если загрузка неявная.

Функция *DllMain* должна возвращать TRUE при отсутствии ошибки (влияет только на вызов с параметром DLL_PROCESS_ATTACH).

В лабораторной работе *DllMain* программировать нет необходимости.

При подключении DLL к исполнимому файлу происходит следующее. Сама библиотека DLL существует в оперативной памяти в одном экземпляре. Windows запускает каждый EXE-шник в изолированном адресном пространстве виртуальной памяти. При подключении DLL к программе образ библиотеки проецируется в адресное пространство нужного процесса. Адрес, с которого проецируется образ, для всех EXE-шников фиксирован и прописывается в самой библиотеке DLL при ее компиляции. По умолчанию он равен 4000 0000*h*. Если его не менять, то может возникнуть неприятная ситуация, когда две DLL-ки проецируются на один и тот же адрес. Поэтому при разработке библиотеки желательно задавать иной адрес (больше того, что по умолчанию) в диапазоне от 4000 0000h до 7FFF FFFFh, в адресе должно быть 4 младших нуля (!!!).

В Delphi и C++ Builder адрес библиотеки устанавливается через меню Project/ Options/ Linker/ **ImageBase**. В Delphi его также можно изменить директивой

{$IMAGEBASE $xxxx0000} xxxx – это числа

В Visual C++ 6.0 устанавливается через меню Project Settings/ **BaseAddress**.

2.2 Создание библиотеки DLL

➢ в *Delphi*:

1. Закройте все проекты в Delphi.
2. Выберите пункт меню File/ New/ Other… Щелкните на значке **DLL Wizard**.
3. В открывшемся окне библиотеки наберите:

   3.1) В первой строке замените стандартное имя библиотеки **Project1** на новое имя.

   3.2) Перед словами *begin… end* наберите ваши подпрограммы.

   3.3.) Перед словами *begin… end* наберите директиву экспорта

   ```
   exports  список_имен_эксопртируемых_подпрограмм;
   ```

   чтобы имена подпрограмм были «видимы» за пределами библиотеки.

Например, текст библиотеки будет выглядеть так (цветом выделено то, что добавилось):

```
library MyDLL;
uses
```

```
        SysUtils, Classes, Forms, Windows;
var     если нужны переменные - описываются здесь;

procedure MyProc(var x,y,z: Integer); stdcall;
begin
     z := x+y;
end;

exports
        MyProc;
begin
end.
```

4. Сохраните проект библиотеки под именем, которое вы записали в п.3.1, и скомпилируйте (запускать по F9 смысла нет, так как DLL – это не исполнимый файл). В результате должен появиться файл с расширением DLL.

➢ в **C++ *Builder***:

1. Закройте все проекты.
2. Выберите пункт меню File/ New/ Other… Щелкните на значке **DLL Wizard**.
3. В открывшемся окне библиотеки после всех стандартных строк наберите свои функции

```
extern              "C"                 __declspec(dllexport)
ИмяФункции(параметры)
{
тело функции
};
```

Директива `extern "C" __declspec(dllexport)` нужна, чтобы имена функций были доступны из библиотеки.

Если `extern "C"` не указывать, то компилятор C++ сгенерирует так называемое *расширенное имя* функции (decorated name), которое включает имя класса, имя функции, типы параметров. Расширенное имя нельзя использовать в других языках, кроме C++. Чтобы библиотеку DLL могли использовать программы, написанные на других языках программирования, нужно запретить искажение имен. Для этого указывается модификатор `extern "C"`.

4. Сохраните проект библиотеки, и скомпилируйте (запускать по F9 смысла нет, так как DLL – это не исполнимый файл). В результате должен появиться два файла, которые будут нужны для подключения к проектам: с расширением .DLL и .LIB.

➢ в ***Visual C++***:

В Visual C++ можно создавать два типа динамических библиотек: обычные DLL и DLL-расширения. DLL-расширения способны экспортировать C++ классы, функции-члены и переопределенные функции. Обычная DLL может экспортировать только обычные функции и глобальные переменные. Внутри обычной DLL можно использовать классы C++ и MFC, но не экспортировать их. В лабораторной работе мы будем делать обычную DLL.

