Chapter Three
Operators and Functions

3.1. Introduction

In the end of this chapter, we will able to program formulas that contain:

1. Arithmetic operators.
2. Relational and Logical operators.
3. Bitwise operators.
4. Ternary operators.
5. Cast operators.
6. Functions.

3.2. Operators

Once we know of the existence of variables and constants, we can begin to operate with them. For that purpose, C++ integrates operators. Unlike other languages whose operators are mainly keywords, operators in C++ are mostly made of signs that are not part of the alphabet but are available in all keyboards. This makes C++ code shorter and more international, since it relies less on English words, but requires a little of learning effort in the beginning.

You do not have to memorize all the content of this page. Most details are only provided to serve as a later reference in case you need it.

3.2.1. Assignment Operator (=)

The assignment operator assigns a value to a variable.

```
 a = 5;
```

This statement assigns the integer value 5 to the variable *a*. The part at the left of the assignment operator (=) is known as the *l-value* (left value) and the right one as the *r-value* (right value). The l-value has to be a variable whereas the r-value can be either *a* constant, *a* variable or the result of an operation or any combination of these. The most important rule when assigning is the *right-to-left* rule: The assignment operation always takes place from right to left, and never the other way:

```
a = b;
```

This statement assigns to variable *a* (the l-value) the value contained in variable *b* (the r-value). The value that was stored until this moment in (*a*) is not considered at all in this operation, and in fact that value is lost.

Consider also that we are only assigning the value of (*b*) to (*a*) at the moment of the assignment operation. Therefore a later change of b will not affect the new value of a.

Ex: Let us have a look at the following code. The evolution of the content stored in the variables as comments have been included:

```
// assignment operator
#include <iostream>
using namespace std;
int main ()
{
int a, b;           // a:?,   b:?
a = 10;             // a:10, b:?
b = 4;              // a:10, b:4
a = b;              // a:4,   b:4
b = 7;              // a:4,   b:7
cout << "a:";
cout << a <<'\n';
cout << " b:";
cout << b;
return 0;
}
```

O/P

```
a:4
b:7
```

This code will give us as result that the value contained in *a* is 4 and the one contained in *b* is 7. Notice how a was not affected by the final modification of *b*, even though we declared *a* = *b* earlier (that is because of the *right-to- left rule*).

A property that C++ has over other programming languages is that the assignment operation can be used as the r value (or part of an r value) for another assignment operation.

Ex:

a = 2 + (b = 5);
is equivalent to:
b = 5;
a = 2 + b;

That means: first assign 5 to variable b and then assign to a the value 2 plus the result of the previous assignment of b (i.e. 5), leaving a with a final value of 7.

The following expression is also valid in C++:
a = b = c = 5;
It assigns 5 to the all the three variables: a, b and c.

3.2.2. Arithmetic operators:

Arithmetic operators divided to two parts:

1. Simple arithmetic operators.
2. Compound arithmetic operators.

1. Simple arithmetic operators.

The five arithmetical operations supported by the C++ language are:

+	**Addition**
-	**Subtraction**
*	**Multiplication**
/	**Division**
%	**Modulo**

Operations of addition, subtraction, multiplication and division literally correspond with their respective mathematical operators. The only one that you might not be so used to see is *modulo*; whose operator is the percentage sign (%). Modulo is the operation that gives the remainder of a division of two values. For example, if we write:

```
a = 11 % 3;
```

The variable a will contain the value 2, since 2 is the remainder from dividing 11 between 3.

2. Compound arithmetic operators.

The compound operators illustrated in table below:

Expression	Means
+=	**Plus equal**
-=	**Minus equal**
*=	**Multiplication equal**
/=	**Division equal**
%=	**Modula equal**
>>=	**Shift right equal**
<<=	**Shift left equal**
&=	**And equal**
^=	**Ex-or equal**
|=	**Or equal**
++	**Increase by one**
--	**Decrease by one**

When we want to modify the value of a variable by performing an operation on the value currently stored in that variable, we can use compound operators:

Expression	Equivalent to
value += increase;	**value = value + increase;**
a -= 5;	**a = a - 5;**
a /= b;	**a = a / b;**
price *= units + 1;	**price = price * (units + 1);**

Ex:

```
// compound assignment operators
#include <iostream>
using namespace std;
int main ()
{
int a, b=3;
a = b;
a+=2;           // equivalent to a=a+2
cout << a;
return 0;
}
o/p : 5
```

The increase operator (++) and the decrease operator (--) increase or reduce by one the value stored in a variable. They are equivalent to +=1 and to -=1, respectively. Thus:
c++; equivalent to c+=1; and equivalent to c=c+1;

The three of them increase by one the value of c. In the early C compilers, the three previous expressions probably produced different executable code depending on which one was used. Nowadays, the compiler generally does this type of code optimization automatically, thus the three expressions should produce exactly the same executable code.

