19PMAE06

PROGRAMMING WITH C++

M.Sc. MATHEMATICS

III - SEMESTER

M. POONGUZHALI

GUEST LECTURER

DEPARTMENT OF MATHEMATICS

GOVERNMENT ARTS AND SCIENCE COLLEGE

KOMARAPALAYAM - 638183

NAMAKKAL (DT).

Unit – II : Token, Expressions and control structures: Tokens – Keywords – Identifiers and Constants – Basic Data types – User defined Data types – Derived data types – Symbolic Constants in C++ - Scope resolution operator – Manipulators – Type cast operator – Expressions and their types – Special assignment expressions – Implicit Conversions – Operator Overloading – Operator precedence – Control Structure.

UNIT - II

TOKENS

The smallest individual units in a program are known as tokens. C++ has following tokens:

- Keywords
- Identifiers
- Constants
- Strings
- Operators

KEYWORDS

- The keywords implemented specific C++ language features.
- They are explicitly reserved identifiers and cannot be used as names for the program variables or other user-defined program elements. Table shows the keywords available in c++:

auto	extern	sizeof
break	float	static
case	for	struct
char	goto	switch
const	if	typedef
continue	int	union
default	long	unsigned
do	register	void
double	return	while
else	short	
enum	signed	

IDENTIFIERS

- Identifiers refer to the names of variables, functions, arrays, classes, etc.
- Each language has its own rules for naming these identifiers. The following rules are common to both C and C++:
    1. Only alphabetic characters, digits and underscores are permitted.
    2. The name cannot start with a digit.

3. Uppercase and lowercase letters are distinct.
4. A declared keyword cannot be used as a variable name.

- A major difference between C and C++ is the limit on the length of a name.
- While ANSI C recognizes only the first 32 characters in a name, ANSI C++ places no limit on its length and, therefore, all the character in a name is significant.
- Ex: int marks;

CONSTANTS

- Constants refer to fixed values that do not change during the execution of a program.
- C++ supports 2 types of constants. They are:
  - Non-Numeric constant
    - Single Character Constant

      It is a character enclosed within single quotes.

      Eg: char c='a';
    - String Constant

      It is sequence of alpha-numeric characters enclosed within double quotation mark whose maximum length is 255 characters.

      Eg: char arr[30]="SSM College of Arts and Science"

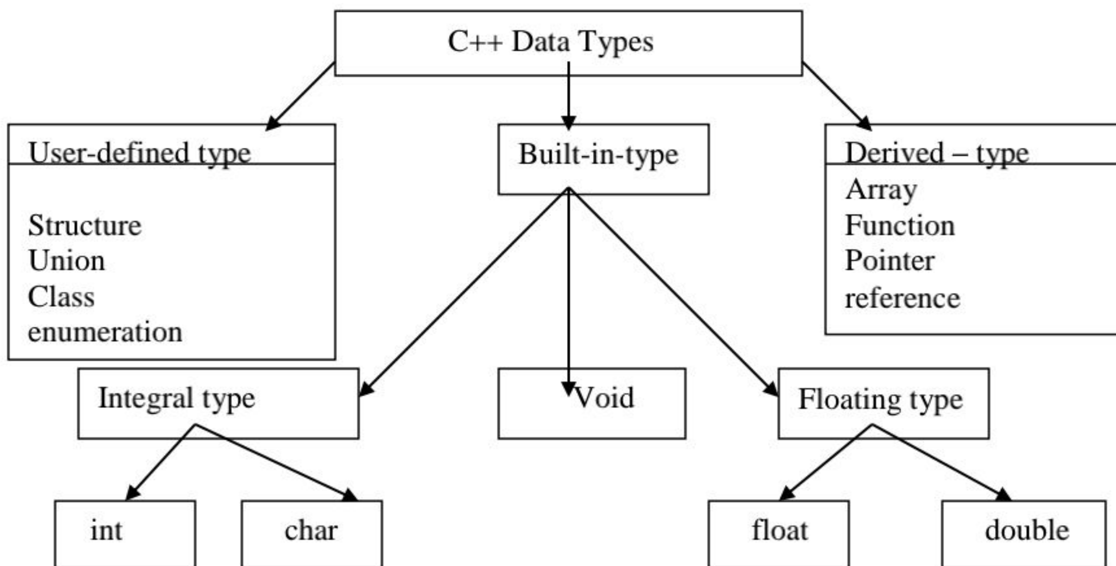
  - Numeric Constant

    There are four types of numeric constants.