Объявления функций делаются так же, как в C++ Builder.

2.3 Неявное связывание DLL

➢ в ***Delphi***:

1. Создайте приложение (оконное или консольное), к которому будем подключать DLL.
2. Сохраните проект.
3. В папку с проектом скопируйте файл динамической библиотеки DLL.
4. В тексте программы, в секции описания процедур и функций, наберите заголовки подпрограмм из библиотеки DLL. Вместо тела подпрограммы записывается директива импорта `external 'ИмяБиблитоеки.dll'`. Например,

```
procedure MyProc(var x,y,z: Integer); stdcall; external 'MyDll.dll';
```

И так для каждой подпрограммы, импортируемой из библиотеки.

5. Теперь в тексте можно вызывать процедуры и функции из библиотеки.
6. Сохраните и скомпилируйте проект.

Для **отладки DLL:**

1. Откройте проект динамической библиотеки.
2. В меню выберите пункт Run/ Parameters / вкладку Local. В параметре *Host Application* укажите путь к EXE-шнику, который подключает вашу библиотеку.
3. В тестируемой подпрограмме установите точку останова.
4. Запустите библиотеку на выполнение (F9). Будет запущен EXE-шник, и в момент вызова вашей процедуры или функции сработает точка останова.

Если вы отлаживаете ассемблерные коды, то для показа окна отладчика вызовите из главного меню View/ Debug Windows / CPU, и для сопроцессора View/ Debug Windows / FPU.

➢ в **C++ *Builder***:

1. Создайте приложение (оконное или консольное), к которому будем подключать DLL.
2. Сохраните проект.
3. В папку с проектом скопируйте файлы DLL и LIB из папки с библиотекой.
4. Подключите библиотечный файл LIB к проекту. Это можно сделать в окне Project Manager, выбрав в контекстном меню команду Add, или в файл проекта вписать строку

```
USELIB("имя_библиотеки.dll")
```

5. В текст программы добавьте заголовки импортируемых из библиотеки DLL функций.

```
extern "C" __declspec(dllimport)
ИмяФункции(параметры);
```

И так для каждой подпрограммы, импортируемой из библиотеки.

5. Теперь в тексте можно вызывать процедуры и функции из DLL.

6. Сохраните и скомпилируйте проект.

Отладка DLL выполняется так же, как в Delphi.

Замечание: бывает удобно все объявления экспортируемых из DLL функций объединить в единый заголовочный файл *h*, и подключать его (меняя, где нужно, *dllexport* на *dllimport*) во все проекты, вызывающие эту DLL и в саму DLL. Замена слов *dlexport* на *dllimport* при объявлении функций реализуется за счет макроопределений. Пример заголовочного файла *h*:

```
#ifdef __DLL__
#define DLL_SPEC extern "C" __declspec(dllexport)
#else
#define DLL_SPEC extern "C" __declspec(dllimport)
#endif
// далее определяются переменные и функции,
например
DLL_SPEC int MyVariable;
DLL_SPEC int MyFunc(int x);
```

Если определен идентификатор __DLL__, макроопределение DLL_SPEC раскрывается через *dllexport*, в противном случае – через *dllimport*. Таким образом, при включении заголовочного файла в проект DLL, нужно определить идентификатор __DLL__, а при включении заголовочного файла в обычную программу – не нужно. То есть, первой строкой DLL должна идти

```
#define __DLL__
```

и еще директива `#include "имя_заголовочного_файла.h"`.

Для **явного связывания** в текст клиентской программы LIB-файл и объявления импортируемых функций включать не нужно. Подключение и работа с DLL выполняется через вызовы *LoadLibrary*, *GetProcAddress* и *FreeLibrary*. Функция *LoadLibrary* пытается отыскать и загрузить динамическую библиотеку. Ей передается строка, содержащая путь и имя файла DLL. Функция возвращает описатель экземпляра типа HINSTANCE, который используется в вызовах *GetProcAddress*. После окончания работы программы следует отключиться от DLL, вызвав функцию *FreeLibrary*.

