Разработка и использование dll-библиотек

<http://natalia.appmat.ru/c&c++/dll.html>

Dll-библиотека – это набор подпрограмм, которые могут использоваться приложением или другой dll-библиотекой. Так же как и модули языка Паскаль dll-библиотеки могут содержать разделяемый код или ресурсы. Но в отличие от модулей dll-библиотека является отдельно компилируемым исполняемым кодом, который подключается к вызывающему приложению не во время компиляции, а во время работы приложения. Это означает, что библиотека может отсутствовать на компьютере во время компиляции приложения, но в тоже время это означает, что во время компиляции нельзя проверить возможность импорта подпрограммы.

**1. Соглашения о вызовах**

При вызовах процедур и функций каждая программа использует определённые *соглашения о вызовах*. Эти соглашения определяют следующие параметры вызова:

* порядок передачи параметров, в том числе использование регистров;
* порядок очистки стека;
* преобразование имени подпрограммы.

При разработке и использовании dll-библиотек, особенно если библиотека и использующая её программа написаны на разных языках программирования, необходимо следить за тем, чтобы при реализации и вызове подпрограмм использовались одни и те же соглашения о вызовах.

**1.1. Соглашения о вызовах языка С++**

Основными соглашениями о вызовах в языке С++ являются **\_cdecl**, **\_stdcall**, **\_fastcall** и **\_thiscall**. По умолчанию используется **\_cdecl**. Модификатор, определяющий, соглашение о вызовах записывается после типа результата функции:

void \_stdcall f() { … }

* Все перечисленные соглашения о вызовах языка С++ подразумевают передачу параметров функции через стек справа налево, т.е. первым в стек кладётся самый правый (последний) параметр.
* При использовании соглашения о вызовах **\_fastcall** первые два параметра, размер которых не больше двойного слова, передаются через регистры ECX и EDX. Остальные параметры передаются через стек.
* Соглашение о вызовах **\_thiscall** используется для функций-членов класса: указатель this передаётся через регистр ECX, а остальные параметры – через стек.
* Соглашение о вызовах **\_cdecl** подразумевает, что очистка стека осуществляется вызывающей программой. Остальные соглашения о вызовах подразумевают, что очистка стека осуществляется подпрограммой.
* Все перечисленные соглашения о вызовах подразумевают использование регистров для передачи результата функции. Если размер результата не превышает двойного слова, результат передаётся через регистр EAX. Если размер результата не превышает 8 байт, результат передаётся через регистровую пару EDX:EAX. В противном случае через регистр EAX передаётся адрес результата функции.

|  |  |  |  |
| --- | --- | --- | --- |
| **Соглашение о вызовах** | **Порядок передачи параметров** | **Очистка стека** | **Использование регистров** |
| *\_cdecl* | Справа налево | Вызывающей программой | EAX или EDX:EAX для передачи результата. |
| *\_stdcall* | Справа налево | Подпрограммой | EAX или EDX:EAX для передачи результата. |
| *\_fastcall* | Справа налево | Подпрограммой | ECX и EDX для передачи двух первых параметров. EAX или EDX:EAX для передачи результата. |
| *\_thiscall* | Справа налево | Подпрограммой | ECX для передачи указателя *this*. EAX или EDX:EAX для передачи результата. |

При использовании соглашения о вызовах **\_cdecl** генерируемый код получается больше, чем при использовании соглашения о вызовах **\_stdcall**, т.к. после каждого вызова функции необходимо вставлять команды очистки стека. Однако это соглашение о вызовах позволяет реализовывать и вызывать функции с переменным числом параметров.

**1.2. Преобразование имён**

Компоновщик использует *преобразованные* (*decorated*) имена функций. Для языка С++ в такое имя добавляется информации о классе или пространстве имён, которому принадлежит функция, о типах параметров и результата, о соглашении о вызовах.

|  |  |
| --- | --- |
| **Функция** | **Преобразованное имя** |
| int a(char) {int i = 3; return i; } | ?a@@YAHD@Z |
| void \_\_stdcall b::c(float) { } | ?c@b@@AAGXM@Z |

Для языка С способ преобразования имени зависит от соглашения о вызовах.

|  |  |
| --- | --- |
| **Соглашение о вызовах** | **Преобразование имени** |
| *\_cdecl* | Перед именем функции добавляется знак подчёркивания |
| *\_stdcall* | Перед именем функции добавляется знак подчёркивания, а после имени функции – знак @ и количество байт, занимаемых параметрами функции |
| *\_fastcall* | То же, что и для *\_stdcall*, но перед именем функции добавляется знак @ |

|  |  |
| --- | --- |
| **Функция** | **Преобразованное имя** |
| extern "C" int f(char, char \*){ } | \_f |
| extern "C" int \_stdcall f(char, char \*){ } | \_f@5 |
| extern "C" int \_fastcall f(char, char \*){ } | @f@5 |

Знать и использовать преобразованные имена обычно не нужно. Но при использовании в программах на С++ dll-библиотек, написанных на других языках в прототипе функции может потребоваться директива extern "C", которая указывает, что необходимо применять правила компоновки языка С.

**1.3. Соглашения о вызовах языка Паскаль**

В языке Паскаль для указания соглашения о вызовах используются директивы **register**, **pascal**, **cdecl**, **stdcall** и **safecall**. Директива размещается в конце заголовка подпрограммы:

**function** MyFunction(x: real): real; **cdecl**;

По умолчанию подпрограммы в языке Паскаль компилируются с использованием соглашения о вызовах **register**.