    1. Integer constant – Integer – Short int, Long int.
    2. Floating point constant – single precision, double, long double.
    3. Octal constant – short , long
    4. Hexa decimal constant - short, long.

BASIC DATA TYPES IN C++

- Both C and C++ compilers support all the built in data types.
- With the exception of void, the basic data types may have several modifications preceding them to serve the needs of various situations.
- The modifiers signed, unsigned, long, and short may be applied to character and integer basic data types.
- The modifier long may also be applied to double.
- The type void was introduced in ANSI C. Two normal uses of void are (1) to specify the return type of function when it is not returning any value, and (2) to indicate empty argument list to a function.
- **Ex: void funct1 (void);**



Type	Bytes Required	Range
char	1	0 to 255
unsigned char	1	0 to 255
signed char	1	-128 to 127
int	2	-32768 to 32767
unsigned int	2	0 to 65535
long int	4	-2,147,483,648 to 2,147,483,647
long unsigned int	4	0 to 4,294,967,295
float	4	3.4E-38 to 3.4E+38
double	8	-1.7E-308 to 1.7E+308
long double	10	3.4E-4932 to 3.4E+4932

USER –DEFINED DATA TYPES

Structures and classes

- We have used user-defined data types such as structure and union in C.
- While these data types are legal in C++, some more features have been added to make them suitable for object-oriented programming.
- C++ also permits use to define another user-defined data type known as class which can be used, just like any other basic data type, to declare variables.

- The class variables are known as objects, which are the central focus of objects-oriented programming.

Enumerated Data Type

- An enumerated data type is another user-defined type which provides a way for attaching names to numbers, thereby increasing comprehensibility of the code,
- The enum keyword automatically enumerates a list of words by assigning then values 0, 1, 2, and so on.
- The facility provides an alternative means for creating symbolic constants.
- The syntax of an enum statement is

Ex:
```
enum shape{circle, square, triangle};
enum colour{red, blue, green, yellow};
enum position{off, on};
```

- The enumerated data types differ slightly in C++ when compared with those in ANSI C.
- In C++, the tag names shape, color, and position become new type names.

We can declare new variables.

Ex:
```
shape ellipse;        // ellipse is of type shape
color background;  // background is of type color
```

- C++ does not permit an int value to be automatically converted to an enum values.

Ex:
```
colour background=blue;  // Allowed
colour background=7;       // Error in c++
colour background=(colour) 7      // Ok
```

<u>DERIVED DATA TYPES</u>

Arrays

- The application of arrays in C++ is similar to that in C.
- The only exception is the way character arrays are initialized.
- When initializing a character array in ANSI C, the compiler will allow us to declare the array size as the exact length of the string constant.

Ex:
```
char string[3] =”xyz”;   // valid in ANSI C.
```

- It assumes that the programmer intends to leave out the null character \0 in the definition. But in C++, the size should be one larger than the number of characters in the string.

Ex:
```
char string [4] =”xyz”;    // valid ic C++.
```

Functions

- Functions have undergone major changes in C++. While some of these changes are simple, others require a new way of thinking when organizing our programs.

- Many of these modifications and improvement were driven by the requirements of the object oriented concept of C++.