Пример: допустим, что DLL экспортирует функцию так:

```
extern "C" __declspec(dllexport) double SquareRoot(double x);
```

Упрощённый пример, без проверок на ошибки, взят из [3]:

```
//объявление указателя на функцию
void (__stdcall *MyFunction)(HWIND);
//загрузка DLL
HINSTANCE dllInstance=LoadLibrary("mylib.dll");
//получение адреса функции
MyFunction=(void(__stdcall*)(HWIND)GetProcAddress(dllInstance,"MyFunc
");
//вызов функции
MyFunction(Application->Handle);
//выгрузка DLL
FreeLibrary(dllInstance);
```

В Visual C++ можно использовать проверки корректности выполнения, например:

```
typedef double (SQRTPROC)(double);
HINSTANCE hInstance;
SQRTPROC* pFunction;
VERIFY( hInstance=::LoadLibrary("c:\\winnt\\system\\mylib.dll"));
VERIFY(
pFunction=(SQRTPROC*)::GetProcAddress((HMODULE)hInstance,"SquareRoot"
));
```

```
double d=(*pFunction)(0.225); // вызов функции из DLL
```

3 ПОРЯДОК ВЫПОЛНЕНИЯ РАБОТЫ

Индивидуальным вариантом определяется функция двух переменных *f(x,y)* и заголовок подпрограммы для её вычисления.

Вид функции *f(x,y)* и заголовок подпрограммы берется из п. 5.

1. Создать динамическую библиотеку DLL, которая содержит одну экспортируемую подпрограмму для вычисления функции *f(x,y)*
2. Создать приложение (оконное или консольное), которое подключает DLL **статически**. Числа *x* и *y* вводятся пользователем с клавиатуры. Результаты вычисления *f(x,y)* выводятся на экран.
3. Преобразовать приложение на ЯВУ так, чтобы библиотека DLL подключалась **динамически**. Предусмотреть обработку ошибок: 1) если приложение не находит DLL, предложить пользователю указать путь к библиотеке самостоятельно; 2) если в библиотеке DLL нет функции с нужным именем (допустим, имя библиотеки случайно совпало), вывести сообщение, что DLL не та, что надо.

4 СОДЕРЖАНИЕ ОТЧЕТА

После выполнения лабораторной работы должно быть три программных модуля:
1) библиотека DLL; 2) приложение со статическим подключением DLL; 3) приложение с динамическим подключением DLL.

Язык программирования высокого уровня – любой, по выбору, под Win32.

1. Индивидуальное задание.
2. Текст библиотеки DLL.
3. Текст приложения с динамическим подключением DLL.

5 ВАРИАНТЫ ЗАДАНИЙ

Номер варианта кодируется четырьмя цифрами **N1.N2.N3.N4**.

Первая цифра (N1) задает вид функции двух переменных *f(x,y)*:

N1	Функция	N1	Функция
1.	$f = \exp(2x - y)$	**16.**	$f = y \cdot \sin(x - \pi/8)$
2.	$f = 5\exp(x + y)$	**17.**	$f = \sqrt{\sin(x/y) + 4}$
3.	$f = -\exp(2xy)$	**18.**	$f = x - \sin(\pi \cdot y/3)$
4.	$f = \cos(2xy - 1)$	**19.**	$f = -\cos(x + y)/2$