* При использовании соглашений о вызовах **register** и **pascal** параметры в подпрограмму передаются слева направо, т.е. самый левый параметр передаётся первым, а самый правый – последним. При использовании соглашений о вызовах **cdecl**, **stdcall** и **safecall** параметры в подпрограмму передаются справа налево.
* При использовании всех соглашений о вызовах кроме **cdecl** подпрограмма удаляет параметры из стека при возврате из подпрограммы. При использовании соглашения о вызовах **cdecl** вызывающая программа удаляет параметры из стека после возврата управления.
* При использовании соглашения о вызовах **register** до трёх параметров могут быть переданы через регистры процессора. При использовании остальных соглашений о вызовах все параметры передаются через стек.
* При использовании соглашения о вызовах **safecall** может возбуждаться исключение ‘firewall’. На платформе Win32 это исключение позволяет передавать уведомления об ошибках между процессами, взаимодействующими через COM-модель.

|  |  |  |  |
| --- | --- | --- | --- |
| **Директива** | **Порядок передачи параметров** | **Очистка стека** | **Использование регистров** |
| *register* | Слева направо | Подпрограммой | Да |
| *pascal* | Слева направо | Подпрограммой | Нет |
| *cdecl* | Справа налево | Вызывающей программой | Нет |
| *stdcall* | Справа налево | Подпрограммой | Нет |
| *safecall* | Справа налево | Подпрограммой | Нет |

Используемое по умолчанию соглашение о вызовах **register** является наиболее эффективным, т.к. во многих случаях позволяет избежать создания стекового фрейма для подпрограммы. Соглашение о вызовах **cdecl** бывает полезно при импорте внешних подпрограмм из динамических библиотек, написанных на языке C++. Соглашения о вызовах **stdcall** и **safecall** обычно используются для внешних подпрограмм. Соглашение о вызовах **pascal** поддерживается для обратной совместимости.

Директивы **near**, **far** и **export** описывают соглашения о вызовах, использовавшихся в 16-битных приложениях. Они не имеют эффекта на платформе Win32 и поддерживаются только для обратной совместимости.

**2. Разработка dll-библиотеки**

**2.1. Структура исходного кода dll-библиотеки**

Исходный код dll-библиотеки имеет схожую структуру во всех языках программирования. Библиотека должна содержать:

* необязательную часть кода, которая отвечает за инициализацию и очистку библиотеки;
* набор подпрограмм библиотеки;
* явное указание, какие подпрограммы должны экспортироваться из библиотеки.

Инициализирующая часть кода оформляется в виде процедуры, которая получает ряд параметров, один из которых указывает на причину обращения. Этот параметр может принимать одно из следующих значений.

|  |  |
| --- | --- |
| **Значение параметра** | **Смысл** |
| DLL\_PROCESS\_ATTACH | Указывает, что библиотека подключена к вызывающему процессу |
| DLL\_PROCESS\_DETACH | Указывает, что библиотека отсоединена от вызывающего процесса |
| DLL\_THREAD\_ATTACH | Указывает, что процесс создаёт новый поток |
| DLL\_THREAD\_DETACH | Указывает, что процесс завершает поток |

Процедура должна возвращать значение *истина* в случае успешного завершения, и значение *ложь* при возникновении ошибки. В этом случае дальнейшие действия прекращаются.

**2.2. Разработка dll-библиотеки на языке С++**

**2.2.1. Принципы разработки**

Для создания dll-библиотеки в приложении Microsoft Visual Studio необходимо создать новый проект Win32 и после задания имени проекта в окне параметров приложения выбрать тип приложения *DLL*. Если остальные параметры оставить без изменений, Microsoft Visual Studio автоматически сгенерирует шаблон библиотеки. Если же создать пустой проект (*empty project*), то инициализирующую часть кода придётся писать самостоятельно. В языке С++ основной функцией dll-библиотеки является функция, которую обычно называют *DllMain*, и которая выглядит следующим образом.

#include <windows.h>

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved)

{ switch (ul\_reason\_for\_call)

{ case DLL\_PROCESS\_ATTACH:

case DLL\_THREAD\_ATTACH:

case DLL\_THREAD\_DETACH:

case DLL\_PROCESS\_DETACH:

break;

}

return TRUE;

}

Функция *DllMain* получает три параметра. Первый является идентификатором библиотеки. Второй определяет причину вызова функции *DllMain* (см. [*раздел 2.1*](http://natalia.appmat.ru/c&c++/dll.html#dll_2_1)). Третий параметр зарезервирован для внутреннего использования в Windows.

Функции, которые будут экспортироваться из dll-библиотеки, должны быть объявлены со спецификацией extern "C" \_\_declspec(dllexport), которая указывает, что функция должна быть доступна приложению, использующему библиотеку, и что имена должны преобразовываться по правилам языка С.

**extern** "C" **\_\_declspec(dllexport)** <тип> <имя> (<параметры>) { … }

**2.2.2. Пример dll-библиотеки на языке С++**

// dllmain.cpp: Defines the entry point for the DLL application

#include <windows.h>

#include <cstdio>

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved)

{

switch (ul\_reason\_for\_call)

{ case DLL\_PROCESS\_ATTACH:

// Будьте внимательны!!! Функция *printf* используется исключительно для примера.

// Библиотека будет работать только в консольном приложении.

// В обычное приложение под Windows подключать эту библиотеку нельзя!

printf("Hello, I am an example library!\n");

break;

case DLL\_THREAD\_ATTACH:

break;

case DLL\_THREAD\_DETACH:

break;

case DLL\_PROCESS\_DETACH:

printf("Goodbye. Example library\n");

break;

}

return TRUE;

}

// ExampleDll.cpp: Defines the exported functions for the DLL application

extern "C" \_\_declspec(dllexport) void ExampleFunction(char \*source, char \*dest)

{ int i;

for (i = 0; source[i]; i++)

dest[i] = source[i];

dest[i] = '\0';

}

После успешной компиляции вы получите файлы *.dll* и *.lib*. Файл с расширением *lib* – это так называемая библиотека импорта, которая требуется при статическом подключении dll-библиотеки.

**2.3. Разработка dll-библиотеки на языке Паскаль**

**2.3.1. Принципы разработки**

Файл с исходным кодом библиотеки аналогичен файлу с исходным кодом программы, за исключением того, что заголовок библиотеки начинается со слова **library**. Таким образом, раздел объявлений библиотеки содержит объявления процедур и функций, а также констант, типов и переменных, необходимых для работы процедур и функций. Раздел операторов, в обычной программе содержащий основной код программы, в dll-библиотеке содержит код инициализации библиотеки.

Для того чтобы подпрограмма из dll-библиотеки была импортирована в другое приложение, она должна быть явно указана в операторе **exports**. Этот оператор имеет следующий синтаксис:

**exports** <элемент1>, …, <элементN>;

Каждый элемент представляет собой имя процедуры или функции (которая должна быть определена до оператора **exports**), за которым должен следовать список параметров (для совместно используемых процедур и функций), и может следовать необязательная спецификация **name**.

