Welcome to Linux Programming using C/C++

The GCC compiler can be used to compile C/C++ programs under Linux. The compiler driver for C programs (.c) is GCC and the driver for C++ (.cpp) is g++. In this article , we use .cpp as the extension and will be using g++ in most of the cases

```
/////////////////
//first.cpp
//
// g++ first.cpp
// ./a.out
//
//
// g++ -o jsph.exe first.cpp
// ./jsph.exe
//

#include <stdio.h>


int main( int argc , char **argv )
{

        printf("hello world..\n");
}
```

Here are the ways which we can compile this programs

```
praseed@debian:~/joseph$ gedit first.cpp
praseed@debian:~/joseph$ g++ first.cpp
praseed@debian:~/joseph$ ./a.out
hello world..
praseed@debian:~/joseph$ gedit first.cpp &
[1] 3914
praseed@debian:~/joseph$ g++ -o jsph.exe first.cpp
praseed@debian:~/joseph$ ./jsph.exe
hello world..
praseed@debian:~/joseph$
```

Compiling Mutliple Files into a Single Executable

Modify first.cpp to call a function Add from another source module

Let us look into the source code of second.cpp

```
//////////////////
//
// g++ -c second.cpp
//     will produce second.o
//
// link these object file with first.o to produce
// first.exe
//
// g++ -c first.cpp
// g++ -c second.cpp
// g++ -o first.exe first.o second.o
// ./first.exe
//

#include <stdio.h>

extern "C" int Add( int a , int b ) {
          return a+b;
}
```

Let us look at the revised version for first.cpp

```
/////////////////
//first.cpp (version 2)
//
// g++ first.cpp second.cpp
// ./a.out
//
//
// g++ -o jsph.exe first.cpp second.cpp
// ./jsph.exe
//
// Another alternative
//
// g++ -c first.cpp
// g++ -c second.cpp
// g++ -o jsph.exe first.o second.o
//
//

#include <stdio.h>

extern "C" int Add( int , int );

int main( int argc , char **argv )
{

          printf("hello world..%d\n",Add(100,1));
```

```
}
```

Look at how we can compile and link programs

```
praseed@debian:~/joseph$ gedit second.cpp
praseed@debian:~/joseph$ g++ first.cpp second.cpp
praseed@debian:~/joseph$ ./a.out
hello world..101
praseed@debian:~/joseph$ g++ -c first.cpp
praseed@debian:~/joseph$ g++ -c second.cpp
praseed@debian:~/joseph$ g++ -o jsph.exe first.o second.o
praseed@debian:~/joseph$ ./jsph.exe
hello world..101
praseed@debian:~/joseph$
```

Creating static libraries under Linux (.a)

Let us create third.cpp which contains a routine to multiple two numbers

```
//////////////////
// third.cpp
//
// g++ -c third.cpp will produce third.o
//
//

#include <stdio.h>

extern "C" int Mul(int a , int b ) {
        return a*b;
}
```

Let us look at the revised version for second.cpp

```
//////////////////
// second.cpp (second version )
// g++ -c second.cpp
//     will produce second.o
//
//

#include <stdio.h>
```

```
extern "C" int Add( int a , int b ) {
	return a+b;
}
```

Let us take a look at the revised version of first.cpp

```
/////////////////
//first.cpp (version 3)
//
// g++ first.cpp second.cpp third.cpp
// ./a.out
//
//
// g++ -o jsph.exe first.cpp second.cpp third.cpp
// ./jsph.exe
//
// Another alternative
// --------------------
// g++ -c first.cpp
// g++ -c second.cpp
// g++ -c third.cpp
// g++ -o jsph.exe first.o second.o third.o
//
// Creating static libraries
// -------------------------
// g++ -c second.cpp
// g++ -c third.cpp
// ar cru libArith.a second.o third.o
// g++ -c first.cpp
// g++ -o jsph.exe first.o libArith.a
// ./jsph.exe
//
//
//

#include <stdio.h>

extern "C" int Add( int , int );
extern "C" int Mul(int , int );

int main( int argc , char **argv )
{

	printf("hello world..%d\t%d\n",Add(100,1),Mul(7,9));
}
```

