

# **OBJECT ORIENTED PROGRAMMING LAB**



**Lab Manual # 01**

**Operators, Strings, Math and Booleans in C++**

**Instructor: Iqra Rehman**

**Semester: Summer 2024**

**Course Code: CL1004**

**Fast National University of Computer and Emerging Sciences  
Peshawar**

**Department of Computer Science**

# OBJECT ORIENTED PROGRAMMING LANGUAGE

## Table of Contents

Operators .....	1
Types of Operators .....	1
Arithmetic Operators in C++ .....	2
Relational/Comparison Operators .....	4
Logical Operators .....	5
C++ Strings .....	9
String Concatenation .....	10
Append .....	11
C++ String Length .....	13
Access Strings .....	13
Changing String Characters .....	14
User Input String .....	14
C++ Math .....	16
C++ <cmath> Headers .....	16
Other Math Functions .....	17
C++ Booleans .....	19
Boolean Values .....	19
Boolean Expressions .....	20
Local and Global Variables .....	21
1. Local Variables .....	21
2. Global Variable .....	22
References .....	23

## Operators

- ❖ Operator is a symbol which is used to perform some operation. Operators are used to perform operations on variables and values. In the example below, we use the + **operator** to add together two values:
- ❖ `int x = 100 + 50;`
- ❖ Although the + operator is often used to add together two values, like in the example above, it can also be used to add together a variable and a value, or a variable and another variable:

### Example

```
int sum1 = 100 + 50;    // 150 (100 + 50)
int sum2 = sum1 + 250;   // 400 (150 + 250)
int sum3 = sum2 + sum2;  // 800 (400 + 400)
```

## Types of Operators

1. Unary operators
2. Binary operators
3. Ternary operators

### 1. Unary Operator

1. Increment (++)
2. Decrement (--)
3. Negation (!)

### 2. Binary Operator

1. Arithmetic (+, -, \*, /, %)
2. Relational (>, <, >=, <=, !=, ==)
3. Logical (&&, ||)
4. Assignment (=)
5. Arithmetic Assignment operator (+=, -=, \*=, /=, %=)

### 3. Ternary Operator

Conditional operator (?:)

**Example** (condition) ? statement 1 : statement 2;

```
int result= (n1>n2) ? n1 : n2;
```

```
#include<iostream>

using namespace std;

int main()
{
    int time = 20;
    string result = (time < 18) ? "Good day." : "Good evening.";
    cout << result;
    return 0;
}

/*
Output
Output: Good evening.
*/
```

## Arithmetic Operators in C++

Arithmetic operators are used to perform common mathematical operations.

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value by another	x / y
%	Modulus	Returns the division remainder	x % y

## Adding two integers

```
#include<iostream>
using namespace std;
int main()
{
    int n1, n2, sum;
    cout<<"Enter first number:\t";
    cin>>n1;
    cout<<"Enter 2nd number:\t";
    cin>>n2;
    sum=n1+n2;
    cout<<"The sum is:\t"<<sum<<endl;
}
/*
Output
Enter first number: 3
Enter 2nd number: 6
The sum is : 9
*/
```

## Assignment Operator

Assignment operators are used to assign values to variables. In the example below, we use the assignment operator (=) to assign the value 10 to a variable called x:

### Example

```
int x = 10;
```

The **addition assignment** operator (+=) adds a value to a variable:

### Example

```
int x = 10;
x += 5;      // x = x+5
```

A list of all arithmetic assignment operators:

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3

## Relational/Comparison Operators

- ❖ Comparison operators are used to compare two values.
- ❖ **Note:** The return value of a comparison is either true (1) or false (0).
- ❖ In the following example, we use the **greater than** operator (>) to find out if 5 is greater than 3:

### Example

```
int x = 5;
int y = 3;
cout << (x > y); // returns 1 (true) because 5 is greater than 3
```

A list of all relational operators:

Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y

>	Greater than	$x > y$
<	Less than	$x < y$
>=	Greater than or equal to	$x \geq y$
<=	Less than or equal to	$x \leq y$

## Logical Operators

Logical operators are used to determine the logic between variables or values:

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	$x < 5 \ \&\& \ x < 10$
	Logical or	Returns true if one of the statements is true	$x < 5 \    \ x < 4$
!	Logical not	Reverse the result, returns false if the result is true	$!(x < 5 \ \&\& \ x < 10)$

## Increment and Decrement Operators

### 1) Increment Operator:

The operators that is used to add 1 to the value of a variable is called increment operator.