Pointers

- Pointers are declared and initialized as in C. For example,

```
int *ip;        //int pointer
ip =&x;         // address of x assigned to ip
*ip=10          // 10 assigned to x through indirection
```

- C++ adds the concept of constant pointer and pointer to a constant.

```
char *const ptr1="GOOD"        // constant pointer
```

- We cannot modify the address that ptr1 is initialized to.

```
int const *ptr2=&m;       // pointer to a constant
```

- Ptr2 is declared as pointer to a constant. It can point to any variable of correct type, but the contents of what it points to cannot be changed.
- We can also declare both the pointer and the variable as constants in the following way:

```
const char * const cp="xyz";
```

- This statement declares cp as a constant pointer to the string which has been declared a constant.

SYMBOLIC CONSTANTS

There are two ways of creating symbolic constants in c++:

- Using the qualifier const, and
- Defining a set of integer constants using enum keyword.

- We can use const in a constant expression, such as

```
const int size =10;
char name[size];
```

- const allows us to create typed constants instead of having use #define to create constants that have no type information.

As with long and short, if we use the const modifier alone, it defaults to int.

Ex: **const size =10;**

means

```
const int size=10;
```

The named constants are just like variables except that their values cannot be changed.

- The scoping of const values differs.
- A const in C++ defaults to the internal linkage and therefore it is local to the file where it is declared.
- Another method of naming integer constants is by enumeration as under;

```
enum{x,y,z};
```

This defines x,y and z as integer constants with values 0,1 and 2 respectively.

OPERATORS IN C++

An operator is a symbol that tells the compiler to perform some operation. The data items that operators act upon are called operands. C++ is very rich in built-in operator. C++ operators are grouped into nine types as:

1. Arithmetic Operator
2. Relational Operator
3. Logical Operator
4. Assignment Operator
5. Increment and Decrement Operator
6. Conditional Operator
7. Bitwise Operator
8. Comma Operator
9. Sizeof Operator

1. Arithmetic Operator

Arithmetic operators are used to perform arithmetic calculations. There are five arithmetic operators in C. They are:

Operator	Purpose
+	Addition
-	Subtraction
*	Multiplication
/	Division
^	Exponentiation (Power)

2. Relational Operator

Relational operators are used to compare two values or expressions. There are six relational operators:

Operator	Meaning
<	Less than
<=	Less than or equal to
>=	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not Equal to

3. Logical Operator

Logical operators are used to combine two or more relational expressions. In addition to relational operators, C contains three logical operators. They are:

Operator	Meaning
&&	Logical AND
||	Logical OR
!	NOT

AND operator

Logical AND(&&) returns true when both operands are true and false otherwise.

Operand1	Operand2	Operand1 && Operand2
1	1	1
1	0	0
0	1	0
0	0	0

OR operator

Logical OR(||) returns true when either of its operands are true and returns false if both the operands are false.

Operand1	Operand2	Operand1 || Operand2
1	1	1
1	0	1
0	1	1
0	0	0

NOT Operator

Not is a complementation operator.

Operand1	!Operand
1	0
0	1

4. Assignment Operator

There are several different assignment operators in C++. All of them are used to form assignment expression, which assign the value of an expression to an identifier.

Syntax

variable = expression; // expression represents a constant, a variable or an expression.

Eg: **x=5;**

a=b;

5. Increment and Decrement Operator

C++ allows two useful unary operators generally not found in other computer languages. These are increment (++) and decrement operators. The operation ++ add 1 to its operand, and – subtract 1. Therefore the following are equivalent operations:

x=x+1; is the same as ++x; or x++;

x=x-1; is the same as –x; or x--;

The operator can be placed either before or after the operand. The operator is placed before the variable as in ++x or –x , it known as pre-incrementing and pre-decrementing, respectively. If the operator appears after the variable like x++ or x--, known as post incrementing and post-decrementing respectively.

6. Conditional Operator

This is also known as ternary operator, because it uses three expressions. It is an abbreviation of if-else statement and uses ? and : symbol in a format

condition ? expression 1 : expression 2;

First the condition is evaluated. If it is true then expression is executed otherwise the expression 2 is executed.

Ex: **min=x<y ?x:y;** // will assign the x value to min if x<y , otherwise y value to min.

The conditional expression operator is generally used only when the condition and expression are very simple.