Спецификация **name** состоит из директивы **name** и следующей за ней строкой. Если элемент не имеет спецификации **name**, то процедура или функция экспортируется под своим исходным именем с тем же написанием и регистром. Спецификация **name** позволяет экспортировать процедуру или функцию под другим именем. Совместно используемые процедуры и функции желательно экспортировать под разными именами.

**exports**

Divide(x, y: Integer) **name** 'Divide\_Ints',

Divide(x, y: Real) **name** 'Divide\_Reals';

Оператор **exports** может встречаться несколько раз в разделе объявлений.

Для экспортируемых процедур желательно использовать соглашение о вызовах **stdcall** или **cdecl**, если предполагается импортировать подпрограммы в программы, написанные на языке C++.

Библиотека может быть построена из нескольких модулей – в этом случае код библиотеки будет содержать только операторы **uses** и **exports** и код инициализации библиотеки. Оператор **exports** можно поместить в раздел описания интерфейса или раздел реализации модуля. Библиотека, подключающая такой модуль, автоматически экспортирует подпрограммы, перечисленные в операторах **exports** модуля.

Функция *IsLibrary*, объявленная в модуле *System*, позволяет определить, выполняется ли код в приложении или в библиотеке. Функция возвращает значение *истина* при выполнении в коде библиотеки, и *ложь* при выполнении в коде приложения. Это может быть полезно в модуле, поскольку разработчик модуля не знает, куда именно будет подключён модуль.

Раздел операторов в блоке dll-библиотеки содержит код инициализации библиотеки. Эти операторы выполняются один раз при каждой загрузке библиотеки. Они обычно выполняют такие действия как регистрация классов окон и инициализация переменных.

Код инициализации библиотеки может сигнализировать об ошибке, присваивая переменной **ExitCode** ненулевое значение. Переменная **ExitCode** определена в модуле *System* и по умолчанию имеет значение 0. Если код инициализации библиотеки присваивает этой переменной другое значение, библиотека выгружается из памяти и вызывающему приложению передаётся уведомление об ошибке.

Код инициализации библиотеки может также устанавливать процедуру точки входа, которая выполняется при загрузке и выгрузке библиотеки. Переменная **DLLProc** позволяет библиотеке отслеживать вызовы операционной системы. Обычно эта возможность необходима только библиотекам, которые поддерживают многопоточность. В противном случае лучше использовать разделы инициализации и завершения модуля.

Для отслеживания вызовов операционной системы надо определить процедуру обратного вызова, получающую один параметр целого типа. Адрес этой процедуры необходимо присвоить переменной **DLLProc**. Когда процедура вызывается, ей через параметр передаётся одно из значений, определённых в [*разделе 2.1*](http://natalia.appmat.ru/c&c++/dll.html#dll_2_1). В теле процедуры можно указать необходимые действия в зависимости от значения, переданного в процедуру.

**library** Test;

**uses**

SysUtils, Windows;

**var**

SaveDllProc: **procedure**(reason: integer);

**procedure** LibraryProc(reason: integer);

**begin**

**case** reason **of**

DLL\_PROCESS\_ATTACH: writeln('PROCESS\_ATTACH');

DLL\_PROCESS\_DETACH: writeln('PROCESS\_DETACH');

**end**;

**if** @SaveDllProc <> **nil** **then**

SaveDllProc(reason);

**end**;

**begin**

SaveDllProc := **DllProc**;

**DllProc** := @LibraryProc;

**end**.

**2.3.2. Пример dll-библиотеки на языке Паскаль**

**library** InputOutput;

**uses**

SysUtils, Classes, Windows;

**const**

NArrayMax = 100;

**type**

IntArray = **array** [1..NArrayMax] **of** integer;

RealArray = **array** [1..NArrayMax] **of** real;

**var**

SaveDllProc: **procedure**(reason: integer);

**procedure** LibraryProc(reason: integer);

**begin**

**case** reason **of**

{ *writeln* опять-таки работает только в консольном приложении }

DLL\_PROCESS\_ATTACH: writeln('PROCESS\_ATTACH');

DLL\_PROCESS\_DETACH: writeln('PROCESS\_DETACH');

**end**;

**if** @SaveDllProc <> **nil** **then**

SaveDllProc(reason);

**end**;

{ Процедура ввода целочисленного одномерного массива из файла }

**procedure** Get(**var** x: IntArray; **var** n: integer; **var** f: TextFile); **overload**; **stdcall**;

**var**

i: integer;

**begin**

readln(f, n);

**for** i := 1 **to** n **do**

read(f, x[i]);

readln(f);

**end**;

{ Процедура ввода вещественного одномерного массива из файла }

**procedure** Get(**var** x: RealArray; **var** n: integer; **var** f: TextFile); **overload**; **stdcall**;

**var**

i: integer;

**begin**

readln(f, n);

**for** i := 1 **to** n **do**

read(f, x[i]);

readln(f);

**end**;

{ Процедура вывода целочисленного одномерного массива в файл }

**procedure** Put(**const** x: IntArray; n: integer; **name**: string; **var** f: TextFile); **stdcall**;

**var**

i: integer;

**begin**

writeln(f, 'The **array** ', **name**, ' **of** ', n:2, ' elements');

**for** i := 1 **to** n **do**

write(f, x[i]:8);

writeln(f); writeln(f);

**end**;

{ Указываем экспортируемые подпрограммы. Имя Get является совместно используемым, поэтому процедуры экспортируются под разными именами. }

**exports**

Get(**var** x: IntArray; **var** n: integer; **var** f: TextFile) **name** 'GetIntArray',

Get(**var** x: RealArray; **var** n: integer; **var** f: TextFile) **name** 'GetRealArray',

Put;

**begin**

SaveDllProc := **DllProc**;

**DllProc** := @LibraryProc;

**end**.

**2.4. Разработка dll-библиотеки на языке Ассемблер**

**2.4.1. Принципы разработки**

Исходный код на языке Ассемблер для dll-библиотеки должен содержать набор процедур, одна из которых – инициализирующая – схожа по структуре с инициализирующей функцией на языке С++. Процедура должна получать такие же три параметра и помещать в регистр EAX значение -1 (*истина*) для продолжения работы и значение 0 (*ложь*) для завершения работы. Остальные процедуры либо предназначены для экспорта из библиотеки, либо являются вспомогательными.

