Contents

1 Hello World

This section will detail how to print "Hello World!" to standard output in C.

1.1 Writing Hello World

First, open a new C file. Name it whatever you want. I will name mine `hello.c`. You should start your file with this line:

```
#include <stdio.h>
```

Any line that starts with a `#` is read by the **C Preprocessor**. This is a program that does simple text edits to our files before the rest of the compiler reads them.

`#include` is a **preprocessor directive**. By inserting `<stdio.h>` after it, we are telling the preprocessor to include the contents of the file `stdio.h` in our file. This is a header file (a .h file), a type of file that is generally included at the top of our source files (.c files). We will explore the topic of header files further later. For now, all you need to know is that `stdio.h` contains code that lets our compiler know how to use various functions from the **C standard library**.

A **library** is a collection of code we can use to accomplish certain tasks. The **C standard library**, also called **libc**, is the standardized library for the C language. It is meant to be portable, meaning you can run code that uses it on multiple different operating systems. We will use it to print "Hello World!"

Let's get back to adding to our file:

```
#include <stdio.h>

int main(void)
```

The new line we've added, `int main(void)`, is a **function header**. A function header describes the name, return type, and parameters for a **function**.

In our case, `int` is the return type for our function named `main`, and `void` signifies that our function `main` takes no parameters.

When using libc, like we are, the `main` function is the entry point of our program. The code in `main` will be the first thing that is executed when our program runs.

Right now, our function has no code. Let's change that:

```
#include <stdio.h>

int main(void)
{
        printf("Hello World!\n");
        return 0;
}
```

When a function is called (another term for a function being executed or run), the **code block**, placed after the function header, is run. This code block is known as the **function definition**. A code block starts with an opening curly bracket `{` and ends with a closing curly bracket `}`. In our code block, we can add **code statements**, which are statements that specify an action for our program to execute. The code statements of a code block are run in order from top to bottom (and left to right if you put multiple statements on one line).

In our code block, the first code statement, `printf("Hello World!\n")` calls the function `printf` from libc and passes `"Hello World!\n"` as input. Our line ends with `;` to denote the end of the code statement.

`printf` is a function that sends text to **standard output**. The place that standard output is linked to depends on how we run our program. By default, standard output is linked to our screen, so when we call `printf`, it will output text to our screen.

Our next code statement, `return 0`, returns the value `0`, which becomes the return value of our function. The return value of the `main` function is known as the **exit code** of a program, which is used by the program that calls our program to tell if our program ran successfully or not. An exit code of `0` indicates that our program ran successfully, and a nonzero exit code indicates a failure.

Finally, we are done writing hello world. If you've made it this far, congratulations!

1.2 Compiling and Running Hello World on UNIX

Before our program can be run, we need to **compile** it, which will convert it from C to machine code which our computer's CPU can run.

To compile our program, run this command in the shell:

```
cc hello.c
```

`cc` is an alias for any C compiler available on your system, if there is one. It will likely evaluate to `gcc`, the GNU Compiler Collection, which is a widely used C compiler. If you want to explicitly use `gcc`, replace `cc` with it. In most cases, `cc` and `gcc` can be used interchangeably, unless you want to use a special feature of GCC that doesn't exist in other compilers.

The above command will produce the output file `a.out`. This is an **executable** or **binary**, which is a program file you can run. To run it, run this command in the shell:

```
./a.out
```

`.` indicates our working directory, so `./a.out` indicates to the shell to run the file `a.out` located in the working directory. Just running `a.out` likely won't do anything because, without `./`, the shell won't assume that `a.out` is in the working directory.

If you want the compiler to give the output file a name, use the `-o` flag:

```
cc hello.c -o hello
```

This will make the compiler name our executable `hello`.

1.3 Further Dissecting Hello World

When we initially wrote hello world, I glossed over some details to not make the program seem too intimidating. If you want to dive deeper into these details, read on.

1.3.1 String literals and escape sequences

The input we give to `printf`, `"Hello World!\n"`, is a **string literal**. A **literal** is a raw value written in the code, and a **string** is a type of variable that contains text. A string literal starts and ends with a double quote character `"`.

The characters `\n` are an example of an **escape sequence**, which is a specific sequence of characters that get translated by the compiler into a single special character that generally can't be typed on a keyboard. An escape sequence consists of a backslash \ and one or more following characters.

In our case, `\n` is an escape sequence that gets translated to the **newline character** or simply a **newline**. This character is used to mark the end of a line of text.

Here is a table of some basic escape sequences:

`\n`	newline
`\t`	tab
`\\`	a literal backslash
`\'`	a single quote
`\"`	a double quote
`\?`	a question mark
`\0`	null terminator

1.3.2 Function parameters and arguments

Functions take in zero or more input values when you call them. The definitions of these input values are called **parameters**. The actual values that are passed as input to a function when the function is called are known as **arguments**. So, in our case, `"Hello World!\n"` was an argument we passed to `printf`.