A characteristic of these operators are that it can be used both as a prefix and as a suffix. That means that it can be written either before the variable identifier (++a) or after it (a++). They may have an important difference in their meaning.

In the case that the increase operator is used as a prefix (++a) the value is increased before the result of the expression is evaluated, and therefore the increased value is considered in the outer expression.

In case that it is used as a suffix (a++) the value stored in (a) is increased after being evaluated, and therefore the value stored before the increase operation is evaluated in the outer expression. Notice the difference:

Example 1	Example 2
B=3;	B=3;
A=++B;	A=B++;
// A contains 4, B contains 4	// A contains 3, B contains 4

In Example 1, B is increased before its value is copied to A. While in Example 2, the value of B is copied to A and then B is increased.

3.2.3. Relational and equality operators (==, !=, >, <, >=, <=)

In order to evaluate a comparison between two expressions we can use the relational and equality operators. The result of a relational operation is a Boolean value that can only be true or false, according to its Boolean result.

A list of the relational and equality operators that can be used in C++:

Expression	Means
==	**Equal to**
!=	**Not Equal to**
>	**Greater than**
<	**Less than**
>=	**Greater than or equal to**
<=	**Less than or equal to**

We may want to compare two expressions, for example, to know if they are equal or if one is greater than the other is.

Ex:

```
(7 == 5) // evaluates to false.
(5 > 4) // evaluates to true.
(3 != 2) // evaluates to true.
(6 >= 6) // evaluates to true.
(5 < 5) // evaluates to false.
```

Instead of using only numeric constants, we can use any valid expression, including variables. Suppose that a=2, b=3 and c=6,

```
(a == 5)         // evaluates to false since a is not equal to 5.
(a*b >= c)       // evaluates to true since (2*3 >= 6) is true.
(b+4 > a*c)      // evaluates to false since (3+4 > 2*6) is false.
((b=2) == a)     // evaluates to true.
```

* Note that The operator = (one equal sign) is not the same as the operator == (two equal signs), the first one is an assignment operator (assigns the value at its right to the variable at its left) and the other one (==) is the equality operator that compares whether both expressions in the two sides of it are equal to each other.

* In the last expression ((b=2) == a), we first assigned the value 2 to b and then we compared it to a, that also stores the value 2, so the result of the operation is true.

3.2.4. Logical operators (!, &&, ||)

The (!) Operator is to perform the Boolean operation NOT, it has only one operand, located at its right, and the only thing that it does is to inverse the value of it, producing false if its operand is true and true if its operand is false. Basically, it returns the opposite Boolean value of evaluating its operand. For example:

Ex:

```
!(5 == 5)  // evaluates to false because the expression at its right (5 == 5) is true.
!(6 <= 4)  // evaluates to true because (6 <= 4) would be false.
!true      // evaluates to false
!false     // evaluates to true.
```

The logical operators (&&) and (||)are used when evaluating two expressions to obtain a single relational result. The operator && corresponds with Boolean logical operation AND. This operation results true if both its two operands are true, and false otherwise. The following panel shows the result of operator (&&) evaluating the expression a && b:

A	B	A && B
True	true	True
False	true	False
True	false	False
False	false	False

The operator || corresponds with Boolean logical operation OR. This operation results true if either one of its two operands is true, thus being false only when both operands are false themselves. Here are the possible results of a || b:

A	B	A || B
true	true	true
false	true	true
true	false	true
false	false	false

Ex:

```
( (5 == 5) && (3 > 6) )        // evaluates to false ( true && false ).
( (5 == 5) || (3 > 6) )        // evaluates to true ( true || false ).
```

3.2.5. Conditional operator.

The conditional operator evaluates an expression returning a value if that expression is true and a different one if the expression is evaluated as false. Its format is:

