Using C++ in a C program

FlyLikeABeagle

December 7, 2022

1 Introduction

Often a programmer, who is using C, might need a library or feature that is only offered in C++. A common example is a lambda expression, STL or Opencv. In this guide we will provide steps to use C++ code in a C program.

2 External Linkage

In the following sections we will be attempting to implement the simple Hello World example. That is, have a C function call a C++ function that prints to the console Hello World. Consider the following C++ header file `hello.h`

```
#ifndef HELLO_WORLD_H_
#define HELLO_WORLD_H_

extern "C" void printMsgToScreen(void);

#endif
```

Note the keyword `extern "C"`. This tells the compiler to use external linkage specific to the C language; it will use C calling conventions and name mangling [1]. The source file `hello.cpp` should look like this

```
#include "hello.h"
#include <iostream>

using namespace std;

extern "C" void printMsgToScreen(void)
{
  cout << "Hello World from a C++ program !" << endl;

}
```

Notice the standard C++ library used is `iostream` and we are printing to the console using `std::cout` and not `printf()`.

3 Making a Shared Library

Include the following `Makefile` in your source directory.

```
SOURCES = hello.cpp
TARGETS = hello

PUBDIR = # Your public directory
OUTDIR = $(PUBDIR)

# Cross compile or not?
CROSS_COMP = arm-linux-gnueabihf-
CC_CPP = g++

# Set flags
CPP_FLAGS = -c -fPIC
```

```
13
14 all:
15   $(CC_CPP) $(CPP_FLAGS) -o hello.o $(SOURCES)
16   $(CC_CPP) -shared -o libhello.so hello.o
```

Line 15 builds the source file `hello.cpp` into an object file `hello.o`. Line 16 creates a shared library file `libhello.so` using the object file `hello.o`. This is much like using the library `libpthread.so` and linking it using `-lpthread`.

4 The C Program

Now that we have the C++ source file `hello.cpp` made as a shared `.so` file we can directly use the `printMsgToScreen` function using the following C code

```
1  #include <stdlib.h>
2  #include<stdio.h>
3
4  extern void printMsgToScreen(void); // declare the external fucntion for usage.
5
6  int main(int argc, char** argv)
7  {
8    // call the function
9    printMsgToScreen();
10   return 0;
11 }
```

Then use the following `Makefile` in your C source directory.

```
1  SOURCES = main.c
2  TARGET = main
3
4  PUBDIR = # Your public directory
5
6  OUTDIR = $(PUBDIR)
7
8  # Cross compile?
9  CROSS_TOOL = arm-linux-gnueabihf-
10 CC_C = $(CROSS_TOOL)gcc
11
12 CFLAGS =-Wall -std=c99
13
14 all:
15   $(CC_C) $(SOURCES) -L ./hello -Wall -o $(OUTDIR)/$(TARGET)  -lhello
16   export LD_LIBRARY_PATH=$(PUBDIR)/hello/:$LD_LIBRARY_PATH
```

Note that this `Makefile` assumes that the `hello.o` and `libhello.so` files reside with your C source files. The `-L ./hello` command tells the compiler to look for the `hello.o` object file and we link the `.so` file using `-lhello`. After running `make` in the terminal you should be able to successfully run `./main`.

5 References

[1] ttps://isocpp.org/wiki/faq/mixing-c-and-cpp