Hello, World! in C

Johann 'Myrkraverk' Oskarsson

October 23, 2018

Contents

1 The Quintessential Example Program

This is Hello, World! An example program for testing the compiler and run time system of the C programming language. It is often the first program people try out. It's main purpose is to teach people how to use the compiler and linker (often the same program) of their choice.

This is in order to get people familiar with the tools of the trade. Many newcomers don't differentiate between their text editor, compiler and the programming language when they start with an *integrated development environment* or IDE.

The *programming language* is the text written on screen, or paper for that matter. And this is what the compiler will read in order to create the executable program. The environment used to create this text is immaterial to the programming language and can be switched at will. Here, the focus is on what each part of the program does, why it is the way it is, and what every part means.

Part I
Printing Text

The first thing we cover here, is how C prints text to standard output. *Standard output* is an operating system concept, and refers to the usual text output device of programs. Because people are used to graphical user interfaces, there is often no visible place for this. On UNIX, OS X and OpenVMS, this habitually refers to the terminal being used to run the application. On Windows and OS/2[1], this is the command prompt, typically run with `CMD`.

The C function for printing out text is called *printf*( ). The *f* stands for *format* and the first argument is a format string — something we don't make use of here — we just print out the format string as–is.

At the end of the format string we have `"\n"` which tells the compiler to include a literal newline into the string. This makes our text, when output by the resulting program, advance the cursor to the next line.

2 ⟨ print hello, world 2 ⟩ ≡

```
printf("Hello,␣world!\n");
```

This code is cited in chunk 3.

This code is used in chunk 3.

[1]Available as ArcaOS at the time of writing

Part II
The Main Function

Every C program has a *main*( ) function which is the starting point of the program. What that means is when the program starts, it executes the *main*( ) function and stops when it reaches the end if. The return value is then returned to the operating system. What exactly that means depends on the operating system. In UNIX and OS X terminals this is the `$?` shell variable, and in Windows and OS/2, this is the `%ERRORLEVEL%`.

First we include the header files we need; more on that in the next section. Then we start the main function. The standard way of defining it is as follows, it starts with the **int** return type, then comes the *main* which is the name of the function, then the parameters follow, the first one is of type **int** and habitually called *argc*. The __unused is a non-standord way to declare to the compiler we're not using that variable. The second variable is an array of pointers to **char** or an array of strings as C programmers may think of it. The $*$ denotes a pointer, and the [ ] denotes an array. This parameter is habitually called *argv*.

We use __unused for both arguments because modern compilers will warn about unused arguments. For the *main*( ) function, the arguments are part of the runtime environment and we really should include them as–is even though we don't use them. There are other and more portable tricks to achieve the same thing, but we don't cover them here.

The code that the function executes is included between the open brace, {, and close brace, }. The first thing we do is to ⟨ print hello, world 2 ⟩ and then we end the function with **return** 0; which tells the operating system we have exited successfully.[2]

```
3   ⟨ include header files 4 ⟩
      int main(int __unused argc, char __unused *argv[])
      {
        ⟨ print hello, world 2 ⟩
        return 0;
      }
```

[2]It's actually system dependent what successfully means here, but most operating systems today use the UNIX convention of zero meaning success.

Part III
The Header Files

There is only one header file we need for this program, it's named `stdio.h` and includes the declaration[3] of the *printf* ( ) function.

On OS X, this header file will also include a macro definition of *__unused* so we don't need to do anything about it.

4 ⟨include header files 4⟩ ≡
```
#include <stdio.h>
```
This code is used in chunk 3.

[3]We refer to any introductory text on the C programming language for the precise meaning of *declaration*

Part IV
Compiling and Running the Program

Clang and GCC. These compilers work on UNIX and compatible systems, including OS X where these examples are taken from.

Both compilers support the `-Wall` and `-Wextra` command line flags, so we use them. On OS X, Clang masquerades as GCC so we only include one command line example here.

```
gcc -Wall -Wextra -o hello hello.c
```

If your header files do not define the *__unused* macro, you can put

```
-D__unused=__attribute__((unused))
```

on the command line.

We can now run the executable as follows.

```
./hello
```

OpenWatcom. This compiler works on DOS, Linux, OS/2, and Windows. The example below was tested on OS/2; we expect the other system to behave very similarly.

The OpenWatcom compiler doesn't understand *__unused* so we define it to be nothing on the command line, and invoke the compile and link utility as such.

```
wcl386 -d__unused= hello.c
```

Which gives us a `hello.exe` file we can run as such.

```
hello
```

Visual Studio. In order to compile with Visual Studio on the command line, it's best to open up a special command prompt such as the *x64 Native Tools Command Prompt for VS 2017.* This can be done from the Windows menu or the search feature.

The Visual Studio compiler doesn't understand *__unused* either, so we define it to the empty string; and invoke the compiler as such.

```
cl /D__unused= hello.c
```

Which gives us a `hello.exe` file we can run as such.

```
hello
```

OpenVMS. VSI C does not understand *__unused* either, so we define it to nothing and invoke the compiler as such.

```
cc /define="__unused=" hello
```

And then we link the program with,

```
link hello
```

and finally to run the program we do:

```
run hello
```

Output. What we should now have on our screen is what follows.

```
Hello, world!
```

Index

List of Refinements

⟨ include header files 4 ⟩ Used in chunk 3.
⟨ print hello, world 2 ⟩ Cited in chunk 3. Used in chunk 3.