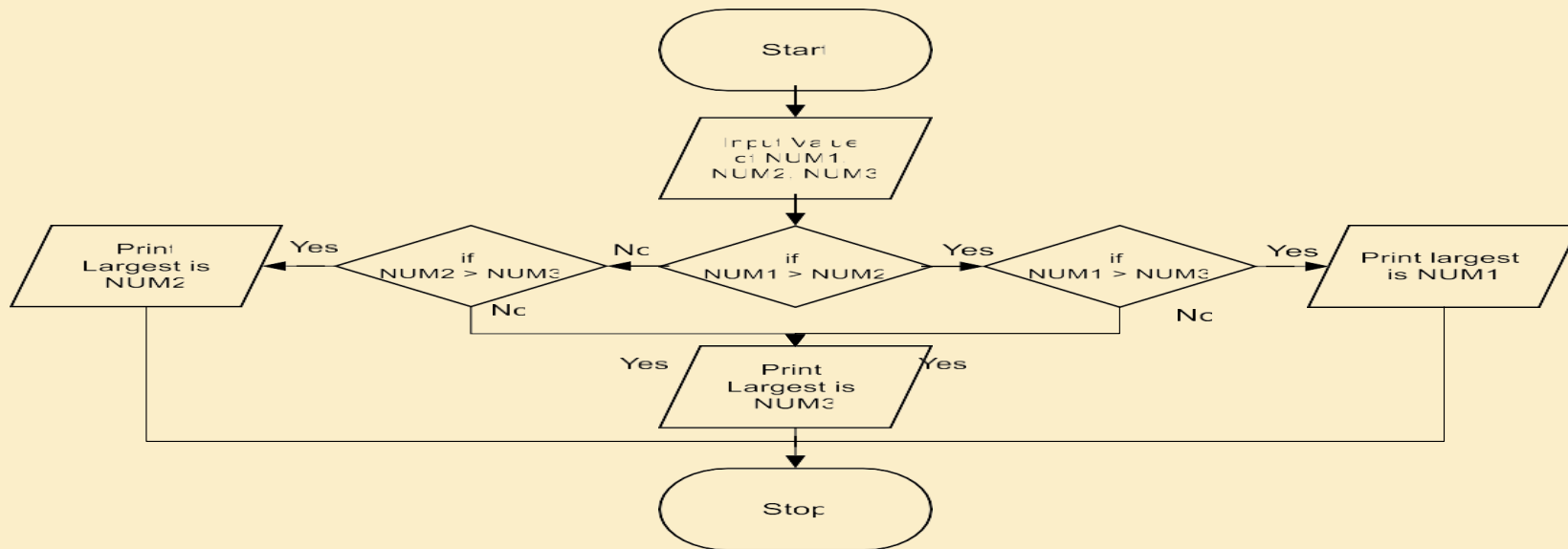


Introduction to C++

COMP101- Part 2



Dr. Zaid Ameen

Arithmetic Expression

..

Arithmetic expressions in C++ must be entered into the computer in straight-line form. Thus, expressions such as “a divided by b” must be written as a / b , so that all constants, variables and operators appear in a straight line. Parentheses are used in C++ expressions in the same manner as in algebraic expressions. For example, to multiply a times the quantity $b + c$ we write $a * (b + c)$.

The main statement in C++ for carrying out computation and assigning values to variables is the assignment statement. For example the following assignment statement:

```
average = (a + b)/2;
```

The general form of an assignment statement is:

```
result = expression ;
```

..

C++ applies the operators in arithmetic expressions in a precise order determined by these rules of operator precedence, which are generally the same as those in algebra:

1. Operators in expressions contained within pairs of parentheses are evaluated first. Parentheses are said to be at the “highest level of precedence.” In cases of nested, or embedded, parentheses, such as the operators in the innermost pair of parentheses are applied first.
2. Multiplication, division and modulus operations are applied next. If an expression contains several multiplication, division and modulus operations, operators are applied from left to right. Multiplication, division and modulus are said to be on the same level of precedence.
3. Addition and subtraction operations are applied last. If an expression contains several addition and subtraction operations, operators are applied from left to right. Addition and subtraction also have the same level of precedence.

The set of rules of operator precedence defines the order in which C++ applies operators. When we say that certain operators are applied from left to right, we are referring to the associativity of the operators. For example, the addition operators (+) in the expression associate from left to right, so $a + b$ is calculated first, then c is added to that sum to determine the whole expression's value. We'll see that some operators associate from right to left. Figure 1.2 summarizes these rules of operator precedence. We expand this table as we introduce additional C++ operators.

