Welcome to Cross Platform C/C++ Programming

By

Praseed Pai (praseedp@yahoo.com)

(This article was published by Reboot Magazine from CUSAT , Kerala)

Most college students feel that C/C++ programming language is dying a slow death and there is no point in learning it beyond the academic “compulsions”. To complicate the matter , the venerable Turbo C/C++ compiler is still the most prominent tool used to teach C/C++ programming in our part of the world. The problem is that Turbo C/C++ is a 16 bit compiler and world has moved on to 32 bit and 64 bit programming long ago. To learn the nuances of modern C/C++ programming one should use a modern compiler like Visual C/C++(for Windows) and GCC (for MAC OS X and LINUX). To be a successful C/C++ programmer , he or she should be a platform agnostic person and should be able to write portable code which can be recompiled on various platforms. In this article , i am planning to cover some of the techniques which is in vogue to write cross platform code. The purpose of this article is to teach a useful subset of C/C++ programming language as well.

Since GCC is used on GNU Linux and MAC OS X , we can treat them as a POSIX based system. There are subtle differences between Windows and POSIX platform. I have written a .h file , which will help us to write the application code in a portable manner.

```
////////////////////////
// Example.h
// A header file which will help the application
// programmer to write code for multiple platform
//
//
#ifndef _EXAMPLE_DOT_H
#define _EXAMPLE_DOT_H
#include <stdio.h>
#include <stdlib.h>

#ifdef _WINDOWS   // if we are compiling for Windows
#include <windows.h>
////////////////////////////////////////////////////
//Comparison function for qsort under Windows should use
//__cdecl calling convention ( The parameters are
//pushed from Right to Left and callee pops the stack )
//
#define CALL_CONV_FOR_QSORT  __cdecl
///////////////////////////
//Functions to be exported should using
//__stdcall function ( The parameters are pushed from
//Right to Leftand callee pops the stack )
//
#define STDCALL_CONV __stdcall
//////////////////////////////
// Directive for Exporting the function from a DLL
#define EXPORTED_FUNC __declspec(dllexport)

/////////////////////////////////
// Windows Function for Loading a DLL
//      LoadLibrary
// Windows Function for retrieved a function address
//      GetProcAddress
// Windows Function for Unloading the DLL
//      FreeLibrary
//

#define LOAD_DLL(hmodule,str) \
 ( ( hmodule = LoadLibrary((str)) ) != INVALID_HANDLE_VALUE )
#define GET_FUNC_PTR(hmodule,str) \
 (GetProcAddress(hmodule,(str)))
```

```
#define UNLOAD_DLL(hmodule) FreeLibrary(hmodule);

#else

#include <dlfcn.h>   // for GNU Linux and MAC OS X
#define CALL_CONV_FOR_QSORT
#define EXPORTED_FUNC
#define STDCALL_CONV

/////////////////////////////////////
// UNIX Function for Loading a DLL
//       dlopen
// UNIX Function for retrieved a function address
//       dlsym
// UNIX Function for Unloading the DLL
//       dlclose
//
#define LOAD_DLL(hmodule,str) \
( ( hmodule = dlopen((str),RTLD_LAZY) ) != 0 )

#define GET_FUNC_PTR(hmodule,str) dlsym(hmodule,(str));
#define UNLOAD_DLL(hmodule) dlclose(hmodule);

#endif

/////////////////////////////////////////////////////
// Function Pointer for our exported function
typedef int ( STDCALL_CONV *QSORT_FUNC)(double *,int);

#endif
```

The default calling convention for Windows is __stdcall. In this calling convention , the parameters are pushed from right to left and the callee pops the stack. In the case of __cdecl , the parameters are pushed from right to left and caller pops the stack . This has to be kept in mind while writing code for Windows. That is why under Windows , we have defined a maco STDCALL_CONV as __stdcall. In the case of Microsoft C run time library , qsort function expects a function which follows __cdecl calling conventions. The macro CALL_CONV_FOR_QSORT is defined as __cdecl under Windows.

Given below is the source code of the QSORT library and the code is being compiled on to a DLL or Shared Library (.so) depending upon the platform.

```
/////////////////////////////////////////////////////////
// Qsort.cpp
// Written by Praseed Pai K.T.
//           http://praseedp.blogspot.com
//
// under Windows @ the Visual C/C++ command prompt
// ------------------------------------------------
// cl /D_WINDOWS /c Qsort.cpp
// link /DLL /out:QSORT.dll /DEF:QSort.def Qsort.obj
//
// under GNU Linux @ the Terminal
// ------------------------------
// g++ -c -o -fPIC Qsort.o Qsort.cpp
// g++ -shared -o libQSORT.so Qsort.o
//
// under MAC OS X @ the Terminal
// ------------------------------
// g++ -c -o -fPIC Qsort.o Qsort.cpp
// g++ -dynamiclib -o libQSORT.so Qsort.o
//
#include "EXAMPLE.h"
/////////////////////////////////////////////////
// Comparison function for qsort ANSI C/C++
// library function. Visual C/C++ mandates that
// it should be __cdecl as the default calling
// convention is __stdcall
// static qualifier is to restrict the visibility
// of CmpFn outside this source file.

static int CALL_CONV_FOR_QSORT CmpFn(const void *dblone,const void *dbltwo ){
        double a = *(double *)dblone;
        double b = *(double *)dbltwo;
        return ( a > b ) ? 1 : ( b > a ) ? -1 :0;
}

////////////////////////////////////////////////////
// QSORT - Sort a double array
// It just makes a call to qsort library function
//

extern "C" EXPORTED_FUNC void STDCALL_CONV QSORT( double *arr , int nelem ) {
        qsort(arr,nelem,sizeof(double),CmpFn);
}
```