Condition? result1: result2

If condition is true, the expression will return result1, if it is not it will return result2.

```
7==5 ? 4 : 3       // returns 3, since 7 is not equal to 5.
7==5+2 ? 4 : 3     // returns 4, since 7 is equal to 5+2.
5>3 ? a : b        // returns the value of a, since 5 is greater than 3.
a>b ? a : b        // returns whichever is greater, a or b.
```

Ex:

```
// conditional operator
#include <iostream>
using namespace std;
int main ()
{
int a,b,c;
a=2;
b=7;
c = (a>b) ? a : b;
cout << c;
return 0;
}
```

In this example *a* was 2 and *b* was 7, so the expression being evaluated (*a*>*b*) was not true, thus the first value specified after the question mark was discarded in favor of the second value (the one after the colon) which was *b*, with a value of 7.

3.2.6. Bitwise Operators.

Bitwise operators modify variables considering the bit patterns that represent the values they store.

symbol	description	Operator equivalent
&	**AND**	**Bitwise AND**
|	**Bitwise Inclusive OR**	**OR**
^	**Bitwise Exclusive OR**	**XOR**
~	**Unary complement (bit inversion)**	**NOT**
<<	**Shift Left**	**SHL**
>>	**Shift Right**	**SHR**

***We will not discuss these operators because you will not read logic gets.**

3.2.7. Explicit type casting operator

Type casting operators allow you to convert a datum of a given type to another. There are several ways to do this in C++. The simplest one, which has been inherited from the C language, is to precede the expression to be converted by the new type enclosed between parentheses (()):

```
int i;
float f = 3.14;
i = (int) f;
```

The previous code converts the float number 3.14 to an integer value (3), where the remainder is lost.

3.2.8. sizeof() operator.

This operator accepts one parameter, which can be either a type or a variable itself and returns the size in bytes of that type or object.

```
a = sizeof (char);
```

This will assign the value 1 to (*a*) because char is a one-byte long type. The value returned by sizeof() is a constant, so it is always determined before program execution.

3.2.9. Comma operator (,)

The comma operator (,) is used to separate two or more expressions that are included where only one expression is expected. When the set of expressions has to be evaluated for a value, only the rightmost expression is considered.

Ex:

```
a = (b=3, b+2);
```

Would first assign the value 3to b, and then assign b+2to variable a. So, at the end, variable (*a*) would contain the value 5while variable bwould contain value 3.

3.3. Precedence of operators

When writing complex expressions with several operands, we may have some doubts about which operand is evaluated first and which later. For example, in this expression:
a = 5 + 7 % 2

we may doubt if it really means:
a = 5 + (7 % 2) // with a result of 6, or
a = (5 + 7) % 2// with a result of 0

The correct answer is the first of the two expressions, with a result of 6. There is an established order with the priority of each operator, and not only the arithmetic ones (those whose preference come from mathematics) but for all the operators which can appear in C++. From greatest to lowest priority, the priority order is as follows:

level	Operator	Description
1.	::	scope
2.	()	postfix
3.	sizeof	unary (prefix)
4.	(type)	indirection and reference
5.	.* ->*	(pointers)
6.	* / %	multiplicative
7.	+ -	additive
8.	<< >>	shift
9.	< > <= >=	relational
10.	== !=	equality
11.	&	bitwise AND
12.	^	bitwise XOR
13.	|	bitwise OR
14.	&&	logical AND
15.	||	logical OR
16.	?:	conditional
17.	= *= /= %= += -= >>= <<= &= ^= |=	Compound assignment
18.	,	comma

All these precedence levels for operators can be manipulated or become more legible by removing possible ambiguities using parentheses signs (and).

Ex:

```
a = 5 + 7 % 2;
Might be written either as:
a = 5 + (7 % 2);
or
a = (5 + 7) % 2;
```

Depending on the operation that we want to perform. So if you want to write complicated expressions and you are not completely sure of the precedence levels, always include parentheses. It will also become a code easier to read.

3.4. Functions of C++

The table below summarize some of important mathematical functions available in C++.