Operator(s)	Operation(s)	Order of evaluation (precedence)
()	Parentheses	Evaluated first. If the parentheses are nested, the expression in the innermost pair is evaluated first. [Caution: If you have an expression such as $(a + b) * (c - d)$ in which two sets of parentheses are not nested, but appear “on the same level,” the C++ Standard does not specify the order in which these parenthesized sub expressions will be evaluated.]
*, /, %	Multiplication, Division, Modulus	Evaluated second. If there are several, they’re evaluated left to right.
+, -	Addition, Subtraction	Evaluated last. If there are several, they’re evaluated left to right.
++/--	Increment	The increment operator increases the value of its operand by 1. The
	/decrement operators	operand must have an arithmetic or pointer data type, and must refer to a modifiable data object. Similarly, the decrement operator decreases the value of its modifiable arithmetic operand by 1. Pointers values are increased (or decreased) by an amount that makes them point to the next (or previous) element adjacent in memory.

Algebra: $y=mx+b$

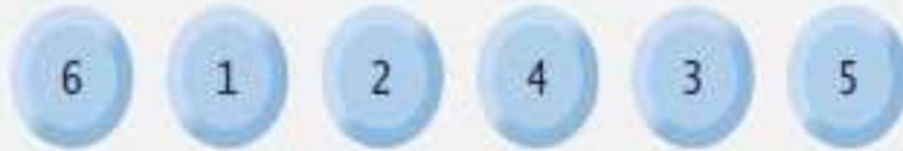
C++: $y=m*x+b$

Algebra: $z = pr \% q + w/x - y$

C++: $z = p * r \% q + w / x - y;$



$y = a * x * x + b * x + c;$




Suppose variables a , b , c and x in the preceding second-degree polynomial are initialized as follows:

$a = 2$, $b = 3$, $c = 7$ and $x = 5$. Figure 1.3 illustrates the order in which the operators are applied and the final value of the expression. As in algebra, it's acceptable to place unnecessary parentheses in an expression to make the expression clearer. These are called redundant parentheses. For example, the preceding assignment statement could be parenthesized as follows:

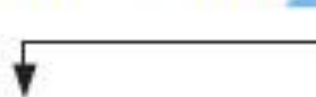
Step 1. $y = 2 * 5 * 5 + 3 * 5 + 7;$ (Leftmost multiplication)

$2 * 5$ is 10




Step 2. $y = 10 * 5 + 3 * 5 + 7;$ (Leftmost multiplication)

$10 * 5$ is 50




Step 3. $y = 50 + 3 * 5 + 7;$ (Multiplication before addition)

$3 * 5$ is 15




Step 4. $y = 50 + 15 + 7;$ (Leftmost addition)

$50 + 15$ is 65



Step 5. $y = 65 + 7;$ (Last addition)

$65 + 7$ is 72



There are many mathematics functions in C++ programming language. So, header <math.h> declares a set of functions to compute common mathematical operations and transformations:

Function	Prototype	Purpose
abs(x)	int abs(int x);	returns the absolute value of an integer.
fabs(x)	double fabs(double x);	returns the absolute value of a floating point number
ceil(x)	double ceil(double x);	rounds up to a whole number cout << ceil(11.2); (prints 12) (not normal rounding)
floor(x)	double floor(double x);	rounds down to a whole number cout<<floor(11.5); (prints 11) (not normal rounding)
pow (x,y)	double pow (double x, double y);	calculates x to the power of y. If x is negative, y must be an integer. If x is zero, y must be a positive integer.
pow10(x)	double pow10 (int x);	calculates 10 to the power of x.
sqrt (x)	double sqrt(double x);	calculates the positive square root of x. (x is >=0)
fmod(x,y)	double fmod(double x, double y);	returns floating point remainder of x/y with same sign as x. Y cannot be zero. Because the modulus operator(%) works only with integers, this function is used to find the remainder of floating point number division.

$\cos(x)$	cosine of x
$\sin(x)$	sine of x
$\tan(x)$	tangent of x
$\arccos(x)$	arc cosine x
$\arcsin(x)$	arc sine of x
$\arctan(x)$	arc tangent x
$\cosh(x)$	hyperbolic cosine of x

$\sinh(x)$	hyperbolic sine of x
$\tanh(x)$	hyperbolic tangent of x
$\exp(x)$	exponential function
$\log(x)$	natural logarithm
$\log_{10}(x)$	base 10 logarithm
$\sin(x)$	sine of x
$\tan(x)$	tangent of x

Logical Expression

Very often, you need to compare two values before deciding on the action to be taken, e.g., if mark is more than or equal to 50, print "PASS". C++ provides six comparison operators (or relational operators):