### 2) Decrement Operator :

The operator that is used to subtract 1 from the value of a variable is called decrement operator.

### 1) The Increment Operator (++)

- ❖ The increment operator is represented by a double plus (++) sign.
- ❖ It is used to add 1 to the value of an integer variable.

- ❖ This variable can be used before or after the variable name.
- ❖ For example, to add 1 to a value of variable xy, it is normally written as

```
xy = xy + 1;
```

- ❖ By using increment operator “++” it is written as

```
xy++
```

- ❖ The increment operator can be written either before or after the variable.
- ❖ If it is written before the variable, it is known as **prefixing**.
- ❖ If it is written after the variable, it is known as **post fixing**.
- ❖ Prefix and postfix operators have different effects when they are used in expressions.

### i) Prefix Increment Operator

When an increment operator is used in prefix mode in an expression, it adds 1 to the value of the variable before the values of the variable is used in the expression.

```
#include<iostream>

using namespace std;

int main()
{
    int a=2;
    int b=3;
    int c=2;
    int result=a+b(++c);
    cout<<"Result is: "<<result;
    cout<<"\nValue of c is: "<<c;

}
/*
Output
Result is:8
nValue of c is : 3
*/
```

- ❖ In the above program, 1 will be added to the value of c before it is used in the expression.
- ❖ Thus, after execution, the result will be equal to 8 and the value of c will be 3.



## ii) Postfix Increment Operator

- ❖ When an increment operator is used in postfix mode in an expression, it adds 1 to the value of the variable after the value of the variable is used in the expression.
- ❖ For Example, if in the above example, increment operator is used in postfix mode, the result will be different. The statement will be shown below:
  - `result =a + b + c++;`
- ❖ In this case, 1 will be added to the value of c after its existing value has been used in the expression. Thus, after execution, the result will be equal to 7 and the value of c will be 3.

```
#include<iostream>

using namespace std;

int main()
{
    int a=2;
    int b=3;
    int c=2;
    int result=a+b+(c++);
    cout<<"Result is: "<<result;
    cout<<"\nValue of c is: "<<c;

}
/*
Output
Result is:7
nValue of c is : 3
*/
```

## 2) The Decrement Operator (--)

- ❖ The decrement operator is represented by a double minus (--) sign.
- ❖ It is used to subtract 1 from the value of an integer variable.
- ❖ This variable can be used before or after the variable name.
- ❖ For example, to subtract 1 from the value of variable xy, the decrement statement is written as

xy--; or --xy;

### i) Prefix Decrement Operator

- ❖ When decrement operator is used in prefix mode in an expression, it subtracts 1 from the value of the variable **before** the values of the variable is used in the expression.

```
#include<iostream>

using namespace std;

int main()
{
    int a=2;
    int b=3;
    int c=2;
    int result=a+b+(--c);
    cout<<"Result is: "<<result;
    cout<<"\nValue of c is: "<<c;

}

/*
Output
Result is:6
nValue of c is : 1
*/
```

- ❖ In the above program, 1 will be subtracted from the value of **c** before it is used in the expression.
- ❖ Thus, after execution, the result will be equal to 6 and the value of **c** will be 1.

