Command Line Build Your Own C/C++ Files

Contents

TO MAKE EXECUTABLE

Step 1: Setup your Environment

- Find location of your VC installation. In this sample, the installation locates at: ““C:\Program Files (x86)\Microsoft Visual Studio 14.0\”
- To setup your make environment, you can run the setup file vcvars32.bat.
- I also highly recommend to view the content inside the bat file to understand how are being set. This will help you to understand how you may set your environment differently.

Run this:

```
“C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin\vcvars32.bat
```

Step 2: Compile & Link

- Go to your working folder where your files are.
- To make it : cl [option...] filename... [/link linkoption...]
- e.g. your files are: filesSample.cpp, lib.cpp, sub1.cpp, sub2.cpp
- the following will produce executable name “filesSample.exe”. This is not because filesSamples.cpp is the first one in the list. It is because the function “main()” is in the filesSample.cpp.

```
cl filesSample.cpp lib.cpp sub1.cpp sub2.cpp
```

- the following will produce executable name “clientApp.exe”.

```
cl filesSample.cpp lib.cpp sub1.cpp sub2.cpp -o clientApp.exe
```

Sample files

```
// filesSample.cpp
#include <stdio.h>
#include <fstream>
#include <iomanip>
#include <iostream>
using namespace std;

#include "lib.h"
#include "memory.h"

int main()
{
	int arr[10];

	memset(arr, 0, sizeof(arr));
	memcpy(Test.a, "abcd", 4);
	Test.x = 10;
	Test.y = 20;

	for (int i = 0; i < sizeof(arr) / sizeof(arr[0]); i++)
		arr[i] = i * 2;

	for (int i = 0; i < Max1; i++)
		cout << i << ": This is a Demo! \n" ;

	cout << "Test Module returns: " << testModule() << endl;
	cout << "Sort dummy returns: " << getList(arr, 4) << endl;
	cout << "Old MyStructType value: a= \"" << Test.a <<
					"\" x= " << Test.x <<
					 " y= " << Test.y << endl;
	func2(&Test);
	cout << "Old MyStructType value: a= \"" << Test.a <<
					"\" x= " << Test.x <<
					 " y= " << Test.y << endl;
	return 0;
}
```

```
// lib.cpp

#include "stdio.h"
#include <fstream>
#include <iomanip>
#include <iostream>

#include "lib.h"

int Max1 = 4;
int Max2 = 4;
MyStructType Test;

MyStructType * func(MyStructType *a)
{
	return &a[0];
}

int finditem(int *a)
{
	return *(a + 5);
}

int getList(int *a, int ct)
{
	return *(a+ct);
```

```
// sub1.cpp
#include "stdio.h"
#include <fstream>
#include <iomanip>
#include <iostream>

#include "lib.h"

int testModule()
{
	return Max1 + 1;
}
```

```
// sub2.cpp
#include "stdio.h"
#include <fstream>
#include <iomanip>
#include <iostream>

#include "lib.h"

void func1()
{
	return;

}

void func2( MyStructType *t)
{
	t->x = Max2+ t->x;
	t->y = Max2 + t->y;
	memcpy(t->a, "new one", 8);
	return ;
}
```

```
//lib.h
#pragma once

typedef struct {
	int x;
	int y;
	char a[4];
} MyStructType;


extern int Max1;
extern int Max2;
extern MyStructType Test;

extern void func1();
extern void func2(MyStructType *);

extern MyStructType * func(MyStructType[]);
extern int getList(int *, int);
extern int testModule();
```

TO CREATE STATIC LIBRARY

A library is basically just an archive of object files.

This will show you how to create libraries files which contain only functions. This is good for program modularity, and code re-use. Write Once, Use Many.

You can turn these source files into libraries that can be used statically or dynamically by other programs.

Step 1: Setup your environment

same as above

Step 2 : Create your source file(s).