Look at how we can compile and link programs

```
praseed@debian:~/joseph$ ar cru libArith.a second.o third.o
praseed@debian:~/joseph$ g++ -c first.cpp
praseed@debian:~/joseph$ g++ -o jsph.exe first.o libArith.a
praseed@debian:~/joseph$ ./jsph.exe
hello world..101        63
praseed@debian:~/joseph$
```

Creating a Shared Object under Linux (.so)

In this case , we can reuse the first.cpp , second.cpp to demonstrate. Rather than giving a verbose explanation , I am planning to spit the command window.

```
praseed@debian:~/joseph$ clear

praseed@debian:~/joseph$ g++ -c second.cpp
praseed@debian:~/joseph$ g++ -c third.cpp
praseed@debian:~/joseph$ g++ -shared -fPIC -o libArith.so second.o third.o
praseed@debian:~/joseph$ g++ -o jsph.exe first.cpp libArith.so
praseed@debian:~/joseph$ ./jsph.exe
./jsph.exe: error while loading shared libraries: libArith.so: cannot open shared object file: No such
file or directory
praseed@debian:~/joseph$ g++ -o jsph.exe first.cpp ./libArith.so
praseed@debian:~/joseph$ ./jsph.exe
hello world..101        63
praseed@debian:~/joseph$ echo $LD_LIBRARY_PATH

praseed@debian:~/joseph$ set LD_LIBRARY_PATH=.
praseed@debian:~/joseph$ export LD_LIBRARY_PATH
praseed@debian:~/joseph$ g++ -o jsph.exe first.cpp libArith.so
praseed@debian:~/joseph$ ./jsph.exe
./jsph.exe: error while loading shared libraries: libArith.so: cannot open shared object file: No such
file or directory

praseed@debian:~/joseph$ LD_LIBRARY_PATH=.
praseed@debian:~/joseph$ export LD_LIBRARY_PATH
praseed@debian:~/joseph$ g++ -o jsph.exe first.cpp libArith.so
praseed@debian:~/joseph$ ./jsph.exe
hello world..101        63
praseed@debian:~/joseph$ echo $LD_LIBRARY_PATH
.
praseed@debian:~/joseph$
```

The environment variable LD_LIBRARY_PATH can be set to avoid giving the irritating ./ character set before the libraries.

Dynamically loading a .so from one's program.

This program assumes that LD_LIBRARY_PATH is set to . This time we write a new source file called dyncaller.cpp to dynamically load a dll and execute a function via function pointer.

```
//////////////////////
//dyncaller.cpp
//
// g++ dyncaller.cpp -ldl
// ./a.out
//
//

#include <stdio.h>
#include <dlfcn.h>

typedef int (*BIN_FUNCTION)(int , int );


int main( int argc , char **argv )
{

  void *handle = dlopen("libArith.so",RTLD_LAZY);

  if ( handle == 0 )
  {
    printf("Failed to load the program ...\n");
    return 0;
  }

  BIN_FUNCTION bn = (BIN_FUNCTION)dlsym(handle,"Add");

  if ( bn == 0 )
  {
    printf("Failed to retrieve the function ....\n");
    return 0;
  }

  int nc = (*bn)(10,10);
  printf("Value is %d\n",nc);
  dlclose(handle);
}
```

Here is the console dump

```
praseed@debian:~/joseph$ g++ dyncaller.cpp
```

```
/tmp/ccqE8Beb.o: In function `main':
dyncaller.cpp:(.text+0x21): undefined reference to `dlopen'
dyncaller.cpp:(.text+0x52): undefined reference to `dlsym'
dyncaller.cpp:(.text+0xa5): undefined reference to `dlclose'
collect2: ld returned 1 exit status
praseed@debian:~/joseph$ g++ dyncaller.cpp -ldl
praseed@debian:~/joseph$ ./a.out
Value is 20
praseed@debian:~/joseph$
```

Finally , as a bonus let us see how we can invoke the shared object from Mono

```
//////////////////////////////////////////
// test.cs
//
// gmcs test.cs
// mono test.exe
//
//
using System;
using System.Runtime.InteropServices;


public class Test
{
  [DllImport("./libArith.so")]
  extern static  int Add(int a, int b );

  public static void Main( String [] args ) {
    System.Console.WriteLine("Value is {0} ...",Add(11,12));
  }


}
```