Programming Fundamentals

Muhammad Ateeq

[Updated: 25-02-2023]

Variables, Data Types, Operators

Variable Representation

What is the difference between variables in mathematics and computer programming?
Variables in mathematics and computer programming have some similarities, but there are also some significant differences.

One key difference between variables in mathematics and computer programming is that in mathematics, variables are often treated as constants once they have been assigned a value, whereas in computer programming, variables can be changed or reassigned throughout the program. Additionally, in computer programming, variables must be declared and given a data type before they can be used, whereas in mathematics, variables are typically understood to represent any real number or quantity.

How are variables represented in computer memory (RAM)?
In computer memory, variables are typically represented as a sequence of bits, or binary digits. The size of the variable in memory depends on the data type of the variable, which determines the number of bits used to store the variable's value.

For example, a variable of the data type "int" (integer) is typically represented using 32 bits in memory, which can store values ranging from -2,147,483,648 to 2,147,483,647. A variable of the data type "float" (floating-point number) is typically represented using 32 bits in memory as well, but the bits are arranged in a way that allows the variable to represent decimal values with greater precision.

When a variable is declared in a computer program, a specific amount of memory is allocated to store the variable's value. The memory location where the variable is stored is identified by an address, which can be accessed by the program to read or modify the variable's value.

For example, if a variable "x" of data type "int" is declared and assigned the value 20, the variable's value will be stored in a specific location in memory, identified by its memory address. The program can then access this memory location to read the value of x or modify its value as needed as shown in Figure 1.

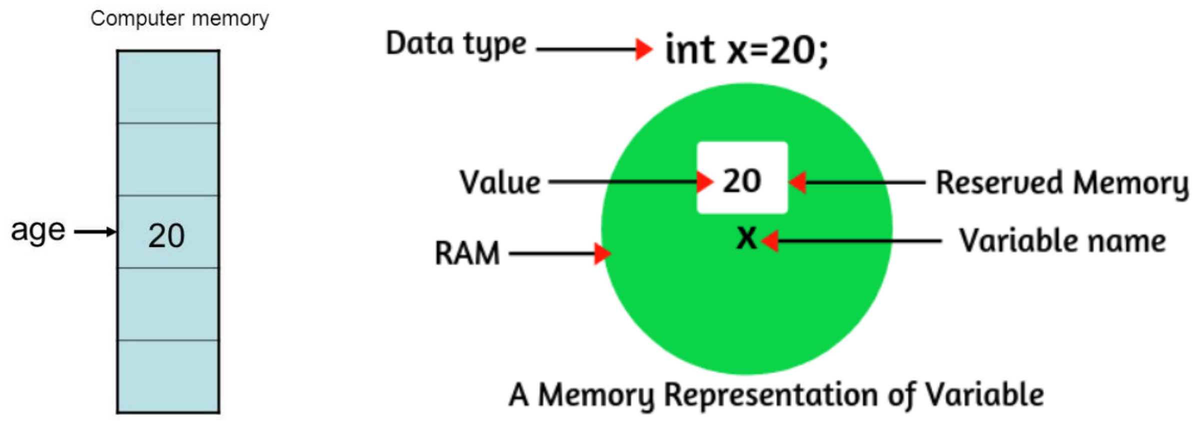


Figure 1. Variable in memory (RAM)

Data Types

What is a data type? Can you introduce to us some common data types in C++?
In computer programming, a data type is a classification of data that determines the type of operations that can be performed on it. In C++, there are several built-in data types that are used to represent different kinds of data. Here are some common data types in C++ and their descriptions:

Integer: An integer data type represents whole numbers (positive, negative, or zero) without a fractional component. In C++, there are several integer data types, including short, int, and long. The int data type is the most commonly used and is usually 32 bits on most platforms.
Float: A floating-point data type represents numbers with a fractional component. In C++, the float and double data types are used to represent floating-point numbers. float is a single-precision floating-point number and usually 32 bits in size, while double is a double-precision floating-point number and usually 64 bits in size.
Boolean: A boolean data type represents a binary value that can be either true or false. In C++, the bool data type is used to represent boolean values. The bool data type takes up 1 byte of memory.
String: A string data type represents a sequence of characters. In C++, the string class is used to represent strings. A string can be created by enclosing a sequence of characters in double quotes, like this:

```
string str = "Hello, world!";
```

In addition to these basic data types, C++ also supports several other data types, such as characters, arrays, and pointers, among others. Different data types are used to represent different kinds of data and provide the necessary operations to work with them.