Однако для указания экспортируемых процедур надо создать так называемый *файл установок модуля*, имеющий расширение *def*. В этом файле указывается имя и описание библиотеки, а также имена и другие параметры экспортируемых процедур.

LIBRARY MyDLL

DESCRIPTION 'MyDLL - пример DLL-библиотеки'

EXPORTS MyFunction

Для каждой процедуры можно указать порядковый номер (после знака @). Этот номер может быть использован для вызова процедуры при динамическом подключении библиотеки (см. [*раздел 3.3*](http://natalia.appmat.ru/c&c++/dll.html#dll_3_3)). На самом деле компилятор присваивает порядковые номера всем экспортируемым объектам. Однако способ, которым он это делает, непредсказуем, если не присвоить эти номера явно. В строке экспорта можно также использовать параметр *NONAME*. Он запрещает компилятору включать имя процедуры в таблицу экспортирования DLL. Иногда это позволяет сэкономить много места в dll-файле. Приложения, использующие библиотеку импортирования для неявного подключения, не «заметят» разницы, поскольку при неявном подключении порядковые номера используются автоматически. Приложениям, загружающим библиотеку динамически, потребуется передавать в функцию *GetProcAddress* порядковый номер, а не имя процедуры.

LIBRARY MyDLL

DESCRIPTION 'MyDLL - пример DLL-библиотеки'

EXPORTS MyFunction @1 NONAME

После создания исходного кода и файла установок модуля нужно обычным способом скомпилировать объектный файл, а компоновщик вызвать с дополнительными параметрами – задать имя файла установок модуля и указать, что необходимо построить именно библиотеку. При вызове компоновщика из директории, в которой находятся файлы *.def* и *.obj* команда вызова будет иметь примерно такой вид:

d:\masm32\bin\link /dll /subsystem:windows /def:MyDLL.def MyDLL.obj

При отсутствии ошибок компоновщик создаст файлы *.dll* и *.lib*. Файл с расширением *lib* – это так называемая *библиотека импорта*, которая требуется при статическом подключении dll-библиотеки.

**2.4.2. Пример dll-библиотеки на языке Ассемблер**

**; Файл ExampleDll.asm**

.686

.model flat, c

option casemap:none

.code

;---------------------------------------------------------------------------

; Инициализирующая процедура библиотеки. Имя процедуры может быть любым

;---------------------------------------------------------------------------

DllMain proc hInstDLL:DWORD, reason:DWORD, reserved1:DWORD

mov eax, -1

ret

DllMain endp

;----------------------------------------------------------------------------

; Процедура, экспортируемая из библиотеки

;----------------------------------------------------------------------------

ExampleProc proc

push ebp

mov ebp, esp

push esi

push edi

mov esi, [ebp + 8]

mov edi, [ebp + 12]

xor eax, eax

L1: mov al, [esi]

mov [edi], al

inc esi

inc edi

test eax, eax

jnz L1

pop edi

pop esi

mov esp, ebp

pop ebp

ret

ExampleProc endp

End DllMain

**; Файл ExampleDll.def**

LIBRARY ExampleDll

EXPORTS ExampleProc

**3. Использование dll-библиотеки**

Прежде чем вызывать подпрограммы, включённые в dll-библиотеку, их надо импортировать в приложение. Это можно сделать двумя способами – путём объявления внешней процедуры или функции или динамически с помощью Win32 API функций. Но в любом случае подпрограммы из dll-библиотеки будут импортированы только во время работы приложения.

**3.1. Поиск dll-библиотек**

Чтобы загрузить dll-библиотеку, операционная система должна найти её. Поиск осуществляется в следующих местах:

* директория, из которой загружено приложение, требующее dll-библиотеку;
* текущая директория;
* системная директория (обычно C:\Windows\System32);
* системная директория для 16-битных приложений (обычно C:\Windows\System);
* Windows-директория;
* директории, указанные в переменной окружения PATH.

Последовательность просмотра директорий может быть разной, это зависит от настроек, но, в общем, просматриваются перечисленные директории до нахождения требуемой библиотеки. Если ни в одной из перечисленных директорий библиотека не найдена, приложение получает уведомление об ошибке.

**3.2. Статическое подключение**

Самый простой способ импортировать процедуру или функцию – тем или иным способом объявить её как внешнюю. Необходимо также указать имя подключаемой dll-библиотеки.

**3.2.1. Статическое подключение dll-библиотеки в программе на языке С++**

Для статического подключения dll-библиотеки в программе на языке С++ необходимо скопировать файл *.lib* в директорию с исходным кодом приложения и указать имя этого файла в настройках компоновщика – *Project → Properties → Configuration Properties → Linker → Input → Additional Dependencies*. Кроме того, поскольку функции должны иметь прототипы, потребуется заголовочный файл с прототипами функций, который необходимо вставить с помощью директивы препроцессора #include в файлы с исходным кодом, в которых предполагается использование функций dll-библиотеки. Заголовочный файл поставляется разработчиком библиотеки. Функции должны быть объявлены со спецификацией extern "C" \_\_declspec(dllimport), которая указывает, что функция будет импортирована из внешнего модуля.

// Использование dll-библиотеки. Пример библиотеки см. в [*разделе 2.2.2*](http://natalia.appmat.ru/c&c++/dll.html#dll_2_2_2)

// ExampleDll.h: Defines the imported functions

extern "C" \_\_declspec(dllimport) void ExampleFunction(char \*source, char \*dest);

// DllTest.cpp

#include "StringDll.h"

#include <cstdio>

void main()

{ char str[256], res[256];

printf("Input a string\n"); gets(str);

ExampleFunction(str, res);

printf("New string - '%s'\n\n", res);

}

**3.2.2. Статическое подключение dll-библиотеки в программе на языке Паскаль**

В языке Паскаль и объявление внешней процедуры, и указание подключаемой библиотеки осуществляется с помощью директивы **external**:

**procedure** <имя> (<список параметров>); **external** <строка, задающая имя библиотеки>;

**function** <имя> (<список параметров>): <тип результата>; **external** <строка, задающая имя библиотеки>;

Если включить в код приложения подобное объявления, библиотека, имя которой задано после ключевого слова **external**, будет загружаться при запуске приложения. Имя импортируемой процедуры или функции будет означать одну и ту же подпрограмму из одной и той библиотеки всё время выполнения приложения.