Function	Porous
ceil (x)	*Return value larger than or equal to x, where x is float*
floor (x)	*Return value of integer x, where x is float (neglect floating points)*
abs (x)	*Return absolute value of x, where x is integer value.*
fabs (x)	*Return absolute value of x, where x is float value.*
sin (x)	*Return sine value of x, where x is radian value*
cos (x)	*Return cosine value of x, where x is radian value*
tan (x)	*Return tangent value of x, where x is radian value*
asin (x)	*Return arc sine value of x, where x is between (-1 ,1)*
acos(x)	*Return arc cosine value of x, where x is between (-1 ,1)*
atan (x)	*Return arc tangent value of x, where x is between (-1 ,1)*
log (x)	*Return the natural logarithm of x, where x > 0*
log10 (x)	*Return the base 10 logarithm of x, where x > 0*
exp (x)	*Return the exponential of x with based e.*
sqrt (x)	*Return the square root of x, where x >= 0*
pow (x,y)	*Return x raised to the y power.*

Ex:

Write a C++ program to compute and display the distance between two points (x1,y1) and (x2,y2) in Cartesian coordinates. Where :

$$distance = \sqrt[2]{(x1 - x2)^2 + (y1 - y2)^2}$$

Sol:

```
#include <iostream>
using namespace std;
int main ()
{
double x1=5.2, y1=3.4, x2=1.32, y2=4.45, distance;
distance=sqrt(pow((x1-x2),2)+(pow((y1-y2),2)));
cout << "distance=" << distance <<'\n';
return 0;
}
```

Questions of chapter three

Q1: Write a C++ program to implement the equation: $y = \sqrt[5]{16}$.

Q2: If you know that the radius of the circle inscribed in triangle with sides a, b, c is:

$$r = \sqrt{\frac{(s-a)(s-b)(s-c)}{s}} \qquad where\;\; s = \frac{1}{2}(a+b+c)$$

Input the triangle sides, and then find and display the radius (r).

Q3: Write C++ program to enter any three numbers, then find and print maximum number between them using conditional operator.

Q4: Suppose we have the following variable declarations in our program:
int x=10 , y=3 , z=5; and double u=-3.7 , v=10.25 , w=12.37;

What will be the values computed for each of the following arithmetic expressions or arithmetic assignment expression statements?

a) $x\%y * z$
b) $x++\%--y+7/z$
c) $((x-6)*(x-6)+y*y)/(z*z)$
d) $x{+}{=}\,y+z$
e) $(int)v\%(int)w-v$
f) $ceil(u)+floor(v-2)$
g) $sqrt\,(pow\,(w,2))$

Q5: Write a C++ program to enter any two numbers (x and y), then find and print the values of Z and W from the following equations:

a) $Z = Sin(x^2) + Tan^3(\frac{y}{3})$

b) $W = Ln(\frac{x}{y})^4 - \sqrt{x^2+y^2}$

Q6: Write a C++ program to evaluate the following:

$$y = \begin{cases} 3\sinh^{-1}(\frac{x}{2}) + x\cosh^{-1}(\frac{\sqrt{x^3}}{2}) \\ x\sinh^{-1}(\frac{1}{2}) + 2\cosh^{-1}(\frac{\sqrt{3}}{2}) \\ \cos(2\cosh^{-1}(\frac{x}{2})) \end{cases}$$

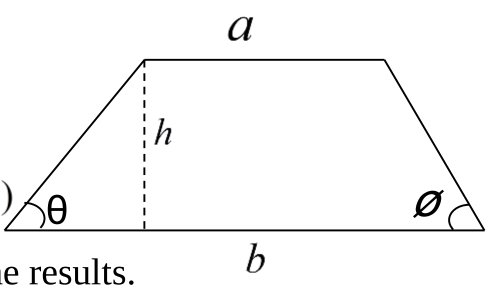
Find y for each entered value for the three equations at the same time.

Hint: $\sinh^{-1}(u) = \log(u + \sqrt{u^2 + 1})$ *and* $\cosh^{-1}(u) = \log(u + \sqrt{u^2 - 1})$

Q7: Write a VB program to find and print the area and the perimeter (circumference) for the shape below from the following equations:

$$Area = \frac{1}{2}h(a+b)$$

$$Perimeter = a + b + h(\csc(\theta) + \csc(\phi))$$



Enter the unknown variables, and then print the results.

Q8: Write a VB program to find and print Y from the following equation:

$$Y = \frac{\cosh(x) + \sinh^2(x) - 4\cosh(x).\sinh(x)}{4x^2 + 2\sinh(x) - \cosh^2(x) - 6}$$

Using the relations:

Sinh(x)=$\dfrac{e^x - e^{-x}}{2}$ and Cosh(x)=$\dfrac{e^x + e^{-x}}{2}$