Command Line Build Your Own C/C++ Files

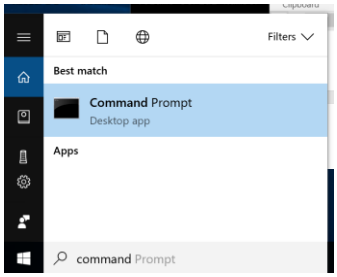
Contents

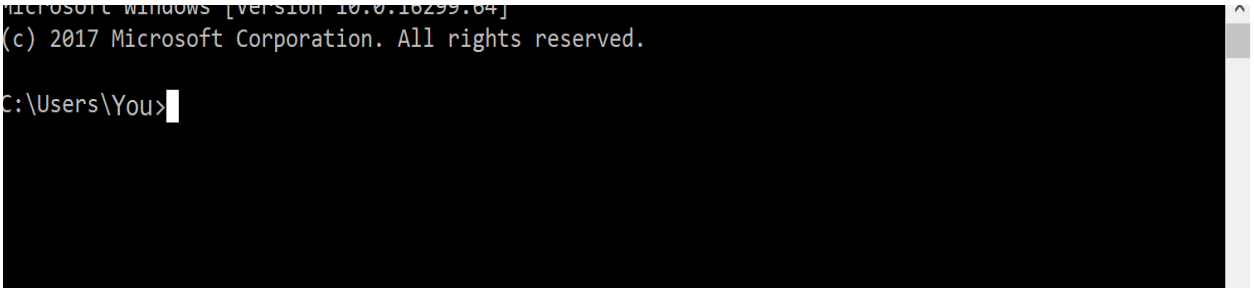
Learn rudimentary stuff about command prompt 3
Start the Command Prompt..... 3
Basic commands you should know: (not case sensitive) 3
Find location of your VC installation. 4
Change to the location 4
Set up the build environment..... 4
Compile & Link your own project 5
Sample files 6
To automate Your USACO project tests with Batch file 9
Sample batch file to automate your test with a typical USACO problem set: 9
to Create Shared Library (Windows) 10
Steps: 10
In the case of building Static Shared library 10
Create Header files..... 10
To build Static linked library: 11
To test with an application: 11
In the case of building dynamic link library: 11
Create Header files..... 11
Build the dynamic link library: 12
To test with an application: 12
A couple of common What if situations: 12
Need to make smaller runtime footprint..... 13
to Create Shared Library (Linux) 14

setup up environment..... 14
In the case of building Static Shared library 14
In the case of building Dynamic link library 14
Test your client app test.cpp: 15

LEARN RUDIMENTARY STUFF ABOUT COMMAND PROMPT

START THE COMMAND PROMPT





BASIC COMMANDS YOU SHOULD KNOW: (NOT CASE SENSITIVE)

- cd
  - cd c:\users*your userlogin here*
  - cd .. (go up one folder)
- dir
  - dir /od
  - dir /os

You should know some very basics about your window file system:

- C:\windows
- C:\program files
- C:\program files (x86)
- C:\users\...
- There is no such folder called “Quick Access” ; unless you manually create it, of course.

FIND LOCATION OF YOUR VC INSTALLATION.

Various version of visual studio can use different folder names.

I have 2 samples here. One is for 2015, another one is for 2017 community version

One for 2015 Version: "C:\Program Files (x86)\Microsoft Visual Studio 14.0\"

One for 2017 Community Version :
"C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Auxiliary\Build"

If you are not sure where it is installed. Go thru the File Explorer. It should be installed under either:

- C:\Program Files (x86) or
- C:\Program Files

CHANGE TO THE LOCATION

e.g. : (depending on which version, of course)

```
cd "C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Auxiliary\Build
```

or

```
cd "C:\Program Files (x86)\Microsoft Visual Studio 14.0\VC\bin
```

SET UP THE BUILD ENVIRONMENT.

At C:\Program Files (x86)\Microsoft Visual Studio\2017\Community\VC\Auxiliary\Build>
At the folder, run this:

```
vcvcvars64.bat
```

You should see a list *.bat batch command files. These are command script files.

To verify :
Type at the command line:

```
cl
```

Your should see:

```
Microsoft (R) C/C++ Optimizing Compiler Version 19.12.25835 for x64
Copyright (C) Microsoft Corporation.  All rights reserved.
```

usage: cl [option...] filename... [/link linkoption...]

NOTE: As a self-learner, you should view the content inside the bat file to learn about batch commands (windows script file). This will help you to understand how you may set your environment differently.

Reference: If you are interested, here is a link to all Compiler Options.

COMPILE & LINK YOUR OWN PROJECT

e.g. location of your project is at: \users\You\Project1

```
> cd c:\Users\You \Project1
```

To build 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);
```

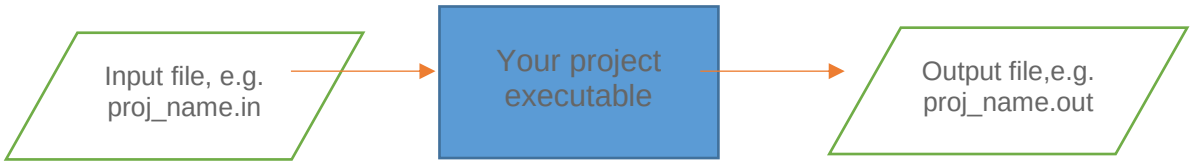
```
// sub1.cpp
#include "stdio.h"
#include <fstream>
#include <iomanip>
#include <iostream>

#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 AUTOMATE YOUR USACO PROJECT TESTS WITH BATCH FILE

USACO practice problem expects:



Eg. : The usaco's project name : ***censor***. Thus, your program assumes the input file name is "***censor.in***", and store your output to "***censor.out***.