// Использование dll-библиотеки. Пример библиотеки см. в [*разделе 2.3.2*](http://natalia.appmat.ru/c&c++/dll.html#dll_2_3_2)

**program** LibraryTest1;

{$APPTYPE CONSOLE}

{$I-}

**uses**

SysUtils;

**const**

NArrayMax = 100;

**type**

IntArray = **array** [1..NArrayMax] **of** integer;

**procedure** GetIntArray(**var** x: IntArray; **var** n: integer; **var** f: TextFile); **stdcall**; **external** 'InputOutput.dll';

**procedure** Put(**const** x: IntArray; n: integer; **name**: string; **var** f: TextFile); **stdcall**; **external** 'InputOutput.dll';

**var**

a: IntArray;

n: integer;

f: TextFile;

**begin**

**if** ParamCount < 2 **then**

**begin**

writeln('There are no enough parameters');

readln;

exit;

**end**;

AssignFile(f, ParamStr(1));

Reset(f);

**if** IOResult <> 0 **then**

**begin**

writeln('It is not possible **to** open file ''', ParamStr(1), ''' for reading');

readln;

exit;

**end**;

GetIntArray(a, n, f);

CloseFile(f);

AssignFile(f, ParamStr(2));

Rewrite(f);

**if** IOResult <> 0 **then**

**begin**

writeln('It is not possible **to** open file ''', ParamStr(2), ''' for writing');

writeln('Output is made **to** the screen. Press ENTER');

readln;

AssignFile(f, '');

Rewrite(f);

**end**;

Put(a, n, 'A', f);

CloseFile(f);

readln;

**end**.

**3.2.3. Статическое подключение dll-библиотеки в программе на языке Ассемблер**

Для статического подключения dll-библиотеки в программе на языке Ассемблер потребуется файл *.lib*, а также файл с расширением *inc*, который аналогичен заголовочному файлу языка С++. В исходный код программы нужно включить файл *.inc* с помощью директивы include и файл *.lib* с помощью директивы includelib. После этого можно вызывать процедуры dll-библиотеки.

; Использование dll-библиотеки. Пример библиотеки см. в [*разделе 2.4.2*](http://natalia.appmat.ru/c&c++/dll.html#dll_2_4_2)

; Файл ExampleDll.inc

; Файл содержит прототип процедуры, экспортируемой из библиотеки, что указывает ключевое слово PROTO.

; Ключевое слово CDECL указывает используемое соглашение о вызовах. Далее описываются параметры, которые необходимо передать в процедуру.

; В связи со спецификой передачи параметров в программах на языке Ассемблер это описание является необязательным.

ExampleProc PROTO CDECL :DWORD, :DWORD

; Файл DllTest.asm

.686

.model flat, c

option casemap:none

include \masm32\include\windows.inc

include \masm32\include\kernel32.inc

include ExampleDll.inc

includelib \masm32\lib\kernel32.lib

includelib ExampleDll.lib

.data

source db 'I am a testing string',0

dest db 25 dup(0)

.code

program:

push offset dest

push offset source

call ExampleProc

add esp, 8

push 0

call ExitProcess

end program

**3.3. Динамическое подключение**

Доступ к подпрограммам dll-библиотеки можно получить с помощью Win32 API функций *LoadLibrary*, *FreeLibrary* и *GetProcAddress*. При импорте подпрограмм данным способом, библиотека загружается только при вызове функции *LoadLibrary* и выгружается при вызове функции *FreeLibrary*. Это позволяет уменьшать требуемое количество памяти и запускать приложение, даже если какие-то библиотеки отсутствуют на компьютере.

Функция *LoadLibrary* ищет и загружает dll-библиотеку по переданному ей имени. Поиск осуществляется в тех же директориях, что и при статическом подключении (см. [*раздел 3.1*](http://natalia.appmat.ru/c&c++/dll.html#dll_3_1)). Функция возвращает идентификатор библиотеки, если библиотека была найдена, и значение 0 в противном случае.

Функция *GetProcAddress* осуществляет поиск нужной процедуры в dll-библиотеке. Функция получает идентификатор библиотеки и имя или номер процедуры и возвращает указатель на процедуру, если она была найдена, и 0 в противном случае.

Функция *FreeLibrary* освобождает dll-библиотеку. Однако библиотека может не выгружаться из памяти, если она используется другими приложениями.

При динамическом подключении dll-библиотеки компилятор не может проверить типы и количество передаваемых параметров. Поэтому программист сам должен следить за тем, чтобы процедура получила корректный набор параметров.

Динамическое подключение dll-библиотек более трудоёмко, чем статическое, т.к. нужно самостоятельно вызывать все необходимые функции и проверять возвращаемые ими значения. Кроме того, при динамическом подключении генерируется чуть более объёмный код. Но динамическое подключение является более гибким механизмом. При статическом подключении в случае отсутствия необходимой dll-библиотеки приложение просто не запустится. При динамическом подключении в случае отсутствия dll-библиотеки программист может предпринять какие-либо действия, например, попытаться загрузить другую библиотеку или позволить пользователю работать с теми функциями приложения, которые определены не в отсутствующей библиотеке.

**3.3.1. Динамическое подключение dll-библиотеки в программе на языке С++**

Для использования функций динамического подключения dll-библиотек и соответствующих типов нужно включить заголовочный файл *windows.h*.