7. Bitwise Operator

C++ allows us to access many operations that are normally served for programming at the assembly language level. The bitwise operators are given below:

Operator	Operation
~	One's Complement
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
>>	Right shift operator
<<	Left shift operator

8. Sizeof() Operator

The sizeof() operator give the size of its operand in bytes.

9. Comma Operator(,)

C uses comma operator two ways:

i. Comma operator is used to separate elements in the variable declaration.

ii. In the for loop, more than one variable can be initialized / incremented at a time using comma operator.

OTHER OPERATORS IN C++

C++ introduces some new operators. We have already seen two such operators, namely, the insertion operator <<, and the extraction operator>>. Other new operators are:

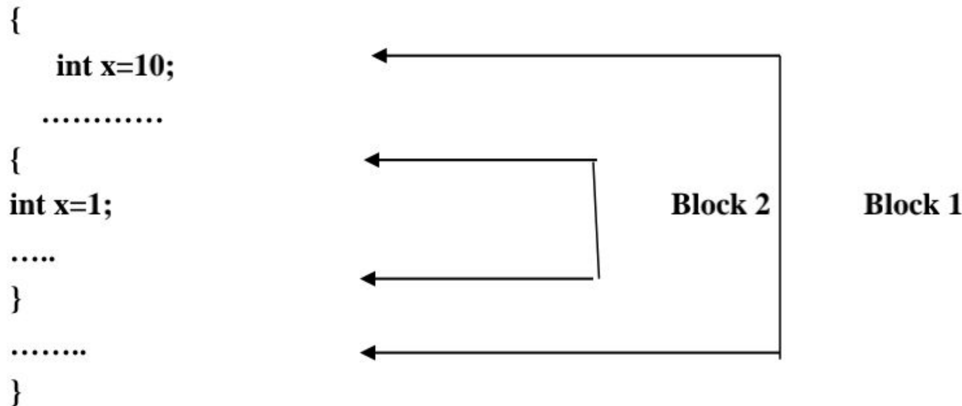
::	**Scope resolution operator**
::*	**pointer-to-member declaration**
->*	**pointer-to-member operator**
.*	**pointer-to-member operator**
new	**memory allocation operator**
delete	**memory release operator**
endl	**line feed operator**
setw	**field width operator**

SCOPE RESOLUTION OPERATOR

- Like, C++ is also a block-structured language. Blocks and scopes can be used in constructing programs. We know that the variable name can be used to have different meaning in different blocks.
- The scope of the variable extends form the point of its declaration till the end of the block containing the declaration. A variable declared inside a block is said to be local to that block.

- The two declarations of x refer to two different memory locations containing different values. Statements in the second block cannot refer to the variable x declared in the first block, and vice versa.

Example



MANIPULATORS

- Manipulators are operators that are used to format the data display.
- The most commonly used manipulators are endl and setw.
- The endl manipulator, when used in an output statement, causes a linefeed to be inserted. It has the same effect as using the newline character "\n".

Ex:

```
cout << " m = " << m <<  endl
      << " n = " <<  n << endl
      << " p = " << p << endl;
```

Would cause three lines of output, one for each variable. If we assume the values of the variable as 2597, 14 and 175 respectively, the output will appear as follows:

m=	2	5	9	7
n=	1	4		
p=	1	7	5	

It is important to note that this form is not the ideal output. If should rather appear as under:

```
m= 2597
n=    14
p=   175
```

Here the numbers are right justified. This form of output is possible only if we can specify a common field width for all the numbers and force them to be printed right justified. The setw manipulator does this job. It is used as follows:

```
cout << setw(4)<< sum << endl;
```

The manipulator setw(4) specifies a field width 4 for printing the value of the variable sum. This value is right justified within the field as shown below:

		2	5

TYPE CAST OPERATOR

It is used to convert a set of declared type to another declared type. It is easy to convert the values from one type to another type. In C++ conversion can be carried in two ways.

1. Converting by assignment
2. Using cast operator

Converting by assignment

It is a usual way of converting a value from one data type to another .By using assignment operator.