Steps:

1) copy 1.in to censor.in (Usually there are 10+ test data files, e.g. 1.in, 2.in, …. 15.in)
2) run your code
3) file compare 1.out with censor.out.

Important reminder:

1) Should compile your code in Release mode.
2) When you submit it as a test, you should always pick C++. Your file should have extension .cpp

Sample batch file to automate your test with a typical USACO problem set:

```
@for /l %%a in (1 1 15)     do    ( call :runIt censor %%a  )
@goto :EOF

:runIt
@set fname=%1
@set ct=%2
@copy %ct%.in %fname%.in
@%fname%.exe %ct%.in
@fc %fname%.out %ct%.out

:EOF
```

TO CREATE SHARED LIBRARY (WINDOWS)

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.

STEPS:

1. Setup your environment; as shown above.
2. Create proper header and library files. Note that you will need:
3. Proper header file for user applications.
4. If you are building dynamic link library instead of static shared library, you will need to create a special header for your library C/C++ source file
    a. Do note this is not the same header file as the one used by the user applications.

IN THE CASE OF BUILDING STATIC SHARED LIBRARY

Create Header files

Create proper headers and function files.

Sample:

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

```
#ifndef __MYLIBH__
#define __MYLIBH__

int factorial(int);

#endif
```

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

```
#include <fact.h>

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

To build Static linked library:

Generate the obj file :

```
cl  /c  /EHsc  fact.cpp  /I./
```

Build the shared library *.lib :

```
lib  fact.obj
```

After this, a library file called fact.lib is generated.

To test with an application:

```
cl  testApp.cpp  fact.lib
```

Do note that testApp.exe can be run in a different folder from fact.lib.

IN THE CASE OF BUILDING DYNAMIC LINK LIBRARY:

Create Header files

Create proper headers and function files.

Sample:

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

```
#ifndef __MYDLL__
#define __MYDLL__

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

MYDLL_API  int  factorial(int);

                                #endif
```

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

```
#include <factdll.h>

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

```
}
```

Build the dynamic link library:

- cl /c /EHsc factdll.cpp /I./h /DMYDLL_EXPORTS
  //note: /I is an upper case "i" to indicate path for header files

This will create two files : factdll.lib and factdll.dll .

To test with an application:

- cl /EHsc testApp.cpp /link fact.lib

Run your app.

A COUPLE OF COMMON WHAT IF SITUATIONS:

1) Your Dll file is in another location other than your current folder. Then, the path to the dll files must be included in the system path.

   Eg. Your location of your newly built dll is at : C:\myfolders\mydlls\

   Your testApp.exe is elsewhere. You need to do:

   - set path=%path%;c:\myfolders\mydlls

2) location of the header files is not in the same folder where the testApp.cpp is.

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 testApp.cpp fact.lib

or

- set include=%include%;c:\sr\cpp\h
- cl /W4 /EHsc testApp.cpp fact.lib

3) Want to name the executable name different from the "main" source file.

- add /out: to generate executable file name not myApp.cpp. Eg.
  - cl myApp.cpp fact.lib /out:hello.exe

NEED TO MAKE SMALLER RUNTIME FOOTPRINT

Produces an output file to run on the Windows Runtime. Footprint is far smaller:

- dir myApp.exe ←---- this is just to check the original size.
- cl myApp.cpp factorial.lib /ZW /EHsc
- dir hello.exe ←---- check the new size.

This will generate even smaller footprint as it removes default window metadata:

- cl myApp.cpp factorial.lib -D_CRT_SECURE_NO_WARNINGS /ZW:nostdlib /EHsc

TO CREATE SHARED LIBRARY (LINUX)

SETUP UP ENVIRONMENT

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
```

IN THE CASE OF BUILDING STATIC SHARED LIBRARY

You should create at least :

- One library *.c or *.cpp file
- One header file containing the applicable global, macros, function prototypes, etc.

More sample:

```
ar r libSimple.a   Simple.o        // this will produce library file *.a. You can name <whatever>.a
ranlib   mySimple.a                 // create indexing inside the library file
```

Test your client app test.cpp:

```
gcc -o testExe  testApp.c   libSimple.a
```

or

```
gcc -o testExe  testApp.c   -L/path/to/library-directory   -lSimple
```

IN THE CASE OF BUILDING DYNAMIC LINK LIBRARY

1) Compile your library file and make it become a dynamically lined “shared object” library.
   e.g. your library files are : fact.cpp and fib.cpp

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

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

It MUST be prefixed with **lib**xxxx.so.

Test your client app test.cpp:

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

note that it is -lSimple NOT, -llibSimple .

If 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.

Ref:

Setting the Path and Environment Variables for Command-Line Builds

Walkthrough: Compiling a C Program on the Command Line