// Использование dll-библиотеки. Пример библиотеки см. в [*разделе 2.2.2*](http://natalia.appmat.ru/c&c++/dll.html#dll_2_2_2)

#include <cstdio>

#include <windows.h>

typedef void (\*LibraryFunction)(char \*, char \*); // Объявляем тип для указателя на библиотечную функцию

void main()

{ char str[256], res[256];

HINSTANCE hLib; // Объявляем идентификатор библиотеки

LibraryFunction f; // Объявляем указатель на библиотечную функцию

hLib = LoadLibrary(TEXT("ExampleDll.dll")); // Загружаем библиотеку

if (hLib == NULL) // Проверяем результат загрузки библиотеки

{ printf("Unable to load the library 'ExampleDll.dll'!\n");

return;

}

// Получаем указатель на функцию ExampleFunction и преобразуем его к нужному типу

f = (LibraryFunction)GetProcAddress(hLib, "ExampleFunction");

if (!f) // Проверяем полученный указатель

printf("Unable to find the function 'ExampleFunction'!\n\n");

else

{ printf("Input a string\n"); gets(str);

f(str, res);

printf("'%s'\n\n", res);

}

FreeLibrary(hLib); // Освобождаем библиотеку

}

**3.3.2. Динамическое подключение dll-библиотеки в программе на языке Паскаль**

Функции для динамического подключения dll-библиотек определены в модуле *Windows*.

// Использование dll-библиотеки. Пример библиотеки см. в [*разделе 2.3.2*](http://natalia.appmat.ru/c&c++/dll.html#dll_2_3_2)

**program** LibraryTest2;

{$APPTYPE CONSOLE}

{$I-}

**uses**

SysUtils, Windows;

**const**

NArrayMax = 100;

**type**

IntArray = **array** [1..NArrayMax] **of** integer;

TProcGet = **procedure** (**var** x: IntArray; **var** n: integer; **var** f: TextFile); **stdcall**;

TProcPut = **procedure** (**const** x: IntArray; n: integer; **name**: string; **var** f: TextFile); **stdcall**;

**var**

a: IntArray;

n: integer;

f: TextFile;

handle: integer;

ProcGet: TProcGet;

ProcPut: TProcPut;

**begin**

{ Загружаем dll-библиотеку }

handle := LoadLibrary('InputOutput.dll');

{ Проверяем результат загрузки библиотеки }

**if** handle = 0 **then**

**begin**

writeln('It is not possible to load the library ''InputOutput.dll''');

readln;

exit;

**end**;

**if** ParamCount < 2 **then**

**begin**

writeln('There are no enough parameters');

readln;

exit;

**end**;

AssignFile(f, ParamStr(1));

Reset(f);

**if** IOResult <> 0 **then**

**begin**

writeln('It is not possible to open file ''', ParamStr(1), ''' **for** reading');

readln;

{ Освобождаем библиотеку }

FreeLibrary(handle);

exit;

**end**;

{ Загружаем адрес требуемой процедуры, *handle* – идентификатор, полученный при загрузке библиотеки }

@ProcGet := GetProcAddress(handle, 'GetIntArray');

{ Проверяем результат загрузки адреса процедуры }

**if** @ProcGet = **nil** **then**

**begin**

writeln('It is not possible to get the procedure ''GetIntArray'' from the library ''InputOutput.dll''');

readln;

{ Освобождаем библиотеку }

FreeLibrary(handle);

exit;

**end**;

ProcGet(a, n, f);

CloseFile(f);

AssignFile(f, ParamStr(2));

Rewrite(f);

**if** IOResult <> 0 **then**

**begin**

writeln('It is not possible to open file ''', ParamStr(2), ''' for writing');

writeln('Output is made to the screen. Press ENTER');

readln;

AssignFile(f, '');

Rewrite(f);

**end**;

{ Загружаем адрес другой процедуры }

@ProcPut := GetProcAddress(handle, 'Put');

{ Проверяем результат загрузки адреса процедуры }

**if** @ProcGet = **nil** **then**

**begin**

writeln('It is not possible to get the procedure ''Put'' from the library ''InputOutput.dll''');

readln;

{ Освобождаем библиотеку }

FreeLibrary(handle);

exit;

**end**;

ProcPut(a, n, 'A', f);

CloseFile(f);

{ Освобождаем библиотеку }

FreeLibrary(handle);

readln;

**end**.

**3.3.3. Динамическое подключение dll-библиотеки в программе на языке Ассемблер**

Для динамического подключения используются те же функции с теми же параметрами. Вызов функций осуществляется с учётом специфики языка Ассемблер. Для вызова функций Win32 API необходимо включить файл *kernel32.inc* и подключить библиотеку импорта *kernel32.lib*.

; Использование dll-библиотеки. Пример библиотеки см. в [*разделе 2.4.2*](http://natalia.appmat.ru/c&c++/dll.html#dll_2_4_2)

.686

.model flat, c

option casemap:none

include \masm32\include\kernel32.inc

includelib \masm32\lib\kernel32.lib

.data

source db 'I am a testing string',0

dest db 25 dup(0)

.data?

hLib dd ?

fAddr dd ?

.const

libName db 'ExampleDll.dll',0

fName db 'ExampleProc',0

lnf db 'Library ''ExampleDll.dll'' not found',13,10

fnf db 'Function ''ExampleProc'' not found',13,10

.code

program:

; Получаем идентификатор вывода

push -11

call GetStdHandle

mov hStdOut, eax

; Получаем идентификатор библиотеки

push offset libName

call LoadLibrary

; Проверяем полученный идентификатор

test eax, eax

jz L0

mov hLib, eax

; Получаем указатель на функцию

push offset fName

push hLib

call GetProcAddress

; Проверяем полученный указатель

test eax, eax

jz L1

; Вызываем функцию

push offset dest

push offset source

mov fAddr, eax

call [fAddr]

add esp, 8

; Освобождаем библиотеку

push hLib

call FreeLibrary

push 0

call ExitProcess

L0: push 0

push 0

push 36d

push offset lnf

push hStdOut

call WriteConsoleA

; Задержка

push 1000h

call Sleep

push 0

call ExitProcess

L1: push 0

push 0

push 34d

push offset fnf

push hStdOut

call WriteConsoleA

; Задержка

push 1000h

call Sleep

; Освобождаем библиотеку

push hLib

call FreeLibrary

push 0

call ExitProcess

end program

**4. Использование dll-библиотеки, реализованной на другом языке программирования**

До сих пор мы рассматривали случаи, когда dll-библиотека и вызывающая программа написаны на одном и том же языке программирования. Но иногда возникает необходимость подключить в программу библиотеку, реализованную на другом языке программирования. Никаких принципиальных сложностей тут нет. Надо только следить, чтобы при реализации и вызове процедур использовались одинаковые соглашения о вызовах, а также за соответствием формальных и фактических параметров. Однако в этом случае соответствие параметров будет не формальным. Программист должен понимать, какие типы одного языка соответствуют тем или иным типам другого языка. С простыми типам, например, целыми и вещественными, проблем возникать не должно. А вот сложные, высокоуровневые типы не всегда будут соответствовать друг другу. Например, строки в языках С++ и Паскаль реализуются по-разному.

