An Introductory Guide to C

Sam Moore

January 2020

1 Introduction

Having built a number of guides to Python, a relatively common request is for students to learn C. In fairness, this is completely understandable. C is a highly versatile language, and is extremely popular because of this. I have also personally seen C used for a number of applications in astrophysics, from high-level cosmological simulations to Particle-in-Cell codes modelling plasmas. It is for this reason that I have decided to create this introductory guide to C. I also confess that I am a firm believer that the best way to fully understand something is to teach it, so here goes nothing. As a quick aside, I assume no prior knowledge of Python or coding, but I will make references to Python where it helps.

Let's talk a bit more about C before we delve into the actual coding. C is a relatively old programming language developed in 1972 by Dennis Ritchie as a method of making things on UNIX. The language was later used for the kernel on UNIX, and has gained so much popularity as a language that it is commonly learnt in all sorts of fields.

Let us begin with the most basic program you can write: Hello World!

2 Hello World!

```
#include <stdio.h>
int main() {
  printf{"Hello World!"}
  return 0;
}
```

Let us dissect this a bit and discuss some important features of C. First of all, note the line:

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

You should include this at the beginning of all of your coding projects in C. The include statement tells the compiler to call the stdio.h file, in which includes the prototypes of functions such as printf(). Without this line, your compiler wouldn't understand the printf() line.

The next part of the code is the main() function. The main() function is special because it is always the first block of code that is read by the compiler. Since there are no other functions, it is in fact the only function that is read.

Within this function, printf() is used to return a string/integer/etc to stdout. In this case, it returns the string ”Hello World!” as an output.

The final part of this code that may come as a surprise to experienced programmers in languages such as Python and Java is the final line:

```
return 0;
```

In C, the main() function is of type int, so it has to return an integer. Typically, 0 is a standard convention for noting that the code was successful in its running. This is also a convention in C++, but whereas in C++ where 0 can be returned implicitly, in C you must explicitly return an integer.

The final thing that I shall note at this point (especially if you are used to Python) is the importance of semi-colons. They are required to end statements.

The next step is working out how exactly to run your program. For UNIX systems, save the file with the extension .c (e.g: helloworld.c). Compile the program in terminal with:

```
cc helloworld.c
```

Now run the file by typing

```
a.out
```

Let us continue by looking at escape sequences. These are ways of introducing special features into character strings. Here are some of the main ones:

Escape Sequences	
Code	Feature
\\n	newline
\\t	tab
\\b	backspace
\\”	”

To test this, modify your helloworld.c file to

```
printf(”Hello world!\n”)
```

and see what happens.

1.1. Experiment with the features above.

To conclude this introduction, a quick note on documentation. It is imperative in my opinion to establish good coding practice early on. Part of this is well-documented code. Anything written between \∗ and ∗\ is ignored by the complier.

1.2. Go back and write a comment of what helloworld.c does before int main().

3 printf()

Let us look a bit more at printf(). Suppose we have some lengthy variable and we want to print it. We can write a printf() statement as follows:

```
int main() {
    float var1;
    var1 = 3.14159;
    printf("%3f", var1);
    return 0;
    }
```

Here, the % specifies how to print var1. The format is as follows:

$$\% + (\text{num. of characters wide}) + .(\text{num, of dp}) + (\text{type of number}) \quad (1)$$

For example, $\%3.2f$ prints a float that is 3 characters wide with 2 decimal points. For the types, here is a reference list.

printf() Types	
Symbol	Type
d	decimal integer
f	float
x	hexadecimal
o	octal
c	character
s	string

4 Data Types

Let us begin by considering identifiers in C. An identifier is a name given to something, such as a variable or a function. In C, you are required to declare the datatype of an identifier. For example:

```
int temperature;
double boltzmann;
```

Here, the words 'temperature' and 'boltzmann' are identifiers (in this case, they are the names of variables). 'int' and 'double' are keywords - a reserved word in C for a specific function. The keyword 'int' declares the variable 'temperature' to be an integer, whereas the keyword 'double' declares the variable 'boltzmann' to be a double. You always have to declare the datatype of an identifier.

For completeness, here is a table of all of the main datatypes in C.

Data Types		
Data Type	Bytes	Format Identifier
char	1	%c
int	4 (usually)	%d
short int	2 (usually)	%hd
long int	8 (usually)	%li
long long int	$\geq$8	%lli
double	8	%lf
long double	12, 16	%Lf

Let us go into some more detail regarding the above. The specifics are perhaps a bit much at this stage, but this should all be in the back of your mind.

4.1 char

char is the keyword used for declaring characters, as follows:

```
char letter_a = 'a';
```

4.2 long/short

4.3 signed

To conclude:

sizeof()

5 Loops

5.1 For loops

Here is an example of how a for loop is implemented in C:

```
int num;
for (num = 1; num <= 10; ++num)
    printf(num)
```

Thus there are three stages to a for loop: the initalisation of the variable, a control condition, and then the increment stage.

6 Functions

7 Arrays

Let us consider how to go about using arrays in C. Typically, most guides to C introduce arrays much later, but as they are highly useful for much of the code we shall look that, I am giving an introduction to them here. To begin though, we must first consider that notion of pointers. A pointer is a variable that contains the address of a variable, i.e: where in the memory of a computer the variable is stored. This is considered to be one of the more challenging aspects of C, particularly when trying to work out what someone else's code is doing. Let us look at this slowly.

8 Sorting Algorithms: C vs Python

Now that we have a stronger understanding of C, let us look at some algorithms designed for sorting. In addition, for those interested, I will compare the code in C to the code in Python.

9 Newton-Raphson Method:

Let us look at some numerical computing codes that use C.

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

// Newton-Raphson Method

double f, df;

// define function
double dB (double x) {
        return (x-5)*exp(x) + 5;
}

// define derivative
double ddB(double x) {
        return (x-4)*exp(x);
}

int main (void) {
        double xold = 4.1;                // initial guess for NR
        int itrtnmax = 1000;              // max. number of iterations
        double eps = 1*pow(10, -10);     // tolerance for convergence

        // Begin NR:

        int notdone = 1;
        int itrtn = 0;
        double xnew, error, root;

        while (notdone == 1 && itrtn < itrtnmax) {
                f = dB(xold);
                df = ddB(xold);
                if (df != 0) {
                        xnew = xold - f / df;
                        error = fabs(xnew - xold);
                        if (error > eps*xnew) {
                                xold = xnew;
                                itrtn = itrtn + 1;
                        } else {
                                root = xnew;
                                notdone = 0;
                        printf("At iteration %2d, the root is %.13f with itera
                        }
                } else {
                        printf("Derivative is zero\n");
                        break;
                        }
        }
        return 0;
}
```