### ii) Postfix Decrement Operator

- ❖ When a decrement operator is used in postfix mode in an expression, it subtracts 1 from the value of the variable **after** the values of the variable is used in the expression.
- ❖ For Example, if in the above example, decrement operator is used in postfix mode, the result will be different. The statement will be shown below:

result =a + b + c--;

- ❖ In this case, 1 will be subtracted from the value of **c** after its existing value has been used in the expression. Thus, after execution, the result will be equal to 7 and the value of **c** will be 1.

```
#include<iostream>

using namespace std;

int main()
{
    int a=2;
    int b=3;
    int c=2;
    int result=a+b+(c--);
    cout<<"Result is: "<<result;
    cout<<"\nValue of c is: "<<c;

}

/*
Output
Result is:7
nValue of c is : 1
*/
```

## C++ Strings

- ❖ Strings are used for storing text.
- ❖ A string variable contains a collection of characters surrounded by double quotes:
- ❖ Example
- ❖ Create a variable of type string and assign it a value:
- ❖ `string greeting = "Hello";`
- ❖ To use strings, you must include an additional header file in the source code, the `<string>` library:

### ❖ Example

- ❖ `// Include the string library`  
`#include <string>`
- `// Create a string variable`  
`string greeting = "Hello";`

```
#include <iostream>
#include <string>
```

```
using namespace std;
int main() {
    string greeting = "Hello";
    cout << greeting;
    return 0;
}

/*
Output
Hello
*/
```

## String Concatenation

The + operator can be used between strings to add them together to make a new string. This is called concatenation:

### Example

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName + lastName;
cout << fullName;
```

```
#include<iostream>

using namespace std;

int main()
{
    string firstName="John ";
    string lastName="Doe";
    string fullName =firstName+lastName;
    cout<<fullName;
}

/*
Output
John Doe
*/
```

In the example above, we added a space after firstName to create a space between John and Doe on output. However, you could also add a space with quotes (" " or ' '):

### Example

```
string firstName = "John";
string lastName = "Doe";
string fullName = firstName + " " + lastName;
cout << fullName;
```

```
#include<iostream>
using namespace std;
int main()
{
    string firstName="John";
    string lastName="Doe";
    string fullName =firstName+" "+lastName;
    cout<<fullName;
}

/*
Output
John Doe
*/
```

## Append

A string in C++ is actually an object, which contain functions that can perform certain operations on strings. For example, you can also concatenate strings with the append() function:

### Example

```
string firstName = "John ";
string lastName = "Doe";
string fullName = firstName.append(lastName);
cout << fullName;
```

It is up to you whether you want to use + or append(). The major difference between the two, is that the append() function is much faster.

However, for testing and such, it might be easier to just use +.

```
#include<iostream>

using namespace std;
```

```
int main()
{
    string firstName="Asad ";
    string lastName="Ullah";

    string fullName =firstName.append(lastName);
    cout<<fullName;
}

/*
Output
Asad Ullah
*/
```

## Adding Numbers and Strings

### WARNING!

C++ uses the + operator for both **addition** and **concatenation**.

Numbers are added. Strings are concatenated.

If you add two numbers, the result will be a number:

### Example

```
int x = 10;
int y = 20;
int z = x + y;    // z will be 30 (an integer)
```

If you add two strings, the result will be a string concatenation:

### Example

```
string x = "10";
string y = "20";
string z = x + y;    // z will be 1020 (a string)
```

If you try to add a number to a string, an error occurs:

Example

```
string x = "10";  
int y = 20;  
string z = x + y;
```

## C++ String Length

To get the length of a string, use the `length()` function:

### Example

```
#include <iostream>  
#include <string>  
using namespace std;  
int main() {  
    string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
    cout << "The length of the txt string is: " << txt.length();  
    return 0;  
}  
/*  
Output  
The length of the txt string is: 26  
*/
```

**Tip:** You might see some C++ programs that use the `size()` function to get the length of a string. This is just an alias of `length()`. It is completely up to you if you want to use `length()` or `size()`:

```
#include <iostream>  
#include <string>  
using namespace std;  
int main() {  
    string txt = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";  
    cout << "The length of the txt string is: " << txt.size();  
    return 0;  
}  
/*  
Output  
The length of the txt string is: 26  
*/
```

## Access Strings

You can access the characters in a string by referring to its index number inside square brackets `[]`.

This example prints the first character in myString:

**Example**

```
string myString = "Hello";  
cout << myString[0];  
// Outputs H
```

Note: String indexes start with 0:

[0] is the first character.

[1] is the second character, etc.

## Changing String Characters

To change the value of a specific character in a string, refer to the index number, and use single quotes:

**Example**

