C++ Variables

C++ Variables

Variables are containers for storing data values.

In C++, there are different **types** of variables (defined with different keywords), for example:

- int - stores integers (whole numbers), without decimals, such as 123 or -123
- float**:** Single-precision floating point value.
- double - stores floating point numbers, with decimals, such as 19.99 or -19.99
- char - stores single characters, such as 'a' or 'B'. Char values are surrounded by single quotes
- string - stores text, such as "Hello World". String values are surrounded by double quotes
- bool - stores values with two states: true or false

Declaring (Creating) Variables

To create a variable, you must specify the type and assign it a value:

Syntax

type variable = *value*;

Where *type* is one of C++ types (such as int), and *variable* is the name of the variable (such as **x** or **myName**). The **equal sign** is used to assign values to the variable.

To create a variable that should store a number, look at the following example:

Example

Create a variable called **myNum** of type int and assign it the value **15**:

```
int myNum = 15;
cout << myNum;
```

You can also declare a variable without assigning the value, and assign the value later:

Example

```
int myNum;
myNum = 15;
cout << myNum;
```

Note that if you assign a new value to an existing variable, it will overwrite the previous value:

Example

```
int myNum = 15;  // myNum is 15
myNum = 10;  // Now myNum is 10
cout << myNum;  // Outputs 10
```

Example

```
// my first string
#include <iostream>

using namespace std;

int main ()
{
  string mystring;
  mystring = "This is a string";
  cout << mystring;
  return 0;
}
```

```
This is a string
```

Other Types

A demonstration of other data types:

Example

```
int myNum = 5;               // Integer (whole number without decimals)
double myFloatNum = 5.99;    // Floating point number (with decimals)
char myLetter = 'D';         // Character
string myText = "Hello";     // String (text)
bool myBoolean = true;       // Boolean (true or false)
```

Display Variables

The cout object is used together with the << operator to display variables.

To combine both text and a variable, separate them with the << operator:

Example

```
int myAge = 35;
cout << "I am " << myAge << " years old.";
```

Add Variables Together

To add a variable to another variable, you can use the + operator:

Example

```
int x = 5;
int y = 6;
int sum = x + y;
cout << sum;
```

C++ Declare Multiple Variables

Declare Many Variables

To declare more than one variable of the **same type**, use a comma-separated list:

Example

```
int x = 5, y = 6, z = 50;
cout << x + y + z;
```

C++ Identifiers

All C++ **variables** must be **identified** with **unique names**.

These unique names are called **identifiers**.

Identifiers can be short names (like x and y) or more descriptive names (age, sum, totalVolume).

Note: It is recommended to use descriptive names in order to create understandable and maintainable code:

Example

```
// Good
int minutesPerHour = 60;

// OK, but not so easy to understand what m actually is
int m = 60;
```

The general rules for constructing names for variables (unique identifiers) are:

- Names can contain letters, digits and underscores
- Names must begin with a letter or an underscore (_)
- Names are case sensitive (myVar and myvar are different variables)
- Names cannot contain whitespaces or special characters like !, #, %, etc.
- Reserved words (like C++ keywords, such as int) cannot be used as names

---

Types of variables based on their scope

1. Global variable
2. Local variable

Global Variable

A variable declared outside of any function (including main as well) is called global variable. Global variables have their scope throughout the program,

they can be accessed anywhere in the program, in the main, in the user defined function, anywhere.

Lets take an example to understand it:

Global variable example

Here we have a global variable `myVar`, that is declared outside of main. We have accessed the variable twice in the main() function without any issues.

```
#include <iostream>
using namespace std;
// This is a global variable
char myVar = 'A';
int main()
{
   cout <<"Value of myVar: "<< myVar<<endl;
   myVar='Z';
   cout <<"Value of myVar: "<< myVar;
   return 0;
}
```

Output:

```
Value of myVar: A
Value of myVar: Z
```

Example 2

```
// Definition and use of variables

#include <iostream>

using namespace std;

   int a = 5;      // Global variable

   int b = 2;      // Global variable

   float c = 2.50;          // Global variable

int main()
```

```
{
   char ch('A');  // local variable
   int sum;
   float add;
   cout << a << endl << b << endl;
   cout << ch << endl;
   sum = a + b;
   add = a + c;
   cout << sum << endl;
   cout << add << endl;
   return 0;
}
```

Local variable

Local variables are declared inside the braces of any user defined function, main function, loops or any control statements(if, if-else etc) and have their scope limited inside those braces.

Local variable example

```
#include <iostream>
using namespace std;

char myFuncn() {
// This is a local variable
char myVar = 'A';
}
int main()
{
  cout <<"Value of myVar: "<< myVar<<endl;
  myVar='Z';
  cout <<"Value of myVar: "<< myVar;
```

```
  return 0;
}
```

Output:

Compile time error, because we are trying to access the variable `myVar` outside of its scope. The scope of `myVar` is limited to the body of function `myFuncn()`, inside those braces.