Cast operator

Converting by assignment operator is carried out automatically but may not get the desired result. The cast operator is a technique to convert one data type to another. The operator is used to force these conversion is known as type cast operator and the process is known as cast in. The syntax of cast operator is

(cast-type) expression;
cast-type (expression);

EXPRESSIONS AND THEIR TYPES

- An expression is a combination of operators, constants and variables arranged as per the rules of the languages.
- It may also include function calls which return values.
- An expression may consist of one or more operands, and zero or more operators to produce a value. Following are the types of expressions:
    1. Constant expressions
    2. Integral expressions
    3. Float expressions
    4. Pointer expressions
    5. Relational expressions
    6. Logical expressions
    7. Bitwise expressions

1. Constant Expressions

Constant expressions consist of only constant values**.**

Ex:

20+5

45/5

2. Integral Expressions

Integer expressions are those which produce integer results after implementing all the automatic and explicit type conversions.

Ex:

m * n – 5

m * 'X'

Where m and n are integer variable.

3. Float Expressions

Float Expressions are those which, after all conversions, produce floating-point results

Ex: **x+y;**

x * y/10

Where x and y are floating point variables.

4. Pointer Expressions

Point expressions produce address values.

Ex: **&m**

ptr+1

Where m is variable and ptr is a pointer.

5. Relational Expressions

Relational Expressions yield results of type bool which takes a value true or false.

Ex: **x<=y**

m+n>100

When arithmetic expressions are used on either side of a relational operator, they will be evaluated first and then the results compared. Relational expressions are also known as Boolean expressions.

6. Logical Expressions

Logical Expressions combine two or more relational expressions and produces bool type results.

Ex: **a>b && x==10**

x==10 || y==5

7. Bitwise Expressions

Bitwise Expressions are used to manipulate data at bit level. They are basically used for testing or shifting bits.

Ex: **x<<3 // shift three bit position to left**

y>>1 // shift one bit position to right

Shift operators are often used for multiplication and division by powers of two.

SPECIAL ASSIGNMENT EXPRESSION

Chained Assignment

x = y = 10 ; (or)

x = (y = 10);

First 10 is assigned to y and then to x. A chained statement cannot be used to initialize variables at the time of declaration. For example, the statement

float a = b = 12.34; // Wrong

is illegal. This may be written as

float a = 12.34, b = 12.34; // Correct

Embedded Assignment

x = (y = 50) + 10;

(y=50) is an assignment expression known as embedded assignment. Here the value 50 is assigned to y and then the result 50 + 10 is assigned to x. This is identical to

y = 50;

x = y + 10;

Compound Assignment

C++ supports a **compound assignment operator** which is a combination of the assignment operator with a binary arithmetic operator. For example, the simple assignment statement

x = x + 10;

may be written as

x += 10;

The operator += is known as **compound assignment operator** or **short-hand assignment operator**. The general form of compound assignment operator is:

variable1 op= variable2;

where op is a binary arithmetic operator. This means that

variable1 = variable1 op variable2;

IMPLICIT CONVERSION

- C++ permits mixing of constants and variables of different types in an expression.
- C++ automatically converts any intermediate values to the proper type so that the expression can be evaluated without loosing any significance.
- This automatic conversion is known implicit type conversion

- During evaluation it adheres to very strict rules of type conversion.
- If the operands are of different types, the 'lower' type is automatically converted to the higher type before the operation proceeds.

<u>Rules for Implicit Conversion</u>

- All **short** and **char** are automatically converted to **int;**
- If one of the operands is **long double**, the other will be converted to **long double** and the result will be **long double**.
- else, if one of the operands is **double,** the other will be converted to double and the result will be **double;**
- else, if one of the operands is **float,** the other will be converted to **float** and the result will be **float.**
- else, if one of the operands is **unsigned long int**, the other will be converted to **unsigned long int** and the result will be unsigned long int;

The following changes are introduced during the final assignment.

1. **float** to **int** causes truncation of the fractional part.
2. **double** to **float** causes rounding of digits.
3. **long int** to **int** causes dropping of the excess higher order bits.

