Memory Model – Memory Allocation in C/C++

Published by **roy** on July 18, 2019

Memory Allocation in programming languages always refers to the process by which a program requests the memory manager to create space for the program to store data. This is a series article about Memory Allocation in C/C++. We will be discussing about the following topics in different distinct articles.

1. Memory Model [Part – 1]
2. Memory Structure of C/C++ Programs [Part – 2]
3. Dynamic Memory Allocation [Part – 3]
4. Pointers: Behind the scene [Part – 4]
5. The Stack [Part – 5]
6. Scope of Variables [Part – 6]

In this article we are going to learn about Memory Model.

Memory Model [Part – 1]

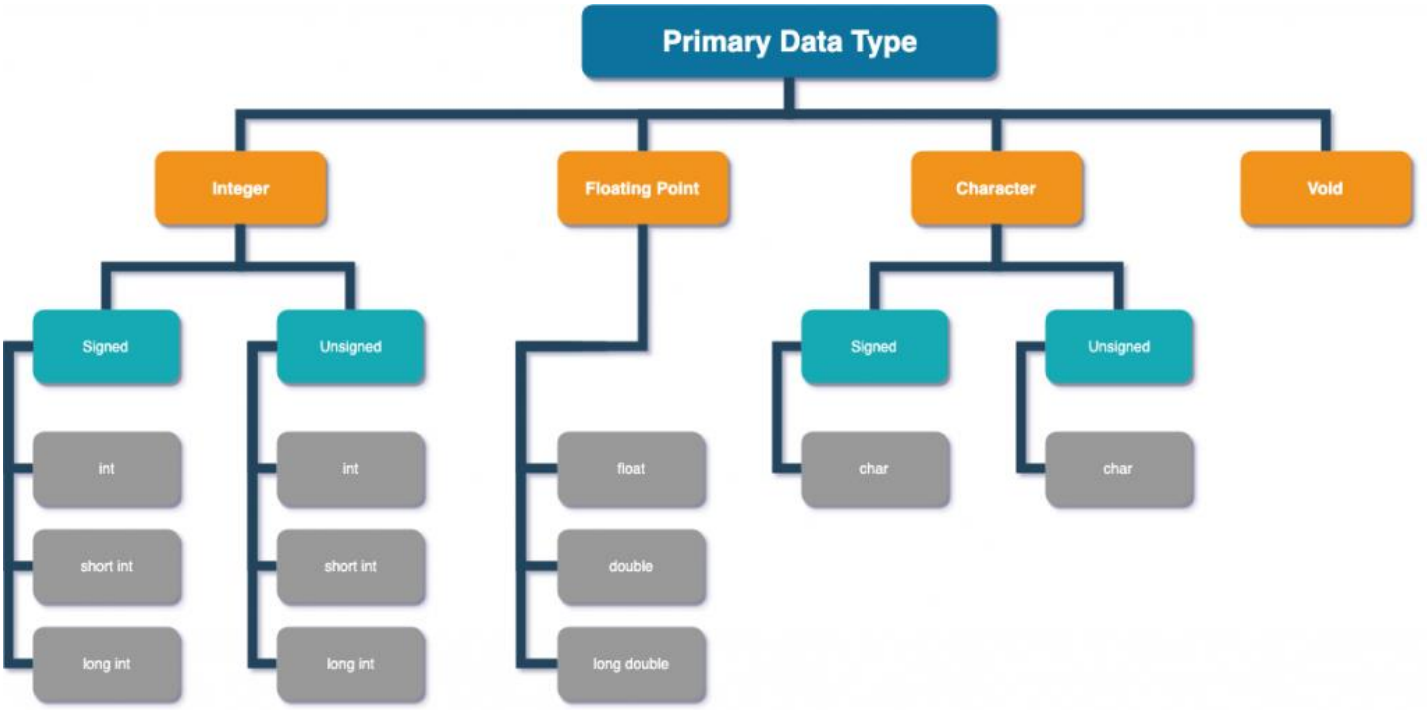
Before we start to explore memory model in C/C++, let's talk about data types. Data types specifies what types of data a variable will be holding and what types of data we should store in that particular variable.

