# Day-4(3.3)

# C++ Language Constructs With Object and Classes

# Namespace

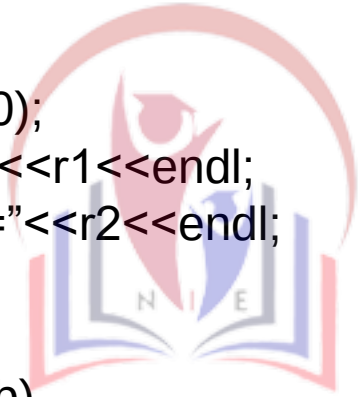
- A namespace is a declarative region that provides a scope to the identifiers(the names of types, function, variables etc) inside it.
- It is used to organize code into logical groups and to prevent name collisions that can occur espacially when our code base includes multiple libraries.
- Namespace provide a class like modularization without class like semantics.

# Function Overloading

- The mechanism of making the function to exhibit different behaviors in different call is known as function overloading.That is overloading means the use of same thing for different purposes. In c++ two or more than two function can be given the same name provided that each function has unique signature(no. of argument and their data type).The function would perform the different operations depending upon the argument list given in function call.The correct function call is determined by checking the number and type of argument.

## Example

```
#include<iostream>
Using namespace std;
int add(int ,int);
int add(int ,int ,int);
int main(){
int r1,r2;
r1 = add(10,30);
r2=add(20,30,40);
Cout<<”result=”<<r1<<endl;
Cout<<”result2=”<<r2<<endl;
Return 0;
}
Int add(int a,int b)
{
return (a+b);
}
Int add(int a,int b,int c){
return (a+b+c);
}
```



# Inline Function

- When a function is likely to be called many times each time the control is passed between calling and called function.Due to this passing of control between calling and called function the execution speed of the program is decreased. If passing of control during the repetitive call of the function is avoided, the program execution speed increases. C++ has a different solution to this problem .To eliminate the costs of call to small functions, c++ purposes a new feature called inline function. An inline function is a function that is expanded inline where it is invoked or called. That means compiler replaces the function call with corresponding function code. If the function code is very large in such case don't use inline function that reduces the program execution speed.

# Default Argument

- They may come in handy when some arguments are having the same values.

For eg. Bank intrest may remain the same for all the customer for a particular period of deposit.

```
int sum(int x, int y, int z=0,int w=0)
{
return(x+y+z+w);
}
int main()
{
cout<<sum(10,15);
cout<<sum(10,15,25);
cout<<sum(10,15,25,30);
return 0;
}
```



Output

25 50 80

Example

```
#include<iostream>
Using namespace std;
Inline int add(int a,int b)
{
return (a+b);
}
Inline float sub(float a ,float b){
return (a-b);
}
Int main(){
Cout<<”ths sum is “<<add(5,2);
Cout<<”the difference is”<<sub(5.0,2.0);
return 0;
}
```



Some situation where inline function may not be used

- If a function contains loop(for ,while ,do while)
- If the function contain static variable.
- If the function is recursive

# Pass/Return by reference

- reference variable is just alias(another name) for a variable and it is like a pointer with a few differences. It is declared using & operator .A reference variable Is a mechanism to define a second name for a variable that we can use to read or modify the original data stored in that variable.

```
#include<iostream>
using namespace std;
Int main(){
int i =10;
int &j =i;
cout<<”i=”<<i<<”j=”<<j<<endl;
j=20;
cout<<”i=”<<i<<”j=”<<j<<endl;
i =30 ;
cout<<”i=”<<i<<”j=”<<j<<endl;
return 0;
}
```

Output

I=10 j=10

I =20 j =20

I =30 j =30

# About Class and Object

- A class is a user defined data type of blue print of similar objects and it combines data and methods for manipulation the data. The data and functions within a class are called members of the class. when we define a class, we define a blueprint for a user defined data type.when we define a class we define a blueprint for a user defined data type. Structure of class:

```
Class class name
{
Private:
Variable declaration;
Public:
variable declaration
Function declaration;
}
```

# Object

- It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

1. **State:** It is represented by attributes of an object. It also reflects the properties of an object

2. **Behavior:** It is represented by methods of an object. It also reflects the response of an object with other objects.

3. **Identity:** It gives a unique name to an object and enables one object to interact with other objects.

# Access Specifier

- The access control specifier define the scope of the data member and member function defined inside the class.I.e from where they can be accessible.The body of the class contain three access control specifier i.e the body of the class can be grouped into three section private, protected and public.
- **Private**

Private indicated that the member can only be accessed from within the class.By default all the data members functions are private if we cannot use any control access specifier.

- **Protected**

Protected indicates that the member can onlybe accessed within the class and from within the child.

- **Public**

Public indicates that the data member can only be accessed from within the class.By default,the data member functions are private if we cannot use any control access specifier.

## Example

```
#include<iostream>
Class specifier{
Private:
Int a;
Protected :
Int b;
Void put(){
Cout<<” I m in protected section”;
}
Public:
Int c;
Void get(){
Cout<<” I m in public section”;
}
};
Int main(){
Specifier s;
s.a=5;//invalid
s.b=6;//invalid
s.c=8;//valid
s.get();//valid
return 0;}
```

# Member access

Member functions can be defined in two place:

- Outside the class definition.
- Inside the class definition.

- Member function Outside the class body

Member functions that are declared inside a class have to be defined separately outside the class.Their defination are very much like the normal functions.They should have a function header and a function body.

The general form of member function definition is:

return type class_name::functionname(argument declaration)

{
Function body
}

```
#include<iostream>
using namespace std;
class box{
private:
float length,breadth,height;
public:
void getdata();
float volume();
};
void box::getdata(){
cout<<"enter length breadth height";
cin>>length>>breadth>>height;
}
float box::volume(){
return (length *breadth *height);
}
int main(){
box b1;
float v;
b1.getdata();
v= b1.volume();
cout<<"the volume is :"<<v;
}
```

**Member function inside the class body**

Example

```
#include<iostream>
using namespace std;
class box{
private:
float length,breadth,height;
public:
void readdata(){
cout<<"enter length ,breadth,height";
cin>>length>>breadth>>height;
}
float volume(){
return(length*breadth*height);
}};
int main(){
box b1;
float v;
b1.readdata();
v = b1.volume();
cout<<"the volume is :"<<v;
return 0;}
```

# Constructor and Its Type

- Constructor is a member function that has the same name as the class itself. constructor being function can be overloaded.It is executed automatically whenever an object is created. constructor may or maynot take argument depending on how the object is to be constructed.There is no return value from the constructor.

## Example

```
class A
{
int a;
public:
A(){
a=0;
};
int main(){
A obj;
return 0;
}
```



## Types of constructor

- **Default constructor**
- **Parameterized constructor**
- **Copy constructor**

# About Destructor

Constructor declared which does not accept any argument is called default constructor. It is invoked when the object is created with no argument .

Syntax:

```
Class classname(){
Public:
Classname(){
}
};
```

**Example:**

```
#include<iostream>
using namespace std;
class integer
{
private:
int m,n;
public:
integer(){
m=0;
n=0;
}
void display(){
cout<<"M="<<m<<endl;
cout<<"N="<<n;
}};
int main(){
integer R;
R.display();
return 0;
}
```

# About DMA

- The process of allocating and freeing memory at runtime is known as dynamic memory allocation.This reserves the memory required by a program and returns this valuable resource to the system once the use of reserved space is utilized Though arrays can be used for data storage ,they are of fixed size .The programmer must know the size of the array or data in advance while writing the programs.A variable cant be used to define size of array,while declaring an array.In most situations ,it is not possible to know the size of the memory required until run time. With the use of dynamic memory ,we can request for a piece of memory at run time.In c++ we can use the operators new and delete to get and release computer memory at a run time.

## a) New and new[] operator

in order to request dynamic memory ,we use new operator .

syntax:

pointer variable = new datatype;

eg. Int *p;

p =new int;

it is used to allocate memory to contain one single element of particular type.If a sequence of more than one element is required,the syntax will be

pointer_variable =new datatype[number_of_element]

it is used to assign a block (an array) of elements of type dataype,where number of element is an integer value representing the amount of these.

Eg.

int *p;

P = new int [10];

In this case,the system dynamicllay assign space for ten elements of type int and returns a address of the first element of the sequence,which is assigned to pointer p.

# b) delete and delete[] operator

- when the reserved memory is no longer needed ,it should be freed so that memory becomes available again for other requests of dynamic memory.We can use delete operator to free allocated memory.

Syntax:

delete pointer_variable;

delete [] pointer variable;

## Example

```
#include<iostream>
using namespace std;
int main(){
int i,n;
int *p;
cout<<"enter value of n";
cin>>n;
p = new int[n];
if(p == 0){
cout<<"memory couldnot be allocated";
}
else{
for (i =0;i<n;i++){
cout<<"enter the number";
cin>>p[i];
}}
cout<<"the number enetered are"<<endl;
for( i =0;i<n;i++){
cout<<p[i]<<endl;
}
delete[]p;
return 0;
}
```

# About This Pointer

- Every object in C++ has access to its own address through an important pointer called **this** pointer. The **this** pointer is an implicit parameter to all member functions. Therefore, inside a member function, this may be used to refer to the invoking object.

```
#include<iostream>
using namespace std;
class Coordinate {
    private:
        int x;
        int y;
    public:
        Coordinate (int x, int y) {
            // Using this pointer inside the constructor
            // to set values in data members.
            this->x = x;
            this->y = y;
        }
        void printCoordinate() {
            cout<<"(x, y) = ("<<this->x<<", "<<this->y<<")"<<endl;
        }};
int main () {
    // Passing x and y coordinate in the constructor.
    Coordinate pointA(2, 3), pointB(5, 6);
    // Pointing the coordinates.
    pointA.printCoordinate();
    pointB.printCoordinate();
    return 0;
}
```

# Static Member and Static Function

- The data member in c++ can be qualified as static.The properties of static data member is a very similar to static variable in c.

Characteristic of static data

 It is initialized to zero when the first object of its class is created.NO other initialization is permitted.

 Only one copy of that member is created for the entire class and is shared by all the objects of tha class, no matter how many objects are created.

 It is visible only within the class but its lifetime is the entire program.

## Example

```
#include<iostream>
using namespace std;
class sample{
private:
static int count;
public: sample(){
count++;
}
static void showcount(){
cout<<"count="<<count<<endl;
}
};
int sample::count = 0;
int main(){
sample s1;
sample::showcount();
sample s2;
sample::showcount();
sample s3;
sample::showcount();
return 0;
}
```

# Constant Variable

If you make any variable as constant, using const keyword, you cannot change its value. Also, the constant variables must be initialized while they are declared.les in C++.

```
#include<iostream>
using namespace std;
int main()
{
const int i = 10;
cout<<i;
i++; //invalid
i=20; //invalid
}
```

Here int I is a constant variable.if we try to increase or assign the variable we will get error.

# Friend function

Non member function cannot have an access to the private data of the class.However,ther could be a situation where we would like two classes to share a particular function.IN c++ this is achived by using the concept of friend function.It permits the function or all other function of class to access a different class of private data member.The function that are declared with a keyword friend is called friend function.It can be defined anywhere in the program like normal function.The function definition does not use either keyword friend nor (::) scope resolution operator.

## Example

```
#include<iostream>
class sample
{
int a;
int b;
public:
void setvalue(){
a =25;
b=40;
}
friend float mean(sample s);
};
friend float mean(sample s)
{
return (s.a+s.b)/2;
}
int main(){
sample obj;
obj.setvalue();
cout<<”mean value=”<<mean(obj);
return 0;
}
```

# Friend Classes

- The member functions of a class can all be friends at the same time when you make the entire class a friend.

**Example**

```
#include<iostream>
using namespace std;
class alpha
{ private:
int data1;
public:
alpha(){data1=99;}
friend class beta;
};
class beta{
public:
void func1(alpha a)
{ cout<<"data1="<<a.data1<<endl;
}
void func2(alpha a)
{ cout<<"data1="<<a.data1;
}};
int main(){
alpha a;
beta b;
b.func1(a);
b.func2(a);
return 0;
}
```


Thank You !!!