OPERATOR OVERLOADING

Overloading means assigning different meanings to an operation depending on the context. C++ permits overloading of operators, thus allowing us to assign multiple meanings to operators. The input/output operators << and >> are good examples of operator overloading. Although the built-in definition of the << operator is for shifting of bits, it is also used for displaying the values of various data types. Thus the statement:

cout<< 75.86;

invokes the definition for displaying a double type value, and

cout<<"well done";

invokes the definition for displaying a char value.

OPERATOR PRECEDENCE AND ASSOCIATIVITY

- The precedence is used to determine how an expression involving more than one operator is evaluated.
- The operators at the higher level of precedence are evaluated first .
- The operators of the same precedence evaluated either from left to right or right to left depending on the level is known as associativity.
- Hierarchy of operators in 'c' are summarized below

- ➢ Any expression within parenthesis is first evaluated, if more than one pair of parenthesis are present, the innermost parenthesis is evaluated first.
- ➢ Unary operators are evaluated first in an expression.
- ➢ Then priority is given for multiplication and division.
- ➢ Then subtractions and addition are performed.
- ➢ Then relational operations are performed.
- ➢ Then equality checking is performed.
- ➢ Then logical operations are performed.
- ➢ Then the conditions are checked.
- ➢ Finally the assignment operation is carried out.

Table. Operator Precedence and Associativity

Operator	Associativity
::	Left to right
->. ( ) [ ] postfix ++ postfix – prefix ++ prefix -- ~ ! unary + unary -	Left to right
unary * unary & (type) sizeof new delete	Right to left
-> **	Left to right
* , / , %	Left to right
+ -	Left to right
<< >>	Left to right
< <= > >=	Left to right
= = !=	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 operator)	Left to right
= *= /= &= ^= |= ,	Right to left
,	Left to right

CONTROL STRUCTURES

- ➢ A control structure is a instruction, statement or group of statements which determines the sequence of execution of other statements.
- ➢ In C++, a large number of functions are used that pass messages, and process the data contained in objects.
- ➢ A function is set up to perform a task.

- When the task is complex, any different algorithms can be designed to achieve the same goal.
- The following three control structures:
  1. Sequence structure(straight line)
  2. Selection structure(branching)
  3. loop structure (iteration or repetition)

Branching statement

Following are the branching statements available in c++:

- Simple If
- If-Else
- Nested If-Else
- Else-If Ladder
- Switch

Simple If

An if statement has the form:

Syntax

```
if (condition)
{
  // code to execute if condition is true
}
Statement-x;
```

In an if statement, condition is a value or an expression that is used to determine whether to execute the statement inside the braces or not.

Ex:

```
if (a>b)
      big = a ;
big = b;
```

If-Else

An if-Else statement has the form:

Syntax

```
if (condition)
{
  // code to execute if condition is true
}
else
{
  // code to execute if condition is false
}
```

In an if statement, condition is a value or an expression that is used to determine which code block is executed, and the curly braces act as "begin" and "end" markers.

Ex:

```
if (a>b)
      big = a ;
else
      big = b;
```

Nested If-Else

A Nested if statement has the form:

Syntax

```
if (test condition-1)
{
        if(test condition-2);
        {
          statement-1;
        }
        else
        {
           statement-2;
        }
}
else
{
   statement-3;
}
Statement-x;
```

Ex:

```
if (a>b)
{
        if (a>c)
        {
                Big = a;
        }
        else
        {
                Big = c;
```

```
        }
    }
    else
    {
        if (b>c)
        {
            Big  = b;
        }
        else
        {
            Big = c;
        }
    }
```

Else-If Ladder

A multi path decision is a chain of ifs in which the statement associated with each else is an if.