The following module defenition file (QSORT.def) file has to be given at the linker command line on Windows.

```
LIBRARY QSORT



EXPORTS

        QSORT
```

The DLL or .SO can be linked statically or we can load them while the program is running. I have shown the later technique as it is more difficult to get it right.

The Windows platform uses LoadLibrary to load a DLL , GetProcAddress for retrieving the address of exported function and FreeLibrary to free the library. The corresponding POSIX calls are dlopen , dlsym and dlclose. The macros LOAD_DLL , GET_FUNC_PTR and UNLOAD_DLL is defined to reflect this depending upon the platform.

Another thing which a programmer should be familiar is function pointers. To execute a function exported from the DLL or .SO at the run time , we need to be familiar with function pointers. Conceptually , a function pointer is a variable which stores the address of a function. The syntax of declaring the function pointers is the thing which confuses beginning students.

Suppose there is function by the name Add which is declared as

```
extern "C" int Add( int a , int b ) {

        return a+b;

}
```

The function pointer for this function can be declared as

`int ( *a ) ( int , int )   = (int ( *) (int , int ) )Add;`

now the variable a contains the address of Add function , we can invoke the function

as (*a)(3,4) . Even a(3,4) would do.

While declaring a function pointer , we need to be careful about the calling convention as well. If the Add function was following __stdcall calling convention , we should have declared the stuff as

`int (__stdcall *a) ( int , int )   = (int (__stdcall *) (int , int ) )Add;`

```
////////////////////////////////////
// caller.cpp
//
// This program calls QSORT.dll or libQSORT.so
// depending on the platform
//
// Written by Praseed Pai K.T.
//          http://praseedp.blogspot.com
//
//
//  under Windows @ the Visual C/C++ command prompt
// ---------------------------------------------
// cl /D_WINDOWS caller.cpp
//
// under GNU Linux and MAC OS X  @ the Terminal
// -------------------------------------------
// g++ -o caller.exe caller.cpp -ldl
// ./caller.exe 10 -4 6 9
//
//

#include <stdio.h>
#include "EXAMPLE.h"

/////////////////////////////////////
//
// Main is the user entry point ...
// Think about this ..Who calls main ?
//
//

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

        if ( argc == 1 ){
                fprintf(stdout,"No command line arguments\n");
                return -1;
        }

        int num = argc-1;
        double *arr = (double *) malloc(num*sizeof(double));

        if ( arr == 0 )
                return -1;

        double *parr = arr;
        char **nums = &argv[1];

        while (num--)
                *parr++ = atof(*nums++);
        QSORT_FUNC sorter;
        #ifdef _WINDOWS
                HMODULE hmodule;
                const char *dllname = "QSORT.dll";
        #else
                void *hmodule;
                const char *dllname = "./libQSORT.so";
        #endif
        if (!LOAD_DLL(hmodule,dllname))
        {
                fprintf(stdout,"Failed to load %s\n",dllname);
                return -1;
        }
        sorter = (QSORT_FUNC) GET_FUNC_PTR(hmodule,"QSORT");
        if ( sorter == 0 ) {

                fprintf(stdout,"failed to retrieve function pointer\n");
                return -2;
        }

        (*sorter)(arr,argc-1);
        num = 0;
        while (num < argc-1 )
                printf("%g\n",arr[num++]);

        free(arr);

        UNLOAD_DLL(hmodule);

        return 0;

}


}
```

In the main program , every C/C++ program will have at least one parameter (argc will always be at least one) and that will be the executable file name. We are assuming that rest of the arguments will be numbers to be sorted. We use malloc to allocate the amount of memory depending upon the number of parameters.

The argv array elements are of type char * and we use atof (ascii to float) to convert the argument into IEEE 754 floating point number. The retrieved numbers are passed to QSORT function to get it sorted and the results are dumped on to the console. The QSORT function relies on the qsort library function from the stdlib.h header.

Finally , qsort is a ANSI C/C++ library function which can sort data by supplying a comparitor function (cmpFn in the Qsort.cpp is the comparitor) . If you have followed me this far , welecome to the wonderful world of C/C++ programming.

Praseed Pai is a renowned software professional from Kochi,India. He is a regular and highly rated speaker at the Microsoft User Groups , BarCamps , OWASP events and the Linux User groups. He is the founder of slangfordotnet (http://slangfordotnet.codeplex.com) which is widely used by students and academia as a base for their curriculum projects. Praseed's personal blog can be accessed from http://praseedp.blogspot.com