C/C++ has some predefined data types, can be divided into two groups:

1. Primary data types
2. Derived data types

Primary data types: primary data types includes Integer(int), Floating Point Number(float, double), void and character

The tree looks like the following picture:



Primary Data Types in C/C++

for the primary data types in C/C++ we have looked so far, followings are their memory requirements

`int`	4 bytes (32 bits)
`short int`	2 bytes (16 bits)
`long int`	8 bytes (64 bits)
`char`	1 byte (8 bits)
`float`	4 bytes(32 bits)
`double`	8 bytes (64 bits)
`void * ( /pointers )`	8 bytes on (64 bits machine)

Derived data types: array, structure, pointers and union

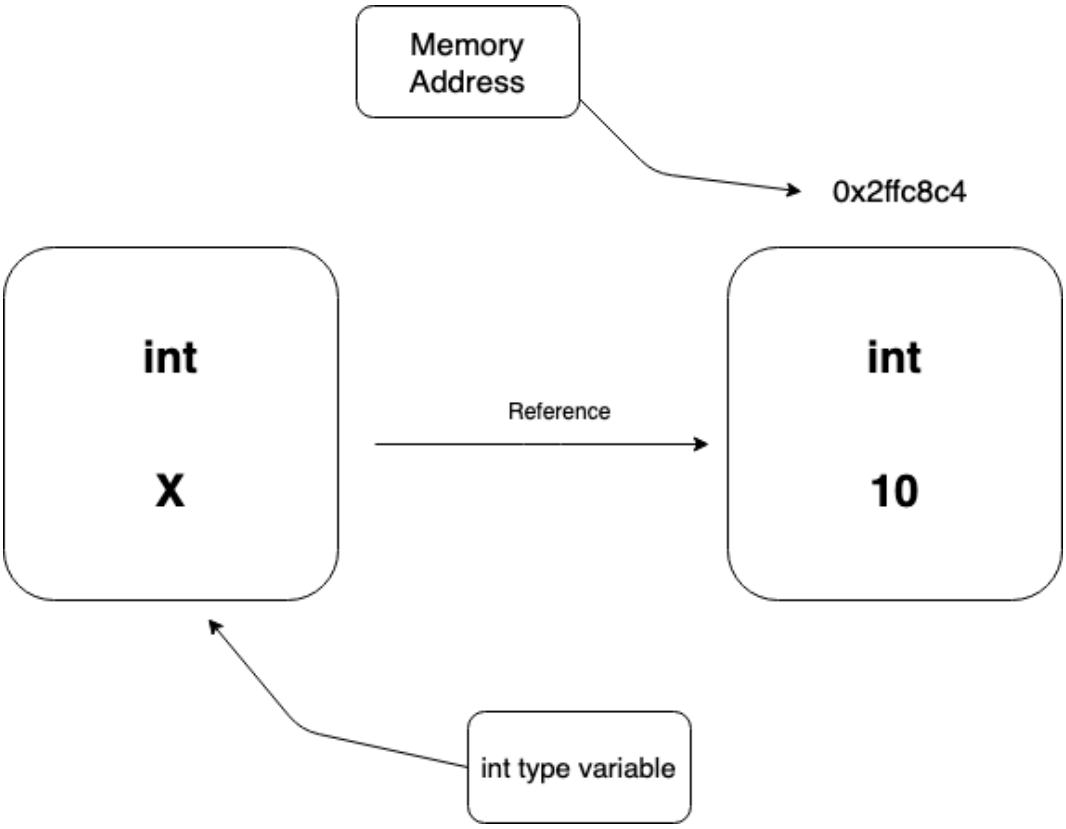
What does it means a particular data type requires 4/8 bytes of memory and where does the memory come from? To become a good programmer, needs to be conscious about the memory, where does it comes from and how its managed by the operating system. Because without proper understanding of memory model in C/C++ can leads to an unintended situation when you have to work on complex projects where heavy memory optimizations are required to increase the performance and write effective programs.

So, What happens when you declare a variable in C/C++? Lets see, how we can declare variables in C/C++ and what happens when we declare a variable.

```c
// Created by Komol Nath Roy on 2019-07-10.

#include <stdio.h>
int main()
{
    int x = 10;
    return 0;
}
```

In this piece of code there, we have stated `int x = 10`. Now, there are few things happening here, here `x` is variable name, `int` is data type, which we are associating with the variable `x` and lastly we have a value, an integer value `10` which we are assigning to `x`.