**4.1. Использование dll-библиотеки, реализованной на языке Ассемблер, в программе на языке С++**

Язык Ассемблер оперирует низкоуровневыми объектами, поэтому никаких несоответствий в типах параметров возникать не должно. Единственное формальное требование, которое можно предъявить в этом случае, – соответствие *размеров* передаваемых данных. Если процедура, написанная на языке Ассемблер, ожидает 4 байта, то можно передать и целое число, и указатель, и даже вещественное число типа *float*. Так что импортируемую функцию в программе на языке С++ можно, в принципе, определить разными способами, причём большинство из них окажется работоспособными. Например, если функция ожидает указатель, можно объявить параметр с типом *int*. Если после этого передать в функцию указатель, преобразованный к типу *int*, программа будет работать без ошибок.

Однако нужно понимать, что каждый передаваемый в процедуру параметр используется определённым образом. И если в процедуру, ожидающую указатель, передать значение, не являющееся адресом реального объекта программы, это приведёт к возникновению ошибок и некорректной работе программы. Таким образом, несмотря на отсутствие формальных требований к типам параметров, смысловое соответствие типов должно сохраняться.

Поскольку библиотеку планируется использовать в программе на языке С++, будем считать, что параметры передаются в обратном порядке, и что вызывающая программа сама освободит стек (т.е. используем команду языка ассемблера *RET* без параметров).

; Файл AsmDll.asm. Кроме этого файла необходимо также создать файл .def и скомпилировать библиотеку, как указано в [*разделе 2.4.1*](http://natalia.appmat.ru/c&c++/dll.html#dll_2_4_1).

.686

.model flat, c

option casemap:none

.code

DllMain proc, hInstDLL:DWORD, reason:DWORD, reserved1:DWORD

mov eax, -1

ret

DllMain endp

;-----------------------------------------------------------------------------------------------------

; Процедура, считающая в одномерном целочисленном массиве сумму элементов, кратных заданному числу.

; Процедура получает указатель на начало массива, количество элементов массива и число.

; Результат возвращается через регистр EAX.

;-----------------------------------------------------------------------------------------------------

SumOfMultiples proc

push ebp

mov ebp, esp

mov ecx, [ebp + 12]

cmp ecx, 0

jle E1

push esi

push edi

mov esi, [ebp + 8]

mov ebx, [ebp + 16]

xor edi, edi

L0: mov eax, [esi]

cdq

idiv ebx

test edx, edx

jnz M1

mov eax, [esi]

add edi, eax

M1: add esi, 4

dec ecx

ja L0

mov eax, edi

pop edi

pop esi

mov esp, ebp

pop ebp

ret

E1: xor eax, eax

mov esp, ebp

pop ebp

ret

SumOfMultiples endp

;---------------------------------------------------------------------------------------------

; Процедура копирования строки. Процедура получает адрес исходной строки и адрес результата.

;---------------------------------------------------------------------------------------------

StrCpy proc

push ebp

mov ebp, esp

push esi

push edi

mov esi, [ebp + 8]

mov edi, [ebp + 12]

xor eax, eax

L1: mov al, [esi]

mov [edi], al

inc esi

inc edi

test eax, eax

jnz L1

pop edi

pop esi

mov esp, ebp

pop ebp

ret

StrCpy endp

End DllMain

// Файл DllTest.cpp. В настройки проекта добавляется файл AsmDll.lib, как указано в [*разделе 3.2.1*](http://natalia.appmat.ru/c&c++/dll.html#dll_3_2_1)

// Объявляем прототипы импортируемых функций в соответствии с назначением передаваемых параметров

// Функции нужно объявить с директивой *extern "С"*, т.к. ассемблер генерирует имена в стиле языка С

extern "C" \_\_declspec(dllimport) int SumOfMultiples(int x[], int n, int y);

extern "C" \_\_declspec(dllimport) void StrCpy(const char \*source, char \*dest);

#include <cstdio>

void main()

{ int a[10] = {1, 2, 3, 5, 7, 10, 15, -6, 8, -21};

char str[100];

printf("%% 2 ( 14) - %3d\n", SumOfMultiples(a, 10, 2));

printf("%% 3 ( -9) - %3d\n", SumOfMultiples(a, 10, 3));

printf("%% 5 ( 30) - %3d\n", SumOfMultiples(a, 10, 5));

StrCpy("Скопируй меня!", str);

printf("%s\n", str);

}

**4.2. Использование dll-библиотеки, реализованной на языке Паскаль, в программе на языке С++**

Базовые типы языков С++ и Паскаль после компиляции (в машинных кодах) реализуются одинаково. Что касается массивов, то массивы языка С++ совместимы со статическими и динамическими массивами языка Паскаль, но не с открытыми массивами.

Соглашение о вызовах, используемое в языке Паскаль по умолчанию, отличается от соглашений о вызовах языка С++. Поэтому для импортируемых процедур надо явно указать соглашение о вызовах **cdecl** или **stdcall**.

Кроме того, компиляторы языка Паскаль не генерируют файл *.lib*, поэтому статическое подключение dll-библиотеки не возможно.

