# VARIABLE AND OPERATORS

## Variable Scope in C++

A scope is a region of the program and broadly speaking there are three places, where variables can be declared −

Inside a function or a block which is called local variables,

In the definition of function parameters which is called formal parameters. Outside of all functions which is called global variables.

## Local Variables

Variables that are declared inside a function or block are local variables. They can be used only by statements that are inside that function or block of code. Local variables are not known to functions outside their own. Following is the example using local variables −

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

int main () {
  // Local variable declaration:
  int a, b;
  int c;

  // actual initialization
  a = 10;
  b = 20;
  c = a + b;

  cout << c;

  return 0;
}
```

## Global Variables

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the life-time of your program.

A global variable can be accessed by any function. That is, a global variable is available for use throughout your entire program after its declaration. Following is the example using global and local variables −

Live Demo

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

// Global variable declaration:
int g;

int main () {
  // Local variable declaration:
  int a, b;

  // actual initialization
  a = 10;
  b = 20;
  g = a + b;

  cout << g;

  return 0;
}
```

A program can have same name for local and global variables but value of local variable inside a function will take preference. **For example** −

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

// Global variable declaration:
int g = 20;

int main () {
  // Local variable declaration:
  int g = 10;

  cout << g;

  return 0;
}
```

When the above code is compiled and executed, it produces the following result −

**Initializing Local and Global Variables**

When a local variable is defined, it is not initialized by the system, you must initialize it yourself. Global variables are initialized automatically by the system when you define them as follows −

| Data Type | Initializer |
|---|---|
| int | 0 |
| char | '\0' |
| float | 0 |
| double | 0 |
| pointer | NULL |

It is a good programming practice to initialize variables properly, otherwise sometimes program would produce unexpected result.

## 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 provides the following type of operators −

- Arithmetic Operators
- Relational Operators
- Logical Operators
- Bitwise Operators
- Assignment Operators
- Misc Operators

This chapter will examine the arithmetic, relational, logical, bitwise, assignment and other operators one by one.

## Arithmetic Operators

There are following arithmetic operators supported by C++ language −

Assume variable A holds 10 and variable B holds 20, then −

**Show Examples**

| Operator | Description | Example |
|---|---|---|
| + | Adds two operands | A + B will give 30 |
| - | Subtracts second operand from the first | A - B will give -10 |
| * | Multiplies both operands | A * B will give 200 |
| / | Divides numerator by de-numerator | B / A will give 2 |

| | | |
|---|---|---|
| % | Modulus Operator and remainder of after an integer division | B % A will give 0 |
| ++ | Increment operator, increases integer value by one | A++ will give 11 |
| -- | Decrement operator, decreases integer value by one | A-- will give 9 |

## Relational Operators

There are following relational operators supported by C++ language

Assume variable A holds 10 and variable B holds 20, then −

**Show Examples**

| 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 | (A <= B) is true. |

| | | |
|---|---|---|
| | equal to the value of right operand, if yes then condition becomes true. | |

## Logical Operators

There are following logical operators supported by C++ language.

Assume variable A holds 1 and variable B holds 0, then −

**Show Examples**

| 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. |

## Bitwise Operators

Bitwise operator works on bits and perform bit-by-bit operation. The truth tables for &, |, and ^ are as follows −

| 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 |

Assume if A = 60; and B = 13; now in binary format they will be as follows −

A = 0011 1100

B = 0000 1101

---

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A  = 1100 0011

The Bitwise operators supported by C++ language are listed in the following table. Assume variable A holds 60 and variable B holds 13, then −

**Show Examples**

| 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 |

## Assignment Operators

There are following assignment operators supported by C++ language −

**Show Examples**

| Operator | Description | Example |
|---|---|---|

| | | |
|---|---|---|
| = | Simple assignment operator, Assigns values from right side operands to left side operand. | C = A + B will assign value of A + B into C |
| += | Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand. | C += A is equivalent to C = C + A |
| -= | Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand. | C -= A is equivalent to C = C - A |
| *= | Multiply AND assignment operator, It multiplies right operand with the left operand and assign the result to left operand. | C *= A is equivalent to C = C * A |
| /= | Divide AND assignment operator, It divides left operand with the right operand and assign the result to left operand. | C /= A is equivalent to C = C / A |
| %= | Modulus AND assignment operator, It takes modulus using two operands and assign the result to left operand. | C %= A is equivalent to C = C % A |
| <<= | Left shift AND assignment operator. | C <<= 2 is same as C = C << 2 |
| >>= | Right shift AND assignment operator. | C >>= 2 is same as C = C >> 2 |
| &= | Bitwise AND assignment operator. | C &= 2 is same as C = C & 2 |
| ^= | Bitwise exclusive OR and assignment operator. | C ^= 2 is same as C = C ^ 2 |
| \|= | Bitwise inclusive OR and assignment operator. | C \|= 2 is same as C = C \| 2 |

## Misc Operators

The following table lists some other operators that C++ supports.

| Sr.No | Operator & Description |
|---|---|
| 1 | **sizeof**<br>sizeof operator returns the size of a variable. For example, sizeof(a), where 'a' is integer, and will return 4. |
| 2 | **Condition ? X : Y**<br>Conditional operator (?). If Condition is true then it returns value of X otherwise returns value of Y. |
| 3 | **,**<br>Comma operator causes a sequence of operations to be performed. The value of the entire comma expression is the value of the last expression of the comma-separated list. |
| 4 | **. (dot) and -> (arrow)**<br>Member operators are used to reference individual members of classes, structures, and unions. |
| 5 | **Cast**<br>Casting operators convert one data type to another. For example, int(2.2000) would return 2. |
| 6 | **&**<br>Pointer operator & returns the address of a variable. For example &a; will give actual address of the variable. |
| 7 | *<br>Pointer operator * is pointer to a variable. For example *var; will pointer to a variable var. |

## Operators Precedence in C++

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator −

For example x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

**Show Examples**

| Category | Operator | Associativity |
|---|---|---|
| Postfix | () [] -> . ++ - - | Left to right |
| Unary | + - ! ~ ++ - - (type)* & sizeof | Right to left |
| Multiplicative | * / % | Left to right |
| Additive | + - | Left to right |
| Shift | << >> | Left to right |
| Relational | < <= > >= | Left to right |
| Equality | == != | Left to right |
| Bitwise AND | & | Left to right |
| Bitwise XOR | ^ | Left to right |
| Bitwise OR | \| | Left to right |
| Logical AND | && | Left to right |
| Logical OR | \|\| | Left to right |
| Conditional | ?: | Right to left |
| Assignment | = += -= *= /= %=>>= <<= &= ^= \|= | Right to left |
| Comma | , | Left to right |