Now, when we are assigning value `10` to `x`, a state is being created in the memory (random memory address, i.e. `0x2ffc8c4`) where data type is `integer` and value is `10` and variable `x` is being created where data type is `int` and a connection between them is created.

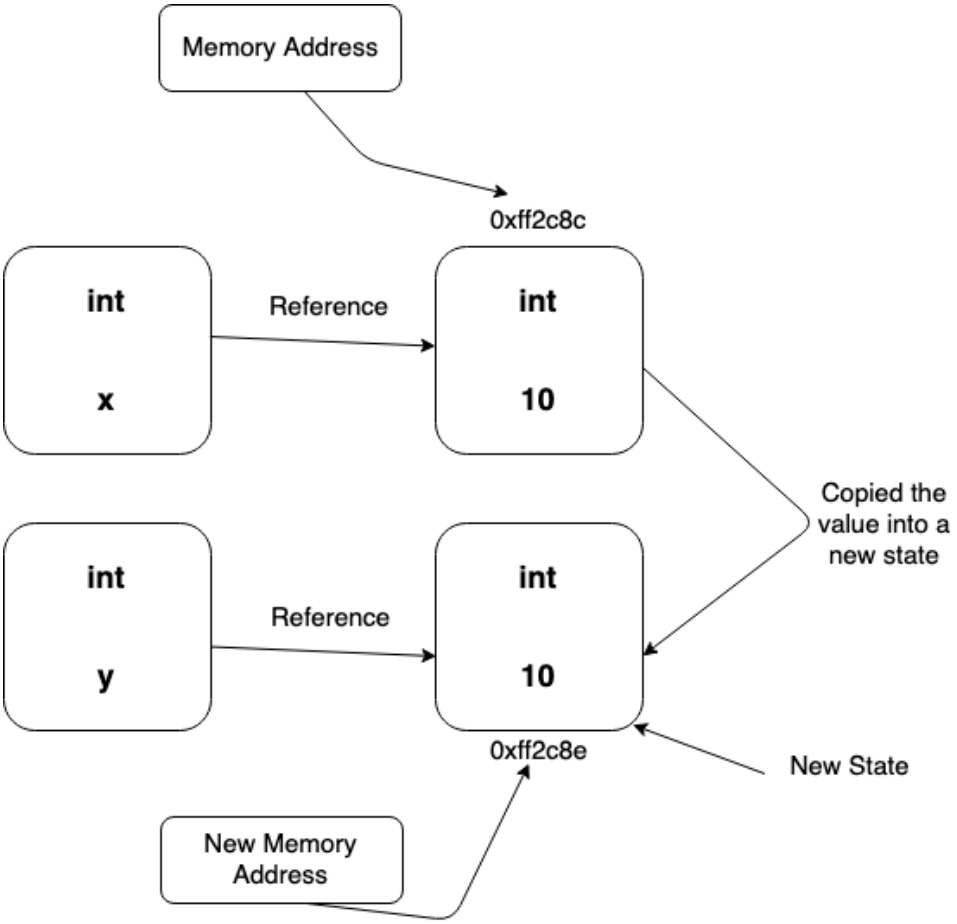
So, when we try to find out, what's inside `x`, we end up finding `10`, because `x` is associated with the memory address `0x2ffc8c4` which contains the value `10`.

Now, let's take a moment and take a look onto the following piece of code written in C below.

```c
// Created by Komol Nath Roy on 2019-07-10.
//
#include <stdio.h>
int main()
{
    int x = 10;
```

```
    int y = x;
    x = 5;
    printf("%d %d\n", x, y);
    return 0;
}
```

In this piece of code, we are assigning `10` into `x` and `x` into `y`. therefore the memory allocation for this program can be explained with the following image



Assigning variable into another variable

Here, after assigning `y` into `x`, then when we stated `x = 5`, this doesn't change the value of `y`, because when we assign one variable into another, C is just copying the value from one state and create a new state with a new memory address. this is why, variable `y` has a new memory address associated with it. so `x = 5` only updated the state of `x`.

Stay Tuned for the next part.

Categories: