OPERATORS IN C++

- All C operators are valid in C++ also
  - << : insertion operator
  - >> : put to operator
  - :: scope resolution operator
  - ::* pointer to member declaration
  - ->* pointer to member operator
  - .* pointer to member operator
  - delete : memory release operator
  - endl : line feed operator
  - new : memory allocation operator
  - setw: field width operator

- Additional meaning can be given to already existing operator

* Reference: Object oriented Programming with C++ -E Balaguruswamy ,TMH publications 6thedition,2015

- **scope resolution operator**
  - The operator allows global version of a variable.
  - This operator allows to identify to which a member function belongs to

Ex. : {

```
        int x=10;
        {
          int x=1;
        }
    }
int m=10;
int main()
{
        int m=20;
                {
                        int m=30;
                        cout<<“m=“<<m<<endl;
                        cout<<“global variable is “<<::m<<endl;
                }
        cout<<“m=“<<m<<endl;
         cout<<“global variable is “<<::m<<endl;
Return 0;
}
```

- Memory management operator:
  - malloc()
  - calloc()
  - free()
  - new – to create a new object
  - delete – to delete an object
  - An object created using ‘new’ will remain in existence until it is explicitly destroyed using ‘delete’
  - Pointer variable=new data type;
  - int *p;
  - p=new int;
  - *p=25; assign 25 to the newly created int object
  - float *q;
  - q=new float;
  - *q=30.5; assign 30.5 to the newly created float object

  - int *p=new int[25]; creates memory space for 25 integers
  - p[0] – refers to first element, p[1] – refers to second element,…..
  - int *p=new int(25); initializes memory with value 25 using new operator

- Multi dimensional arrays can be created with new; all array dimensions must be supplied.
- int *arrayptr=new int [3][4][5]; legal
- int *arrayptr=new int [m][5][5]; legal – first dimension may be an integer , all others must be constants
- int *arrayptr=new int [][4][5]; illegal

- ‘new’ can also be used to create class objects dynamically

- When the object is no longer needed, it is destroyed to release the memory space for reuse.
  - delete pointer variable
  - delete p;
  - delete q;
  - delete []p; releases the entire array pointed to by p.

EXPRESSIONS

- Constant expressions – has only constant values
  - 10+5
  - ‘x’
  - 20/3-10;
- Integral Expressions-produces integer results after implementing all the automatic and explicit conversions
  - M;
  - M*n-1
  - M*’a’;
  - 5+int(2.0)

  M and n are integer variables
- Float expressions-produce floating point results after all conversions
  - M;
  - M*n-1
  - M*’a’;
  - 5+float(2)

- Pointer expressions
  - Produce address value
  - &m; ptr; ptr+1;
- Relational expression- results of bool type (true or false)
  - X<=y
  - A+b==c+d
  - M+n>10
- Logical expressions
- A>b && x==10
- X==10||y==5

- Bitwise expression
  - X<<3;
  - Y>>1

- Special assignment expression
  - X=(y=10); valid
  - X=y=10; invalid

- Y=10;
- X=y+10;
  - Same as x=(y=10)+10;
- Compound assignment
  - X+=10; same as x=x+10;

- **Control structure**
  - Sequence structure (straight line)
  - Selection structure (branching)
  - Loop structure (iteration or repetition)

- **if statement:**

```
If (true)
{
Action 1;
}
Action 2;
Action 3;
```

- **if….else:**

```
if (true)
{
Action 1;
}
else
{action 2;
}
Action 3;
```

- switch….case
- switch (variable)

```
{
Case 1:statement;
Break;
Case 2: { s1;s2;break;}
 default: s3;
}
```

- **do…….while**

  do

  {action 1;

  }

  while (condition is true);

- **while statement**

```
while (condition)
{ action 1;
}
action 2;
```

- **for statement**

```
for (initial value; condition; increment/decrement)
{
Statements;
}
Action 2;
```

```
#include<iostream>
#include<conio.h>
using namespace std;
int main()
{
    int i,j,n,fact,sign=-1;
    float x,p,sum=0.0;
    cout<<"enter the value of x"<<endl;
    cin>>x;
    cout<<"enter the value of n"<<endl;
    cin>>n;
    for(i=1;i<=n;i+=2)
    {
        p=1;
        fact=1;
        for(j=1;j<=i;j++)
        {
            p=p*x;
            fact=fact*j;
        }
        sign=-1*sign;
        sum+=sign*p/fact;
    }
    cout<<"sin"<<x<<"="<<sum<<endl;
    return 0;}
```

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \cdots$$

Storage classes

- Storage class-refers to the lifetime and visibility of a variable within the program.
- Life time-refers to the duration till which the variable remains to be active during program execution
- Visibility-refers to the scope of the variable i.e.in which program modules the variable is accessible.
- Four types of storage classes:
  - **Automatic**-default storage class of any type of variable; values are stored in primary memory.
    - Visibility- restricted to the function in which it is declared.
    - Lifetime-limited till its container function is executing
    - Keyword-auto (no need to declare data type)
    - Ex: auto x=3;
    - Auto b=3.5;
  - **External**- declared outside of a function but is accessible inside the function block.
    - Also called as global variable
    - Visibility- accessible by all functions present in the program.
    - Lifetime-same as lifetime of a program
    - Keyword-extern
    - Ex: extern int x;

- Four types of storage classes:
  - **Static** -
    - Visibility- of local variable
    - Lifetime-of an external variable; once declared inside a function block, it does not get destroyed after the function is executed, but retains its value so that it can be used by future function calls.
    - Keyword-static
    - Ex: static int count=0;
  - **Register** – similar to automatic variable; but values are stored in CPU registers that eventually increase the speed of execution of the program.
    - Also called as global variable
    - Visibility- accessible by all functions present in the program.
    - Lifetime-same as lifetime of a program
    - Keyword-Register
    - Ex: register char b='B';

```
//Program to calculate the variance and standard deviation of N numbers
#include <iostream>
#include<math.h>
#include <conio.h>
using namespace std;

int main()
{
int a[100],sum=0,sum1=0,e,N;
    int X;
float st;
   cout<<"enter the number of terms in sequence"<<endl;
   cin>>N;
    cout<<"enter the value to calculate the variance and std deviation of N-
numbers"<<endl;
    for(int i=1;i<=N;i++)
    {
        cin>>a[i];
    }

    for(int i=1;i<=N;i++)
    {
```

$$\text{Variance} = \frac{1}{N}\sum_{i=1}^{N}(x_1 - \bar{x})^2$$

$$\text{Standard Deviation} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2}$$

$$\text{where } \bar{x} = \frac{1}{N}\sum_{i=1}^{N}x_1$$

```
 sum=sum+a[i];
    }
      X=sum/N;

      for(int j=1;j<=N;j++)
      {
          sum1=sum1+((a[j]-X)^2);
      }

      e=sum1/N;
      st=sqrt(e);
      cout<<"mean:"<<X<<endl;
      cout<<"variance= "<<e<<endl;
      cout<<"std deviation= :"<<st<<endl;

      return 0;
}
```