OOAD-CLASSES AND OBJECTS
UNIT-3

P. Balamurugan, Assistant Professor
PG & Research Department of Computer Science
Government Arts College, Coimbatore -641018
Email: spbalamurugan@rediffmail.com

Topics to be Discussed….

1. Structures in C++
2. Classes in C++
3. Declaring Objects
4. Access Specifiers
5. Member Function
6. Inline Function
7. Data Hiding/Encapsulation
8. Static Members
9. Array of Objects
10. Objects as Function Arguments
11. Friend Function
12. Constructors and Destructors

Structures in C ++

- **Structures in C++** are user defined data types which are used to store group of items of non-similar data types.
- A structure creates a data type that can be used to group items of possibly different types into a single type.
- **Syntax:**

```
struct structureName{
        datatype-1 x1[,x2,…];
        [datatype-2 y1[,y2,…];
        ……
        ……
         datatype-n z1[,z2,…] ;]
};
```

Tag/Strucuture name

Members/Fields

Structures in C ++

Example :

```
struct Student  {
   char name[20];
    int id;
    int age;
}
```

Note : When the structure is declared, no memory is allocated. When the variable of a structure is created, then the memory is allocated.

How to create the instance of Structure?

Student s;

Here, s is a structure variable of type **Student**. When the structure variable is created, the memory will be allocated.

How to access the variable of Structure?

The variable of the structure can be accessed by simply using the instance of the structure followed by the dot (.) operator and then the field of the structure. **Example : s.id = 4;**

Structure with Constructor Method

Example :

```
struct Rectangle    {
   int width, height;
  Rectangle(int w, int h)
    {
        width = w;
        height = h;
    }
  void areaOfRectangle() {
    cout<<"Area of Rectangle is: "<<(width*height); }
 };
int main(void) {
    struct Rectangle rec=Rectangle(4,6);
    rec.areaOfRectangle();
   return 0;
}
```

C ++ Objects & Classes

- Object is a real world entity, for example, chair, car, pen, mobile, laptop etc.
- Object is an entity that has state and behavior. Here, state means data and behavior means functionality.
- Object is a runtime entity, it is created at runtime.
- Object is an instance of a class. All the members of the class can be accessed through object.
- **Example: Student** s1;
- Here **Student** is the type and s1 is the reference variable that refers to the instance of **Student** class.
- Class is a group of similar objects. It is a template from which objects are created. It can have fields, methods, constructors etc.

C ++ Objects & Classes

Example :

```
class Student  {
    public:
    int id;  //field or data member
    String name;//field or data member
 }
```

Example : Classes and Objects

```
class Student {
  public:
    int id;//data member (also instance variable)
    string name;//data member(also instance variable) };
int main() {
   Student s1; //creating an object of Student
   s1.id = 201;
   s1.name = “Kannan";
   cout<<s1.id<<endl;
   cout<<s1.name<<endl;
   return 0;
}
```

C ++ Objects & Classes

Example : Initialize and Display data through method

```
class Student {
  public:
     int id;//data member (also instance variable)
     string name;//data member(also instance variable)
     void insert(int i, string n) {
         id = i;
         name = n;
      }
     void display()  {
         cout<<id<<"  "<<name<<endl;
      }
};
int main(void) {
   Student s1,s2; //creating an objects of Student
   s1.insert(201, “Kannan"); s2.insert(202, “Krishnan");
   s1.display();  s2.display();
   return 0;
}
```

C ++ Constructors

- Constructor is a special method which is invoked automatically at the time of object creation.
- It is used to initialize the data members of new object generally.
- The constructor in C++ has the same name as class or structure.
- There can be two types of constructors in C++.
    - Default constructor
    - Parameterized constructor
- A constructor which has no argument is known as **default constructor**.
- A constructor which has parameters is called **parameterized constructor**.
- It is used to provide different values to distinct objects.