```
string myString = "Hello";  
myString[0] = 'J';  
cout << myString;  
  
// Outputs Jello instead of Hello
```

```
#include<iostream>  
using namespace std;  
int main()  
{  
    string myString="Aalim";  
  
    myString[0]='H';  
  
    cout<<myString;  
    return 0;  
}  
/*  
Output  
Halim  
*/
```

## User Input String



It is possible to use the extraction operator >> on cin to display a string entered by a user:

### Example

```
string firstName;  
cout << "Type your first name: ";  
cin >> firstName; // get user input from the keyboard  
cout << "Your name is: " << firstName;
```

**// Type your first name: John**

**// Your name is: John**

However, cin considers a space (whitespace, tabs, etc) as a terminating character, which means that it can only display a single word (even if you type many words):

### Example

```
#include<iostream>  
using namespace std;  
  
int main()  
{  
    string fullName;  
    cout<<"Enter full name: ";  
    cin>>fullName;  
    cout<<"Your name is: "<<fullName;  
    return 0;  
}  
/*  
Output  
Enter full name: John Doe  
Your name is: John  
*/
```

- ❖ From the example above, you would expect the program to print "John Doe", but it only prints "John".
- ❖ That's why, when working with strings, we often use the getline() function to read a line of text. It takes cin as the first parameter, and the string variable as second:

```
#include<iostream>  
using namespace std;  
  
int main()  
{  
    string fullName;  
    cout<<"Enter full name: ";
```

```
    getline(cin, fullName);  
    cout<<"Your name is: "<<fullName;  
    return 0;  
}  
/*  
Output  
Enter full name: John Doe  
Your name is: John Doe  
*/
```

## C++ Math

C++ has many functions that allows you to perform mathematical tasks on numbers.

### Max and min

The max(x,y) function can be used to find the highest value of x and y:

#### Example

```
cout << max(5, 10);
```

And the min(x,y) function can be used to find the lowest value of x and y:

#### Example

```
cout << min(5, 10);
```

### Max and min Example

```
#include<iostream>  
using namespace std;  
int main()  
{  
    cout<<"Maximum Number is: "<<max(5,8)<<endl;  
    cout<<"Minimum Number is: "<<min(5,8)<<endl;  
  
    return 0;  
}  
/*  
Output  
Maximum Number is: 8  
Minimum Number is: 5  
*/
```

## C++ <cmath> Headers

Other functions, such as sqrt (square root), round (rounds a number) and log (natural logarithm), can be found in the <cmath> header file:

Example

```
// Include the cmath library
#include <cmath>
cout << sqrt(64);
cout << round(2.6);
cout << log(2);
```

### C++ <cmath> Headers Example

```
#include<iostream>
using namespace std;
#include<cmath>
int main()
{
    cout<<"Square root of 64 is: "<<sqrt(64)<<endl;
    cout<<"log of 2 is: "<<log(2)<<endl;
    cout<<"Round of 2.6 is: "<<round(2.6)<<endl;
    return 0;
}

/*
Output
Square root of 64 is:8
log of 2 is: 0.693147
Round of 2.6 is:3
*/
```

## Other Math Functions

A list of other popular Math functions (from the <cmath> library) can be found in the table below:

Function	Description
abs(x)	Returns the absolute value of x
acos(x)	Returns the arccosine of x
asin(x)	Returns the arcsine of x

atan(x)	Returns the arctangent of x
cbrt(x)	Returns the cube root of x
ceil(x)	Returns the value of x rounded up to its nearest integer
cos(x)	Returns the cosine of x
cosh(x)	Returns the hyperbolic cosine of x
exp(x)	Returns the value of $E^x$
expm1(x)	Returns $e^x - 1$
fabs(x)	Returns the absolute value of a floating x
fdim(x, y)	Returns the positive difference between x and y
floor(x)	Returns the value of x rounded down to its nearest integer
hypot(x, y)	Returns $\sqrt{x^2 + y^2}$ without intermediate overflow or underflow
fma(x, y, z)	Returns $x*y+z$ without losing precision
fmax(x, y)	Returns the highest value of a floating x and y
fmin(x, y)	Returns the lowest value of a floating x and y
fmod(x, y)	Returns the floating point remainder of x/y
pow(x, y)	Returns the value of x to the power of y
sin(x)	Returns the sine of x (x is in radians)
sinh(x)	Returns the hyperbolic sine of a double value

tan(x)	Returns the tangent of an angle
tanh(x)	Returns the hyperbolic tangent of a double value

## C++ Booleans

Very often, in programming, you will need a data type that can only have one of two values, like:

- YES / NO
- ON / OFF
- TRUE / FALSE

For this, C++ has a bool data type, which can take the values true (1) or false (0).

## Boolean Values

A boolean variable is declared with the bool keyword and can only take the values true or false:

### Example