Sample:

Mylib.cpp in source folder c:\sr\cpp

```
#include <myLib.h>

int factorial(int n)
{
    if (n==2)
        return 2;
    return n * factorial(n-1);
}
```

Mylib.h in source folder c:\sr\cpp\h

```
#ifndef __MYLIBH__
#define __MYLIBH__

int factorial(int);

#endif
```

Step 3 : To build static library:

In *Windows:*

- cl /c /EHsc myLib.cpp /I ./h //note: /I is an upper case "i" . this will produce *.obj
- lib /out:myLib.lib myLib.obj

or

- lib /out:myLib.lib mylib.obj /I ./h

In *Linux;*

- gcc -c myLib.cpp // this will produce *.obj
- ar r myLib.a myLib.o // this will produce library file *.a. You can name <whatever>.a
- ranlib myLib.a // create indexing inside the library file

To use the static library:

Sample test file which uses the function from the library : clientApp.cpp

```
#include <myLib.h>
void main()
{
        return factorial(6);
}
```

To build the test file - clientApp.cpp

Note: you must specify "-I./" to tell the compiler where to find the header files. The alternative is to set your system include path environment variable:

In Windows: e.g. your header files are in c:\sr\cpp\h

- cl /W4 /EHsc /Ic:\sr\cpp\h clientApp.cpp myLib.lib /link /out:clientApp.exe

or

- set include=%include%;c:\sr\cpp\h
- cl /W4 /EHsc clientApp.cpp myLib.lib /link /out:clientApp.exe

In Linux: e.g. your header files are in /myNewPath

- gcc -o clientApp -c clientApp.cpp -I /myNewPath -L. -lmyLib clientApp.o

or

- export C_INCLUDE_PATH=$C_INCLUDE_PATH;/myNewPath
- gcc -o clientApp -c clientApp.cpp -L. -lmyLib clientApp.o

IMPORTANT:

- Make sure you are "appending", instead of overwrite the system environment variable.)
- -I, -L, etc., take precedence over environment variables
- C_INCLUDE_PATH may be different based on the version of compiler you use. Look up the proper system variable path.

TO CREATE SHARED LIBRARY

In Windows:

Sample myDll.h

```
#ifndef  __MYDLLH__
#define __MYDLLH__

#ifdef    MYDLL_EXPORTS
#define       MYDLL_API __declspec(dllexport)
#else
#define       MYDLL_API __declspec(dllimport)
#endif

MYUTIL_API   int   factorial(int);
#endif
```

e.g. your header files are in c:\sr\cpp\h, your source file is in c:\sr\cpp.

- set include=%include%;c:\sr\cpp\h
- cl /EHsc /DMYDLL_EXPORTS /LD myDll.cpp

NOTE:

- if you do not append the "./h" in your system include path, you need to add in "-I . /h".
- MYDLL_EXPORTS is a symbol name which you will need to use

To test with the client app:

- cl /W4 /EHsc dllClient.cpp /DMDLL_EXPORTS /out:dllClient.exe

In Linux:

1) A few system environment variables you should know:
   LD_LIBRARY_PATH
   C_INCLUDE_PATH

   e.g.

   ```
   export LD_LIBRARY_PATH=/home/pi/wk/libfolder1:/home/pi/wk/libfolder2
   export C_INCLUDE_PATH=/home/pi/wk/inc
   ```

2) You should create at least :
   - One library *.c or *.cpp file
   - One header file containing the applicable global, macros, function prototypes, etc.

3) Compile your library file and make it become a shared library.
   e.g. your library files are : fact.c and fib.c

```
gcc -c -Wall -Werror -fpic  fact.c    fib.c
gcc -shared -o libSimple.so    fact.o fib.o
```

note that : libSimple.so is the shared library. The extension must be *.so

4) Test your client app test.c:

```
gcc  -Wall -o test  test.c   -lSimple   -L$LD_LIBRARY_PATH
```

5) If you want to see all the libraries your app is linked to :

```
ldd  /home/pi/wk/test
```

Ref:

Setting the Path and Environment Variables for Command-Line Builds

Walkthrough: Compiling a C Program on the Command Line