Syntax

```
if(condition 1)
        Statement -1;
                else if(condition 2)
                 Statement -2;
                        else if(condition 3)
                                Statement -3;
                                else if(condition n)
                                        Statement –n;
                                        else
                                                Default-statement;
                                Statement –x;
```

- The conditions are evaluated from the top, downwards.
- True conditions is found , the statement associated with it is executed and the control is transferred to the statements-x;
- When all the n conductions become false, then the final else containing the default statement will be executed,

Ex:

```
if(marks>79)
        grade="honors";
    else if(marks>59)
            grade="first division";
        else if(marks>49)
            grade="second division";
                else if(marks>39)
                    grade="third division";
                else
                    grade="fail";
    cout << grade;
```

Switch statement

- The Switch statement is a selection statement that can be used instead of a series of If-Else statements. Switch statements are much better for complex expressions.
- A case label is the word case followed by a constant expression. An integral expression is called a switch expression is used to match one of the values on the case labels.
- Execution then continues sequentially from the matched label until the end of the switch statement is encountered or a break statement is encountered.

Syntax

```
switch(expression)
{
  case  value 1: statements;
                 break;
  case value 2:  statements;
                 break;
     …………..
     …………..
   default   : statements;
               break;
}
```

Ex:

```
switch (grade)
{
        case 'A': cout << "Great work. " << endl;
                break;
        case 'B': cout << "Good work. " << endl;
                break;
        case 'C': cout << "Passing work. " << endl;
                break;
        case 'D':
        case 'F': cout << "Unsatisfictory work. " << endl;
                break;
        default:  cout << grade << " is not a legal grade." << endl;
                break;
}
```

Looping Statements

Following are the looping statements available in c++:

- Do-While
- While
- For

Do-While

- The do-while is an exit-controlled loop. Based on a condition, the control is transferred back to a particular point in the program.

Syntax

```
do
{
        action1;
}
while (condition  is true);
action 2;
```

- The while loop makes a test of condition before the loop is executed. Therefore, the body of the loop may not be executed at all, if the condition is not satisfied at first attempt.
- In some situations it may be necessary to execute the body of the loop before the rest condition is performed, such a situation the do…while loop is useful.

- It is also repetitive control structure and executes the body of the loop once irrespective of the condition, and then it checks the condition and continues the execution until the condition becomes false.

Ex: Program for addition of numbers using do….while loop

```
#include<iostream.h>
void main()
{
int i=1,sum=0;
do
{
sum = sum + i;
i++;
}
while(i<=10);
cout << "sum of the numbers upto 10 is ….." << sum;
}
```

While

- The while loop is an entry controlled loop statement, means the condition is evaluated first and it is true, and then the body of the loop is executed.

Syntax

```
while(condition is true)
{
  action1;
}
action2;
```

- After executing the body of the loop, the condition is once again evaluated and if it is true, the body is executed once again, the process of repeated execution of the body of the loop continues until the condition finally becomes false and the control is transferred out of the loop.

Ex: Program for addition of numbers using while loop.

```
#include<iostream.h>
void main()
{
```

```
int i=1,sum=0;
while(i<=10)
{
sum = sum + i;
i++;
}
cout << "sum of the numbers upto 10 is ….." << sum;
}
```

For

- The For is an entry-enrolled loop and is used when an action is to be repeated for a predetermined number of times.
- For loop is another repetitive control structure, and is used to execute set of instructions repeatedly until the condition becomes false.
- The assignment, increment or decrement and condition checking is done in for statement only, where as other structures are not offered all these features in one statement.

Syntax

```
for (initialize counter; test condition; increment/decrement counter)
{
        Body of the loop;
}
```

For loop has three parts:

i) **Initialize counter** is used to initialize counter variable.
ii) **Test condition** is used to test the condition.
iii) **Increment /decrement counter** is used to increment or decrement counter variable.

If there is a single statement within the for loop, the blocking with braces is not necessary, if more than one statement includes in body of the loop, the statements within the body must be blocked with braces.

Ex: Program for addition of numbers using for loop.

```
#include<iostream.h>
void main()
{
int i,sum=0;
for(i=1;i<=10;i++)
sum = sum + i;
cout << "sum of the numbers upto 10 is ….." << sum;
}
```