Constants

In this lesson, you will get to know about constants and their different types in C.

We'll cover the following

Definition

> Constants are values that do not change after they have been defined.

Numeric Constants

An example of an `int` constant is the number `1234`. An example of a floating-point constant (by default typed as a `double`) is `123.4` and `1e-2`. We can write numbers in octal or hexadecimal instead of decimal: octal by using a leading zero (0) and hexadecimal by using a leading zero-x (0x). Decimal 31 can be written as 037 in octal and 0x1f or 0X1F in hexadecimal. Here are some examples of defining numeric constants:

```
int year = 1984;       // integer constant 1984
int octalYear = 03700; // 1984 in octal
int hexYear = 0x7c0;   // 1984 in hexadecimal
```

Here is some code to show how to print integers in various representations. Execute the following code.

```
#include <stdio.h>

int main() {
  printf("1984 in decimal is %d\n", 1984);
  printf("1984 in octal is 0%o\n", 1984);
  printf("1984 in hexadecimal is 0x%x\n", 1984);
  printf("0123 is octal for %d\n", 0123);
  printf("0x12f is hexadecimal for %d\n", 0x12f);
  return 0;
}
```

Character Constants

A character constant is written between **single quotes**, for example, '`x`'. Characters in C are represented using **integer** values, from the **ASCII character set**. ASCII codes range between 0 and 255. The upper-case alphabet starts at 65 (`A`) and ends at 90 (`Z`); the lowercase alphabet starts at 97 (`a`) and ends at 122 (`z`). Other symbols such as `(`, `!`, Tab, carriage return, etc., are also represented in the ASCII table. See ASCII (Wikipedia) and AsciiTable for the mapping between characters and integer ASCII codes.

An important character constant to know about is the constant '`\0`' which represents the character with value zero, sometimes called the `NULL` character. We will see later when we talk about string handling in C that '`\0`' is used to terminate variable-length strings.

String Constants

String constants can be specified using a sequence of zero or more characters enclosed within **double quotes**, e.g., "`C is fun`". A string constant is technically an array of characters that is terminated by a null character '`\0`' at the end. This means that the storage required to represent a string of length `n` is actually `n+1`. Thus we can store strings of arbitrary length in memory as long as they are terminated by a null character (so we know when they stop). We will talk about arrays later.

Enumeration Constants

An enumeration constant is a list of constant integer values that you can assign to arbitrary labels. They provide a convenient way to associate constant values with names. For example, you could store the months of the year like this:

```
enum months { JAN=1, FEB, MAR, APR, MAY, JUN, JUL, AUG, SEP, OCT, NOV, DEC };
```

Now you have defined a new enumerated constant data type called `months`. Now a variable of type `months` can only take on values as defined above. You can use the symbolic names (e.g. `JAN`) in place of their integer counterparts, for example like this:

```
months the_month;
...
if (the_month == JAN) {
  printf("it's January\n");
}
```

Why not just use strings to represent months? One reason is that in C strings are slightly clunky to work with, especially compared to interpreted languages like Python, R, etc. Comparing two strings in C is not as easy as typing `if (the_month == "JAN")` ... it requires a call to a function in `string.h` called `strcmp()`.

Another reason is that because `enum` data types are represented as integers, you can do integer operations (comparisons, arithmetic, etc.) on them... so, for example, you could do something clever like:

```
if ((the_month > APR) && (the_month < SEP)) {
  printf("it's summer!\");
}
```

That's it for variables and constants. We will now learn how to declare them in C.