Operator	Description	Usage	Example (x=5, y=8)
==	Equal to	$expr1 == expr2$	$(x == y) \rightarrow \text{false}$
!=	Not Equal to	$expr1 != expr2$	$(x != y) \rightarrow \text{true}$
>	Greater than	$expr1 > expr2$	$(x > y) \rightarrow \text{false}$
>=	Greater than or equal to	$expr1 >= expr2$	$(x >= 5) \rightarrow \text{true}$
<	Less than	$expr1 < expr2$	$(y < 8) \rightarrow \text{false}$
<=	Less than or equal to	$expr1 <= expr2$	$(y <= 8) \rightarrow \text{true}$

Logical Expression

In C++, these comparison operations returns a bool value of either false (0) or true (1 or a non-zero value). Each comparison operation involves two operands, e.g., $x \leq 100$. It is invalid to write $1 < x < 100$ in programming. Instead, you need to break out the two comparison operations $x > 1$, $x < 100$, and join with with a logical AND operator, i.e., $(x > 1) \&\& (x < 100)$, where $\&\&$ denotes AND operator. C++ provides four logical operators (which operate on boolean operands only):

Operator	Description	Usage
$\&\&$	Logical AND	$expr1 \&\& expr2$
$\ \ $	Logical OR	$expr1 \ \ expr2$
$!$	Logical NOT	$!expr$
\wedge	Logical XOR	$expr1 \wedge expr2$

The truth tables are as follows:

And (&&)	True	True	True
	True	False	False
	False	True	False
	False	False	False
Or ()	True	True	True
	True	False	True
	False	True	True
	False	False	False
Not (!)	True	False	
	False	True	
Xor (^)	True	True	False
	True	False	True
	False	True	True
	False	False	False

Example

```
// Return true if x is between 0 and 100 (inclusive)
(x >= 0) && (x <= 100)
// wrong to use 0 <= x <= 100
// Return true if x is outside 0 and 100 (inclusive)
(x < 0) || (x > 100) //or
!((x >= 0) && (x <= 100))
// Return true if year is a leap year
// A year is a leap year if it is divisible by 4 but not by 100, or it is
divisible by 400.
((year % 4 == 0) && (year % 100 != 0)) || (year % 400 == 0)
```

Input and Output statements

C++ defines an input stream that retrieves data from the keyboard as a consecutive series of characters. The class or blueprint is called an **istream** and the specific instance that we use to obtain keyboard input is called **cin**. This definition of **cin**, as an instance of the **istream**, is parallel to our use of `cost` as an instance of the intrinsic double data type. It is the same with `cout` being an instance of the **ostream** class of data. The **istream** class and the definition of **cin** are contained in the header file **iostream.h**. The operator that causes transfer of data from the keyboard into our variable is called the extraction operator which is `>>`. The extraction operator can be thought of as extracting the next data item from the input stream of characters as they are entered on the keyboard. The extraction operator's syntax is similar to the insertion operator that is used to send data to the screen.

```
cin >> variable;
```

Assume that we have defined both **qty** and **cost** as an integer and a double as above. If we code **cin >> qty**; then the input stream waits until the user has entered their data and pressed the enter key. Just as the insertion operator can be chained with other insertion operators to output more than one item at a time, so can the extraction operator. For example, one can code: **cin >> qty >> cost**;

When inputting data from the keyboard, a program must always prompt the user notifying them what data is to be entered at this point. A prompt is nothing more than a simple **cout** line. For example, one could code:

```
cout << "Enter the quantity: ";
```

```
cin >> qty;
```

```
cout << "Enter the cost: ";
```

```
cin >> cost;
```


Notes

- endl: This function is used to print new line.
- The setw() or set width function can be used to set the total width for the next item being displayed. It applies solely and only to the next item to be displayed. The function takes one parameter, the total width of the next item. For numeric types, the values are right justified within the total specified width. A more optimum way to display the line is as follows.

```
cout << setw (4) << qty << setw (6) << cost << setw (7) << total  
<< endl;
```

This is in fact exactly what was used in the above proper columnar alignment example.

Program

```
#include <iostream.h>
```

```
int Integer;  
char aCharacter;  
char string [20];  
unsigned int NumberOfSons;
```

Global variables

```
main ()
```

```
{
```

```
    unsigned short Age;  
    float ANumber, AnotherOne;
```

Local variables

```
    cout << "Enter your age:"  
    cin >> Age;  
    ...
```

Instructions

```
}
```

Global variables can be referred to anywhere in the code, within any function, whenever it is after its declaration.