N1	Функция	N1	Функция
5.	$f = 3\sin x + 2\cos y$	**20.**	$f = tg(\pi/9)\cdot x + y$
6.	$f = x^2 - xy/3 - y^2$	**21.**	$f = x\cdot 2^{y-1} + 0.01$
7.	$f = 1 + \sqrt{\lvert tg(x/y)\rvert}$	**22.**	$f = -2{,}5\cdot\sqrt{x^2+y^2}$
8.	$f = \dfrac{x^2+y^3}{\sqrt{2}}$	**23.**	$f = \dfrac{x^2-y^2}{\ln 2}$
9.	$f = \begin{cases} y - 0.01x; & x \le 0; \\ \sin\sqrt{x}, & x > 0 \end{cases}$	**24.**	$f = \begin{cases} y\cdot\ln\lvert x\rvert, & x \ne 0; \\ 1, & x = 0 \end{cases}$
10.	$f = \begin{cases} 2x - y, & x > 0; \\ \cos x, & x < 0 \end{cases}$	**25.**	$f = \begin{cases} \sin(\pi/y); & y \ne 0; \\ \sqrt{\lvert x\rvert}, & y = 0 \end{cases}$
11.	$f = \begin{cases} \log_y\lvert x\rvert, & y > 0 \text{ и } x \ne 0; \\ (y-0.1)^{-1}, & y \le 0 \text{ или } x = 0 \end{cases}$	**26.**	$f = \begin{cases} y, & x \le 0; \\ \exp x, & x > 0 \end{cases}$
12.	$f = \begin{cases} 3y - \sqrt{x}, & x \ge 0; \\ 2^y/x, & x < 0 \end{cases}$	**27.**	$f = \begin{cases} x^y, & y > 0.1; \\ 0, & y \le 0.1 \end{cases}$
13.	$f = \begin{cases} 0.05\cdot\sin(xy), & xy \ge 0; \\ \dfrac{x-y}{xy}, & xy < 0 \end{cases}$	**28.**	$f = \begin{cases} xy/3, & y > 1; \\ x\cdot e^y, & y \le 1 \end{cases}$
14.	$f = \begin{cases} \sqrt{\sin(\pi y)+1}, & \pi y > x; \\ -1, & \pi y \le x \end{cases}$	**29.**	$f = \begin{cases} 1/(x-y), & x \ne y; \\ -\cos x\cdot\sin x & x = y \end{cases}$
15.	$f = -\log_2\lvert x^2/(y-1)\rvert$	**30.**	$f = \begin{cases} \ln\lvert x+y\rvert, & x+y \ne 0; \\ 0, & x+y = 0 \end{cases}$

Вторая цифра определяет заголовок подпрограммы (если лабораторная работу будете делать на C++, то процедуру преобразовывайте в функцию):

Значение **N2**	Заголовок подпрограммы
1	Function F(X,Y : Type1) : Type2
2	Function F(X : Type1; Y : smallint) : Type2
3	Function F(X : Type1; Y : longint) : Type2
4	Procedure P(X,Y : Type1; var F : Type2)
5	Procedure P(X : Type1; Y : smallint; var F : Type2)
6	Procedure P(X : Type1; Y : longint; var F : Type2)

Цифры **N3** и **N4** кодируют типы *Type1* и *Type2* соответственно:

Значения **N3, N4**	Тип *Delphi, C++ Builder*	Тип *Visual C++*
1	single	float

2	double	double
3	extended	long double

СПИСОК ЛИТЕРАТУРЫ

1. Пахмурин Д.О. Операционные системы ЭВМ [Электронный ресурс]: Учебное пособие. – Томск: Томский государственный университет систем управления и радиоэлектроники, 2013. – 254с. Режим доступа: https://biblioclub.ru/index.php?page=author_red&id=175413
2. Дж. Рихтер. Windows для профессионалов. – 3-е изд.: М: Русская редакция. 1993; 4-е изд: СПб: Русская редакция, 2001. – 752 с.
3. Д. Круглински, С. Уинглоу, Дж. Шеферд. Программирование на Microsoft Visual C++ 6.0 для профессионалов. – СПб: Питер, М: Русская редакция, 2001. – 864 с.
4. Архангельский А.Я. Программирование в C++ Builder 5. – М: БИНОМ, 2000. – 1152 с.
5. Вильямс А. Системное программирование в Windows 2000 для профессионалов. – СПб.: Питер, 2001 – 624 с.