Object Oriented Programming

Lecture 2

Engr. Sara Rehmat, MS(CS)

Variables

- A variable is a name given to a memory location.
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable affect that memory location.
- In C++, all the variables must be declared before use.
  - You have to tell the compiler the type of the variable.
  - C++ is a strongly typed language
- A variable name can consist of alphabets (both upper and lower case), numbers and the underscore ‘_’ character. However, the name must not start with a number.

Variables

```
/ Declaring a single variable

type variable_name;


// Declaring multiple variables:

type variable1_name, variable2_name, variable3_name;
```

Types of Variables

Primitive Data Types:

These data types are built-in or predefined data types and can be used directly by the user to declare variables.

Primitive data types available in C++ are:

Integer

Character

Boolean

Floating Point

Double Floating Point

Valueless or Void

Wide Character

Datatype Modifiers

- As the name implies, datatype modifiers are used with the built-in data types to modify the length of data that a particular data type can hold.
- Data type modifiers available in C++ are:
  - Signed
  - Unsigned
  - Short
  - Long

```cpp
#include <iostream>

using namespace std;

int main() {
    cout << "Size of char : "<< sizeof(char) << 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 << "Size of wchar_t : "<< sizeof(wchar_t) << endl;

    return 0;
}
```

Type Casting

- Conversion of an expression of a given type into another type is called as type casting.
- Type Casting is a mechanism which enables a variable of one datatype to be converted to another datatype.
- When a variable is typecast into a different type, the compiler basically treats the variable as of the new data type.
- Two types:
  - Implicit or automatic
  - Explicit or given

Implicit Casting

- Compiler will automatically change one type of data into another.
- For instance, if you assign an integer value to a floating-point variable, the compiler will insert code to convert the int to a float.
- Typecasting should always be used in right order (low to higher datatype).
- Typecasting in wrong places may result in loss of precision, which the compiler can signal with a warning. for example like for instance truncating a float when typecasting to an int.
- **short int -> int -> unsigned int ->long int -> unsigned long int -> float -> double -> long double**

Explicit Casting

- Explicit Casting allows you to make this type conversion explicit, or to force it when it wouldn't normally happen.
- To perform type casting, put the desired type including modifiers (like double) inside parentheses to the left of the variable or constant you want to cast. In C++, there are two ways we can perform generic type casting –
  - `type (expression)`
  - `(type) expression`

Operators in C++

- An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations.
- C++ is rich in built-in operators and provide the following types of operators
  - Arithmetic Operators
  - Relational Operators
  - Logical Operators
  - Bitwise Operators
  - Assignment Operators
  - Misc Operators

Arithmetic Operators

- +, -, *, /, %, ++, --

Relational Operators

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

A = 10,
B=20

Logical Operators

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A && B) is false.
||	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A || B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true.

A= True,
B= False

Bitwise Operators

- Bitwise operator works on bits and perform bit-by-bit operation.

p	q	p & q	p | q	p ^ q
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Bitwise Operators

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands.	(A & B) will give 12 which is 0000 1100
|	Binary OR Operator copies a bit if it exists in either operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A ^ B) will give 49 which is 0011 0001
~	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(~A) will give -61 which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240 which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15 which is 0000 1111

A= 60, B =13

Assignment Operator

- =, +=, -=, *=, /=, <<=, >>= , &=, …..

Functions in C++

- A function is a group of statements that together perform a task.
- A **function declaration** tells the compiler about a function's name, return type, and parameters.
- A **function definition** provides the actual body of the function.
- General form of C++ definition is as follows

```
return_type function_name( parameter list ) {

   body of the function

}
```