```
bool isCodingFun = true;
```

```
bool isFishTasty = false;
```

```
cout << isCodingFun; // Outputs 1 (true)
```

```
cout << isFishTasty; // Outputs 0 (false)
```

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

int main()
{
    bool isCodingFun=true;
    bool isFishTasty=false;

    cout<<isCodingFun<<endl;
    cout<<isFishTasty<<endl;
    return 0;
}
```

```
/*  
Output  
1  
0  
*/
```

- ❖ From the example above, you can read that a true value returns 1, and false returns 0. However, it is more common to return boolean values from boolean expressions (see next page).

## Boolean Expressions

A Boolean expression is a C++ expression that returns a boolean value: 1 (true) or 0 (false).

You can use a comparison operator, such as the greater than (>) operator to find out if an expression (or a variable) is true:

### Example

```
int x = 10;  
int y = 9;  
cout << (x > y); // returns 1 (true), because 10 is higher than 9
```

```
#include<iostream>  
using namespace std;  
#include<cmath>  
  
int main()  
{  
    int n1=10;  
    int n2=7;  
    cout<<(n1>n2);  
    return 0;  
}  
  
/*  
Output  
1  
*/
```

**Or even easier:**

### Example

```
cout << (10 > 9); // returns 1 (true), because 10 is higher than 9
```

In the examples below, we use the equal to (==) operator to evaluate an expression:

Example

```
int x = 10;
cout << (x == 10); // returns 1 (true), because the value of x is equal to 10
```

Example

```
cout << (10 == 15); // returns 0 (false), because 10 is not equal to 15
```

## Local and Global Variables

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 are called global variables.

### 1. Local Variables

Local variables can be used only by statements that are inside that function or block of code. Local variables are not known to functions on their own.

```
#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;
}
```

```
/*  
Output  
This will give the output:  
30  
*/
```

## 2. Global Variable

Global variables are defined outside of all the functions, usually on top of the program. The global variables will hold their value throughout the lifetime of your program. A global variable can be accessed by any function.

```
#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;  
}  
/*  
Output  
This will give the output:  
30    */
```

### Note

- ❖ A program can have the same name for local and global variables but the value of a local variable inside a function will take preference.
- ❖ For accessing the global variable with same name, you'll have to use the scope resolution operator.

```
#include <iostream>  
using namespace std;  
// Global variable declaration:  
int g = 20;
```



```
int main () {  
    // Local variable declaration:  
    int g = 10;  
  
    cout<<"Local variable g: "<<g<<endl;    // Local  
    cout<<"Global variable g: "<<::g<<endl; // Global  
    return 0;  
}  
/*
```

**Output:**

```
Local variable g: 10  
Global variable g: 20  
*/
```

## References

<https://beginnersbook.com/2017/08/cpp-data-types/>

[http://www.cplusplus.com/doc/tutorial/basic\\_io/](http://www.cplusplus.com/doc/tutorial/basic_io/)

<https://www.w3schools.com/cpp/default.asp>

<https://www.javatpoint.com/cpp-tutorial>

<https://www.geeksforgeeks.org/object-oriented-programming-in-cpp/?ref=lbp>

<https://www.programiz.com/>

<https://ecomputernotes.com/cpp/>