Can you explain operations that can be performed on variables in C++?
In C++, variables are used to store and manipulate data. Variables can be used in various operations, such as arithmetic, logical, and relational operations. Here are some common operations that can be performed on variables in C++:

Arithmetic operations: Arithmetic operations are used to perform basic mathematical calculations on variables. The basic arithmetic operators in C++ are:

+ (addition)
- (subtraction)
* (multiplication)
/ (division)
% (modulus)

Here is an example that uses arithmetic operators on two variables:

```
int x = 5;
int y = 3;
int z = x + y;     // z contains the sum of x and y (8)
int a = x * y;     // a contains the product of x and y (15)
int b = x % y;     // b contains the remainder of x divided by y (2)
```

Logical operations: Logical operations are used to evaluate the truth or falsity of a condition. The basic logical operators in C++ are:

&& (logical AND)

|| (logical OR)
! (logical NOT)

Here is an example that uses logical operators on two variables:

```
bool a = true;
bool b = false;
bool c = a && b;   // c contains the result of a AND b (false)
bool d = a || b;   // d contains the result of a OR b (true)
bool e = !a;       // e contains the negation of a (false)
```

Relational operations: Relational operations are used to compare the values of two variables. The basic relational operators in C++ are:

== (equality)
!= (inequality)
< (less than)
> (greater than)
<= (less than or equal to)
>= (greater than or equal to)

Here is an example that uses relational operators on two variables:

```
int x = 5;
int y = 3    ;
bool a = x == y;   // a contains the result of x equals y (false)
bool b = x != y;   // b contains the result of x not equals y (true)
bool c = x < y;    // c contains the result of x less than y (false)
bool d = x > y;    // d contains the result of x greater than y (true)
bool e = x <= y;   // e contains the result of x less than or equals y (false)
bool f = x >= y;   // f contains the result of x greater than or equals y (true)
```

These are just some of the basic operations that can be performed on variables in C++. C++ also supports many other operations, such as bitwise operations, assignment operations, and type conversions, among others. The specific operations that can be performed on variables depend on the data type of the variables and the context in which they are used.

Let's take a pause here. We have seen something like "`int argc`" and "`char *argv[]`". What are these?
These are called command line arguments. Command line arguments are values provided to a program when it is run, which can be accessed by the program through the command line interface. They allow the user to pass input or options to a program at runtime (when the program is executed not when it is compiled).

In C++, command line arguments are stored in the `argv` array (argument vector) and the number of arguments is stored in the `argc` variable (argument count). The first element of the `argv` array, `argv[0]`, contains the name of the program itself.

Here's an example of how command line arguments can be used in a C++ program:

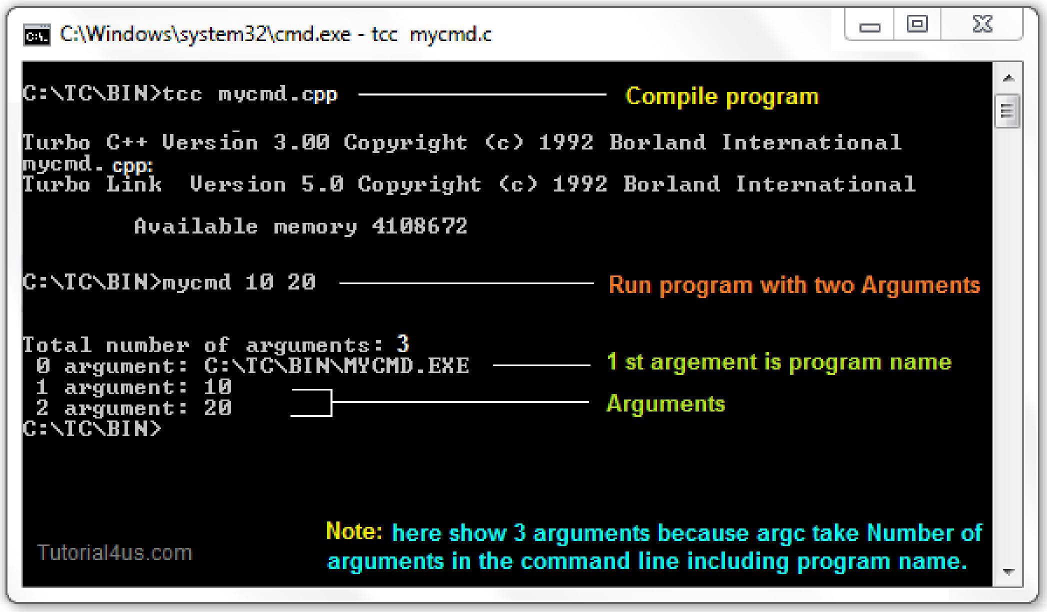
```
#include <iostream>

int main(int argc, char *argv[]) {
    std::cout << "Number of arguments: " << argc << std::endl;
    std::cout << "Program name: " << argv[0] << std::endl;
    std::cout << "First argument: " << argv[1] << std::endl;

    return 0;
}
```

In this program, argc stores the number of command line arguments passed to the program, and argv is an array of strings containing those arguments. The program prints out the number of arguments, the name of the program, and the first argument (if there is one).

For example, if the program is called myprogram.exe and is run with the command myprogram.exe argument1, the output would be:

```
Number of arguments: 2
Program name: ./myprogram
First argument: argument1
```

An illustrative example is also shown in figure 2 below.



```
C:\TC\BIN>tcc mycmd.cpp

Turbo C++ Version 3.00 Copyright (c) 1992 Borland International
mycmd.cpp:
Turbo Link  Version 5.0 Copyright (c) 1992 Borland International

        Available memory 4108672

C:\TC\BIN>mycmd 10 20

Total number of arguments: 3
 0 argument: C:\TC\BIN\MYCMD.EXE
 1 argument: 10
 2 argument: 20
C:\TC\BIN>
```

C:\Windows\system32\cmd.exe - tcc mycmd.c
Compile program
Run program with two Arguments
1 st argement is program name
Arguments
Note: here show 3 arguments because argc take Number of arguments in the command line including program name.
Tutorial4us.com

Figure 2. Command line arguments on console

Ok, we understand that argc is a variable of type int and has count of command line arguments stored in it. We kind of understand that char *argv[] stores the values of command line arguments (separated by space). But what exactly is char and why are there * and [] before and after argv (which probably is the name of variable)?

In C++, char is a built-in data type that represents a single character, typically encoded using the ASCII character set. It can hold any of the printable or non-printable characters, including letters, digits, symbols, and control characters.

The char data type is commonly used to store single characters, as well as to represent strings of characters as null-terminated arrays of char values.

Here's an example of how to declare and use char variables in C++:

```
#include <iostream>
Using namespace std;
int main() {
    char letter = 'A';
    cout << "The letter is: " << letter << endl;
    return 0;
}
```

In this example, a char variable named letter is declared and initialized with the character 'A'. The program then prints the value of the letter variable to the console.

It's worth noting that in C++, char is actually an integer type, and can be treated as such. This means that you can perform arithmetic operations on char values, such as incrementing or decrementing them, or using them in comparisons with other integer values. Why? This is because characters store ASCII values behind all characters.

```
char letter = 'A';
letter++;
cout << "The next letter is: " << letter << endl;
```

In this example, the letter variable is incremented using the ++ operator, which causes its value to be updated to 'B'. The program then prints the updated value of the letter variable to the console.

So we have `string` data type enclosed in double quotes and `char` data type enclosed in single quotes. Apart from the fact that `char` can store a single character at a time and `string` can store a sequence of characters, what is the difference between the two and why do we need two separate types?

In summary, `char` and `string` are separate data types in C++ because they represent different kinds of data (individual characters versus sequences of characters) and have different characteristics (fixed size versus dynamic size, low-level versus high-level). Using the appropriate data type for the task at hand can make code more efficient, easier to write and read, and less error-prone.