C++ Constants

Constants

When you do not want others (or yourself) to override existing variable values, use the `const` keyword (this will declare the variable as "constant", which means **unchangeable and read-only**):

Example

```
const int myNum = 15;  // myNum will always be 15
myNum = 10;  // error: assignment of read-only variable 'myNum'
```

You should always declare the variable as constant when you have values that are unlikely to change:

Example

```
const int minutesPerHour = 60;
const float PI = 3.14;
```

C++ User Input

You have already learned that cout is used to output (print) values. Now we will use cin to get user input.

cin is a predefined variable that reads data from the keyboard with the extraction operator (>>).

In the following example, the user can input a number, which is stored in the variable x. Then we print the value of x:

Example

```
#include <iostream>

using namespace std;

int main()

{

    int a, b;

    cout << "Enter the first number" << endl;

    cin >> a;

    cout << "Enter the second number" << endl;

    cin >> b;

    cout << a << endl;

    cout << b << endl;

    return 0;

}
```

Example

```
#include <iostream>

using namespace std;

int main()

{

    int a, b;

    cout << "Enter the first number" << endl;

    cin >> a;

    cout << "Enter the second number" << endl;

    cin >> b;

    cout << a*b << endl;

    cout << b/2 << endl;

    return 0;

}
```

Example

```
int x;
cout << "Type a number: "; // Type a number and press enter
cin >> x; // Get user input from the keyboard
cout << "Your number is: " << x; // Display the input value
```

cout is pronounced "see-out". Used for **output**, and uses the insertion operator (<<)

cin is pronounced "see-in". Used for **input**, and uses the extraction operator (>>)

Creating a Simple Calculator

In this example, the user must input two numbers. Then we print the sum by calculating (adding) the two numbers:

Example

```
int x, y;
int sum;
cout << "Type a number: ";
cin >> x;
cout << "Type another number: ";
cin >> y;
sum = x + y;
cout << "Sum is: " << sum;
```

C++ Data Types

As explained in the Variables, a variable in C++ must be a specified data type:

Example

```
int myNum = 5;               // Integer (whole number)
float myFloatNum = 5.99;     // Floating point number
double myDoubleNum = 9.98;   // Floating point number
char myLetter = 'D';         // Character
bool myBoolean = true;       // Boolean
string myText = "Hello";     // String
```

Basic Data Types

The data type specifies the size and type of information the variable will store:

Data Type	Size	Description
int	4 bytes	Stores whole numbers, without decimals
float	4 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 7 decimal digits
double	8 bytes	Stores fractional numbers, containing one or more decimals. Sufficient for storing 15 decimal digits
boolean	1 byte	Stores true or false values
char	1 byte	Stores a single character/letter/number, or ASCII values

The sizeof Operator

The amount of memory needed to store an object of a certain type can be ascertained using the sizeof operator: sizeof(name) yields the size of an object in bytes, and the parameter name indicates the object type or the object itself. For example, sizeof(int)represents a value of 2 or 4 depending on the machine. In contrast, sizeof(float) will always equal 4.

```
#include <iostream>

using namespace std;

int main()

{

cout << "Size of char : " << sizeof(char) << endl;

cout << "Size of bool : " << sizeof(bool) << endl;

cout << "Size of int : " << sizeof(int) << endl;

cout << "Size of short int : " << sizeof(short int) << endl;
```

```
cout << "Size of long int : " << sizeof(long int) << endl;

cout << "Size of float : " << sizeof(float) << endl;

cout << "Size of double : " << sizeof(double) << endl;

cout << "The size if long double is:   " << sizeof(long double) << endl;

cout << "Size of wchar_t : " << sizeof(wchar_t) << endl;

return 0;

}
```

Output

```
Size of char : 1

Size of bool : 1

Size of int : 4

Size of short int : 2

Size of long int : 4

Size of float : 4

Size of double : 8

Size of long double : 16

Size of wchar_t : 4
```

Debugging

There are two kinds of errors you'll run into when writing C++ programs: **compilation errors** and **runtime errors**. Compilation errors are problems raised by the compiler, generally resulting from violations of the syntax rules or misuse of types. These are often caused by typos and the like. Runtime errors are problems that you only spot when you run the program: you did specify a legal program, but it doesn't do what you wanted it to. These are usually more tricky to catch, since the compiler won't tell you about them.

Operators

We can perform arithmetic calculations with operators. Operators act on expressions to form a new expression. For example, we could replace "Hello, world!\n" with (4 + 2) / 3, which would cause the program to print the number 2. In this case, the + operator acts on the expressions 4 and 2 (its operands).

Operator types:

- Mathematical: +, -, *, /, and parentheses have their usual mathematical meanings, including using -for negation. % (the modulus operator) takes the remainder of two numbers: 6%5 evaluates to 1.
- Logical: used for "and," "or," and so on. More on those in the next lecture.
- Bitwise: used to manipulate the binary representations of numbers. We will not focus on these.