// Файл DllForC.pas. Используем соглашение о вызовах **stdcall**

**library** DllForC;

**type**

IntArray = **array** **of** integer;

RealArray = **array** **of** real;

// Функция, считающая в вещественном массиве сумму элементов, больших заданного числа

**function** Sum(x: RealArray; n: integer; y: real): real; **stdcall**;

**var**

i: integer;

**begin**

result := 0;

**for** i := 0 **to** n - 1 **do**

**if** x[i] > y **then**

result := result + x[i];

**end**;

// Функция, считающая в целочисленном массиве количество элементов, кратных заданному числу

**function** Count(x: IntArray; n: integer; y: integer): integer; **stdcall**;

**var**

i: integer;

**begin**

result := 0;

**for** i := 0 **to** n - 1 **do**

**if** x[i] mod y = 0 **then**

result := result + 1;

**end**;

// Функция, проверяющая, есть ли в целочисленном массиве элементы, равные заданному числу

**function** Check(x: IntArray; n: integer; y: integer): boolean; **stdcall**;

**var**

i: integer;

**begin**

Check := false;

**for** i := 0 **to** n - 1 **do**

**if** x[i] = y **then**

**begin**

Check := true;

break;

**end**;

**end**;

**exports**

Sum, Count, Check;

**begin**

**end**.

// Файл DllTest.cpp

#include <fstream>

#include <iostream>

#include <locale.h>

#include <windows.h>

using namespace std;

const int nmax = 100;

// Типы указателей для функций dll-библиотеки

typedef double (\_stdcall \*F1)(double \*, int, double);

typedef int (\_stdcall \*F2)(int \*, int, int);

typedef bool (\_stdcall \*F3)(int \*, int, int);

// Шаблон функции для ввода массивов разного типа

template <typename T> int ArrayInput(T x[], int \*n, T \*y, char \*fname);

void main(int argc, char \*argv[])

{ double a[nmax];

double ya;

int b[nmax], c[nmax];

int na, nb, nc;

int yb, yc;

HINSTANCE hLib;

F1 f1;

F2 f2;

F3 f3;

setlocale(LC\_ALL, "rus");

if (argc < 4)

{ printf("Недостаточно параметров!\n"); return; }

if (!ArrayInput(a, &na, &ya, argv[1])) return;

if (!ArrayInput(b, &nb, &yb, argv[2])) return;

if (!ArrayInput(c, &nc, &yc, argv[3])) return;

hLib = LoadLibrary(TEXT("DllForC.dll"));

if (hLib == NULL)

{ printf("Unable to load the library 'DllForC.dll'!\n");

return;

}

f1 = (F1)GetProcAddress(hLib, "Sum");

if (!f1)

printf("Unable to find the function 'Sum'!\n\n");

else

printf("Сумма элементов, больших %8.3lf = %8.3lf\n", ya, f1(a, na, ya));

f2 = (F2)GetProcAddress(hLib, "Count");

if (!f2)

printf("Unable to find the function 'Count'!\n\n");

else

printf("Количество элементов, кратных %3d = %3d\n", yb, f2(b, nb, yb));

f3 = (F3)GetProcAddress(hLib, "Check");

if (!f3)

printf("Unable to find the function 'Check'!\n\n");

else

if (f3(c, nc, yc))

printf("Есть элементы, равные %3d\n", yc);

else

printf("Нет элементов, равных %3d\n", yc);

FreeLibrary(hLib);

}

template <typename T> int ArrayInput(T x[], int \*n, T \*y, char \*fname)

{ ifstream f(fname);

if (!f.is\_open())

{ cout << "Невозможно открыть файл '" << fname << "'\n"; return 0; }

f >> \*n;

if (f.fail())

{ cout << "Ошибка чтения из файла '" << fname << "'\n"; f.close(); return 0; }

for (int i = 0; i < \*n; i++)

{ f >> x[i];

if (f.fail())

{ cout << "Ошибка чтения из файла '" << fname << "'\n"; f.close(); return 0; }

}

f >> \*y;

if (f.fail())

{ cout << "Ошибка чтения из файла '" << fname << "'\n"; f.close(); return 0; }

f.close();

return 1;

}

**4.3. Использование dll-библиотеки, реализованной на языке С++, в программе на языке Паскаль**

При использовании в программе на языке Паскаль dll-библиотеки, реализованной на языке С++, также необходимо следить за фактическим соответствием типов и размеров параметров и за используемым соглашением о вызовах. Кроме того, функции dll-библиотеки нужно объявлять с директивой *extern "C"*, т.к. язык С++ использует сложный алгоритм модификации имён, не совместимый с языком Паскаль.

// dllmain.cpp: Defines the entry point for the DLL application.

#include <windows.h>

#include <cstdio>

BOOL APIENTRY DllMain(HMODULE hModule, DWORD ul\_reason\_for\_call, LPVOID lpReserved)

{ switch (ul\_reason\_for\_call)

{ case DLL\_PROCESS\_ATTACH:

printf("Hello, I am a C library!\n");

break;

case DLL\_PROCESS\_DETACH:

printf("Goodbye. C library\n");

break;

case DLL\_THREAD\_ATTACH:

case DLL\_THREAD\_DETACH:

break;

}

return TRUE;

}

// DllForPascal.cpp: Defines the exported functions for the DLL application.

// Функция, считающая в вещественном массиве сумму элементов, больших заданного числа

extern "C" \_\_declspec(dllexport) double Sum(double x[], int n, double y)

{ double s = 0;

for (int i = 0; i < n; i++)

if (x[i] > y)

s += x[i];

return s;

}

// Функция, считающая в целочисленном массиве количество элементов, кратных заданному числу

extern "C" \_\_declspec(dllexport) int Count(int x[], int n, int y)

{ int k = 0;

for (int i = 0; i < n; i++)

if (x[i] % y == 0)

k++;

return k;

}

// Функция, проверяющая, есть ли в целочисленном массиве элементы, равные заданному числу

extern "C" \_\_declspec(dllexport) bool Check(int x[], int n, int y)

{ for (int i = 0; i < n; i++)

if (x[i] == y)

return true;

return false;

}

**program** DllTest;

{$APPTYPE CONSOLE}

**uses**

SysUtils;

**function** Sum(x: RealArray; n: integer; y: real): real; **cdecl**; **external** 'DllForPascal.dll';

**function** Count(x: IntArray; n: integer; y: integer): integer; **cdecl**; **external** 'DllForPascal.dll';

**function** Check(x: IntArray; n: integer; y: integer): boolean; **cdecl**; **external** 'DllForPascal.dll';

**var**

a: RealArray;

b, c: IntArray;

na, nb, nc: integer;

ya: real;

yb, yc: integer;

f: TextFile;

**begin**

// Ввод массивов

...

writeln('Sum of elements > ', ya:6:2, ' = ', Sum(a, na, ya):6:2);

writeln('Count of elements multiple of ', yb, ' = ', Count(b, nb, yb));

**if** Check(c, nc, yc) **then**

writeln('There are elements = ', yc)

**else**

writeln('There is no element = ', yc);

**end**.