The scope of the **local variables** is limited to the code level in which they are declared. If they are declared at the beginning of a function (like in **main**) their scope is the whole **main** function. In the example above, this means that if another function existed in addition to `main()`, the local variables declared in **main** could not be used in the other function and vice versa.

In C++, the scope of a local variable is given by the block in which it is declared (a block is a group of instructions grouped together within curly brackets `{ }` signs). If it is declared within a function it will be a variable with function scope, if it is declared in a loop its scope will be only the loop, etc...

Examples:-

Write C++ program to print welcome message “Hello Word”?

```
#include <iostream>

int main ()
{
    cout << "Hello World!";
    return 0;
}
```

```
#include <iostream.h>

int main ()
{
    cout << "Hello World! ";
    cout << "I'm a C++ program";
    return 0;
}
```

Examples:-

Write C++ Program to output an integer, a floating point number and a character?

```
#include <iostream.h>
#include <conio.h>

void main()
{
    clrscr(); int x = 10;
    float y = 10.1; char z =
    'a';
    cout << "x = " << x << endl;
    cout << "y = " << y << endl;
    cout << "z = " << z << endl;
    getch();
}
```

Examples:-

Write C++ program to enter two integers and find their sum and average?

```
#include <iostream.h>
#include <conio.h>

void main()
{
clrscr(); int x = 10; int y = 2;
    int sum, difference, product, quotient;
    sum = x + y; difference = x - y;
    product = x * y; quotient = x / y;
    cout << "The sum of " << x << " & " << y << " is " << sum << "." << endl;
    cout << "The difference of " << x << " & " << "y <<    is " << difference
    << "." << endl;
    cout << "The product of " << x << " & " << y << " is " << product << "."
    << endl;
    cout << "The quotient of " << x << " & " << y << " is " << quotient << "."
    << endl;

    getch();
}
```


Examples:-

Write C++ program to enter two integers and find their sum and average?

```
#include <iostream.h>
#include <iostream.h>
#include <conio.h>

void main()
{
clrscr();
int x,y,sum;
float average;
cout << "Enter 2 integers : " << endl;
cin>>x>>y; sum=x+y;
average=sum/2;
cout << "The sum of " << x << " and " << y << " is " << sum <<
"." << endl;
cout << "The average of " << x << " and " << y << " is " <<
average << "." << endl;
getch();
}
```

Examples:-

Write Program to enter an integer and output the cube of that integer?

```
#include <iostream.h>
#include <conio.h>
void main()
{
    clrscr(); int a;
    cout << "Enter an integer : ";
    cin>>a;
    b=a*a*a;
    cout << "The cube of " << a << " is : " << b <<
    endl; getch();
}
```

Examples:-

Write C++ program to convert Converts gallons to liters

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

int main()
{
    float gallons, liters;

    cout << "Enter number of gallons: ";

    cin >> gallons; // Read the inputs from the
    user liters = gallons * 3.7854; // convert to
    liters cout << "Liters: " << liters << endl;

    return 0;
}
```

Examples:-

Write c++ program to find area of circle find area of circle?

```
#include <iostream.h>
int main()
{
    float Rad, Area;

    cout << "Enter Radius of circle: ";
    cin >> Rad; // Read the inputs from the user

    Area = 0.5* rad*rad*3.14;
    cout << "Area: " << Area << endl;

    return 0;
}
```

Examples:-

Write c++ program to find area of circle find area of circle?

```
#include <iostream.h>
int main()
{
    float Rad, Area;

    cout << "Enter Radius of circle: ";
    cin >> Rad; // Read the inputs from the user

    Area = 0.5* rad*rad*3.14;
    cout << "Area: " << Area << endl;

    return 0;
}
```

Examples:-

Write Program to enter your age and print if you should be in grade 10?

```
#include <iostream.h>
#include <conio.h>

void main()
{
    clrscr();
    int age;
    cout << "Enter your present age : " << endl;
    cin>>age;
    if(age==16)
    {
        cout << "Your present age is " << age << " years." << endl;
        cout << "You are of the right age for joining grade 10 !" << endl;
    }
    else
    {
        cout << "Your present age is " << age << " years." << endl;
        cout << "You are not of the right age for joining grade 10 !" << endl;
    }
    getch();
}
```


Exercises:

Write C++ program to convert days into years and weeks?

Write C++ program to convert temperatures from Celsius to

Fahrenheit and Vice Versa. $Ftemp=(1.8*Ctemp)+32$.

Write C++ program to compute area of rectangle?

Write C++ program to enter two values of the integer type and then swap one replace the other (ex. $X=3; Y=4 \rightarrow X=4; Y=3$)?