Default Constructors-Example

```
#include <iostream.h>
class Employee
 {
   public:
       Employee()
       {
          cout<<"Default Constructor Invoked"<<endl;
       }
};
int main(void)
{
    Employee e1; //creating an object of Employee
    Employee e2;
    return 0;
}
```

Parameterized Constructors-Example

```
#include <iostream>
class Employee {
 public:
 int id;//data member (also instance variable)
 string name;//data member(also instance variable)
 float salary;
 Employee(int i, string n, float s) {
 id = i; name = n; salary = s;
}
 void display() {
 cout<<id<<" "<<name<<" "<<salary<<endl;
} };
 int main(void) {
 Employee e1 =Employee(101, “Kumar", 890000);
 Employee e2=Employee(102, “Nagesh", 59000);
 e1.display();
 e2.display();
 return 0;
 }
```

C ++ Destructors

- A destructor works opposite to constructor; it destructs the objects of classes.
- It can be defined only once in a class.
- Like constructors, it is invoked automatically.
- A destructor is defined like constructor. It must have same name as class. But it is prefixed with a tilde sign (~).
- **Example :**

```
class Employee   {
  public:
     Employee()   {
        cout<<"Constructor Invoked"<<endl;
     }
     ~Employee()    {
        cout<<"Destructor Invoked"<<endl;
     }
};
int main(void)   {
   Employee e1; //creating an object of Employee
   Employee e2; //creating an object of Employee
   return 0;
}
```

C ++ this Pointer

- **this** is a keyword that refers to the current instance of the class.
- Uses of this keyword in C++.
  - to pass current object as a parameter to another method.
  - to refer current class instance variable.
  - to declare indexers.

C ++ this Pointer- Example

```
class Employee {
  public:
     int id; //data member (also instance variable)
     string name; //data member(also instance variable)
     Employee(int id, string name, float salary){
          this->id = id;    this->name = name;
       }
     void display()   {
         cout<<id<<"  "<<name<<endl;
      }
};
int main(void) {
   Employee e1 =Employee(101, “Kannan”); //creating an object of Employee
   Employee e2=Employee(102, “Kumar”); //creating an object of Employee
   e1.display();    e2.display();
   return 0;
}
```

C ++ Static Keyword

- **'static'** is a keyword or modifier that belongs to the type not instance.
- So, instance is not required to access the static members.
- static can be field, method, constructor, class, properties, operator and event.
- A field which is declared as static is called static field.
- Unlike instance field which gets memory each time whenever you create object.
- Only one copy of static field created in the memory.
- It is shared to all the objects.
- Memory efficient.

C ++ Static Keyword-Example

```
class Account {
  public:
     int accno; //data member (also instance variable)
     string name; //data member(also instance variable)
     static float rateOfInterest;
     Account(int accno, string name)  {
          this->accno = accno;
         this->name = name;
      }
     void display()    {
         cout<<accno<< "<<name<< " "<<rateOfInterest<<endl;
      }
};
float Account::rateOfInterest=6.5;
int main(void) {
   Account a1 =Account(201, “Kannan"); //creating an object of Employee
   Account a2=Account(202, “Kumar"); //creating an object of Employee
   a1.display();    a2.display();    return 0;
}
```

Structures Vs Classes

Structure in C++	Class in C++
1. Value data type that can hold related data.	1. Blueprint that defines data and methods create objects.
2. Does not support inheritance	2. Support inheritance.
3. An instance is a structure variable.	3. An instance is an object.
4. A keyword to define structure is 'struct'.	4. A keyword to define class is 'class'.
5. There is no access specifiers, then default is public.	5. There is no access specifiers, then default is private.
6. Its objects are created in stack memory.	6. Its objects are created in heap memory
7. Member variables can not be initialized directly.	7. Member variables can be initialized directly.
8. It can have only parameterized constructor.	8. It can have all types of constructor and destructor.

Access Specifiers

- In C++, there are three access specifiers:
  - public - members are accessible from outside the class
  - private - members cannot be accessed (or viewed) from outside the class (Default)
  - protected - members cannot be accessed from outside the class, however, they can be accessed in inherited classes.
- Example:

```
class myClass {
    public:    // Public access specifier
      int x;   // Public attribute
    private:   // Private access specifier
      int y;   // Private attribute
};
int main() {
  myClass myObj;
  myObj.x = 25;  // Allowed (public)
  myObj.y = 50;  // Not allowed (private)
  return 0;
}
```

Access Specifiers(Contd..)

	Within the Class	Derived Class	Outside the Class
Public	**YES**	**YES**	**YES**
Protected	**YES**	**YES**	**NO**
Private	**YES**	**NO**	**NO**

Access Specifiers-Example

```
class base
    {
        private:
        int a;
        public:
        int b;
        protected:
        int c;
        public :
        void show()
        {
        a=10;b=20;c=30;
        cout<<"a="<<a<< "\n"<<"b="<<b<< "\n"<<"c="<<c<<"\n";
        }
    };
```

Access Specifiers-Example(Contd…)

```
class derived : public base
    {
        public:
         void show()
        {
        b=25;c=35;
        cout<<"b="<<b<< "\n"<<"c="<<c<<"\n";
        }
    };
void main()
    {
       clrscr();
       base b; b.show();
       derived d; d.show();
       cout<<"b="<<b.b;
    }
```

Inline Function

- It is a powerful concept in C++ programming language.
- If a function is inline, the compiler places a copy of the code of that function at each point where the function is called at compile time.
- To make any function inline function just preceded that function ith **inline** keyword.

Why use Inline function?

- Whenever we call any function many time then, it take a lot of extra time in execution of series of instructions such as saving the register, pushing arguments, returning to calling function.
- The main advantage of inline function is it make the program faster.

Syntax:

```
inline function-name()
{
//function body
}
```

Encapsulation

- The meaning of **Encapsulation**, is to make sure that "sensitive" data is hidden from users.

Why Encapsulation?

- It is considered good practice to declare your class attributes as private (as often as you can).
- Encapsulation ensures better control of your data, because you (or others) can change one part of the code without affecting other parts
- Increased security of data.

Static Members of Class

- A static member is shared by all objects of the class.
- All static data is initialized to zero when the first object is created, if no other initialization is present.
- We can't put it in the class definition but it can be initialized outside the class by re-declaring the static variable, using the scope resolution operator (::) to identify which class it belongs to.

Example:

```
class box {
        public:
            static int count;
           box()
             {
               count++;
             }};
```

Static Members of Class

```
int box :: count=0;
int main(void)
{
  box b1;
  box b2;
  cout<<"Total objects created="<<box::count<<endl;
  return  0;
}
```

- Similarly, A static member function can be called even if no objects of the class exist.
- the **static** functions are accessed using only the class name and the scope resolution operator (::).
- Static member functions have a class scope and they do not have access to the **this** pointer of the class.

Static Members of Class

Exercise-1: Display object count using static member function.

Array of Objects

- An array which contains the class type of element is called array of objects.
- It is declared and defined in the same way as any other type of array.

Example :

```
class stock
{
    int itemno;
    char itemname[10];  // array as member variable
    float price;
    public:
    void getdata() {
    cin>>itemno>>price;
    gets(itemname);
     }
     void putdata() {
     cout<<'\n'<<itemno<<'\t'<<itemname<<'\n'<<price;
     }
};
```

Array of Objects

```
int main()
{
stock s[10];
int i;
cout<<”Enter the details\n”;
for( i=0;i<10;i++)
s[i].getdata();
cout<<”\n Item Details\n”;
cout<<”\nITEM NO \t ITEM NAME\t PRICE”<<endl;
for( i=0;i<10;i++)
 s[i].putdata();
 return 0;
}
```

Objects as Function Arguments

- Passing and Returning Objects in C++ functions

  - Pass class's objects as arguments and also return them from a function the same way we pass and return other variables.

- Passing an Object as argument

  - To pass an object as an argument, write the object name as the argument while calling the function.

Syntax :

```
function_name(object_name);
```

Example :

```
class Example {
public:
    int a;
    void add(Example E)
    {
        a = a + E.a;
    }
};
```

Passing an Object as argument

```
int main()
{
    // Create objects
   Example E1, E2;
    // Values are initialized for both objects
   E1.a = 50;
   E2.a = 100;
    cout << "Initial Values \n";
    cout << "Value of object 1: " << E1.a << "\n object 2: " << E2.a << "\n\n";

   // Passing object as an argument to function add()
   E2.add(E1);
    // Changed values after passing object as argument
   cout << "New values \n";
   cout << "Value of object 1: "<< E1.a<< "\n object 2: " << E2.a << "\n\n";
    return 0;
}
```

Retur an Object as argument

Syntax :

object = return object_name;

Example :

```
class Example {
public:
    int a;
    Example add(Example Ea, Example Eb)
    {
        Example Ec;
        Ec.a = 50;
        Ec.a = Ec.a + Ea.a + Eb.a;
       // returning the object
        return Ec;
    }
};
```

Return an Object as argument

```
int main()
{
    Example E1, E2, E3;
    E1.a = 50;
    E2.a = 100;
    E3.a = 0;
    cout << "Initial Values \n";
    cout << "Object-1: " <<E1.a<< ",\nObject-2: "<<E2.a
         <<",\nObject-3:"<<E3.a<<"\n";
     // Passing object as an argumentto function add()
    E3 = E3.add(E1, E2);
     // Changed values after passing object as an argument
    cout << "New values \n";
    cout << "Object-1: " <<E1.a<< ",\nObject-2: "<<E2.a
         <<",\nObject-3:"<<E3.a<<"\n";
     return 0;
}
```

Friend Functions

- Protected and private data of a class can be accessed using the function.
- By using the keyword friend compiler knows the given function is a friend function.
- For accessing the data, the declaration of a friend function should be done inside the body of a class starting with the keyword friend.
- **Syntax :**

```
class class_name
{
    friend data_type function_name(argument/s);
};
```

- The function definition does not use either the keyword **friend or scope resolution operator(::)**.
- **Characteristics of a Friend function:**
  - The function is not in the scope of the class to which it has been declared as a friend.
  - It can not be called using the object as it is not in the scope of that class.
  - It can be invoked like a normal function without using the object.
  - It cannot access the member names directly and has to use an object name and dot membership operator with the member name.
  - It can be declared either in the private or the public part.

Friend Functions

```
class Box
{
    private:
        int length;
    public:
        Box() {length=0; }
        friend int printLength(Box); //friend function
};
int printLength(Box b)
{
   b.length += 10;
    return b.length;
}
int main()
{
    Box b;
    cout<<"Length of box: "<< printLength(b)<<endl;
    return 0;
}
```

Friend Classes

- Friends should be used only for limited purpose.
- Too many functions or external classes are declared as friends of a class with protected or private data, it lessens the value of encapsulation of separate classes in object-oriented programming.
- Friendship is not mutual. If class A is a friend of B, then B doesn't become a friend of A automatically.
- Friendship is not inherited.
- **Example :**

```
class A {
private:
    int a;
public:
    A() { a = 0; }
    friend class B; // Friend Class
};
```

Friend Classes

```
class B {
private:
    int b;
public:
    void showA(A& x)
    {
        // Since B is friend of A, it can access private members of A
        cout<<"A::a=" << x.a;
    }
};
int main()
{
    A a;
    B b;
    b.showA(a);
    return 0;
}
```

References

- https://www.tutorialspoint.com/object_oriented_analysis_design/ooad_object_oriented_model.htm
- https://www.geeksforgeeks.org/object-oriented-programming-in-cpp
- https://www.javatpoint.com/cpp-oops-concepts

TEXT BOOKS

1) Grady Booch, "Object Oriented Analysis and Design with Applications", Second Edition, Pearson Education.
2) Ashok N. Kamthane, "Object Oriented Programming with ANSI &Turbo C++" , First Indian Print, Pearson Education, 2003.

REFERENCE BOOKS

1) Balagurusamy "Object Oriented Programming with C++", TMCH, Second Edition, 2003.