Message, Instance and Initialization

Chapter 3

Message passing

- A request for an object to perform its operation is called message
- A message for an object is a request for execution of a procedure and therefore will invoke a function in the object that generates the desired results
- Message passing involves specifying
    - name of the object
    - name of the function
    - information to be sent

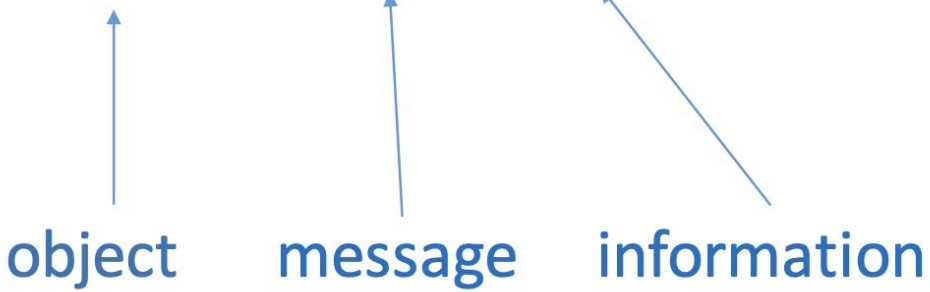
Message Passing

Syntax for message passing:

objectName . functionName (attributes/ information);

Example:

objectA . setData(20,30);



Initialization in C++:

```
class Test
        {
                int x,y;
                public:
                void setData()
                {
                        cout<<"Enter values of x and y";
                        cin>>x>>y;
                }
        };
void main()
{
        Test objOne;
        objOne.setData(); /*objOne.setData() invokes the member function setData() which assigns initial value of private data members x and y of objOne. */
}
```

Initialization in C++:

```
class Test
        {
                int x,y;
                public:
                void setData(int a,int b)
                {
                        x=a;
                        y=b;
                }
        };
void main()
{
        Test objOne;
        objOne.setData(10,20); /*objOne.setData() invokes the member function setData(int,int) and passes values to assign initial value of private data members x and y of objOne. */
}
```

Constructor

- A constructor is a member function of a class which initializes object of the class
- Constructor is automatically called when object of the class is created
- Constructor are useful for initializing the values of data of object

Constructor has following characteristics:

1. Constructor name should be same as its class name
2. Constructor does not have any return type not even void
3. It is called automatically when an object is created
4. It should be declared in public section
5. It cannot be static

Constructor

Syntax for declaration of constructor:

```
class ClassName {
        ...// data members
        public:
        ClassName();        //constructor
        …
        //other member functions };
```

Example:

```
class Construct {
        int x,y;
        public:
        Construct() {
                x=10; y=20;
                } };
void main() {
Construct obj1 ; …
}
```

Types of Constructor:

1. Default Constructor

2. Parameterized Constructor

3. Copy Constructor

1. Default Constructor:

- A constructor without any argument is called default constructor
- It has no parameters
- If we do not specify a constructor explicitly, the compiler generates a default constructor implicitly

```
#include<iostream>
Using namespace std;
class Construct
{
int x,y;
public:
      Construct() {
            x=0;
            y=0;
            }
      void display()
      {
      cout<<“Value of x=“<<x<<endl<<“Value of y
=“<<y;
      }
};
int main()
{
      Construct obj;
      obj.display();
      return 0;
 }
```

2. Parameterized Constructor:

1. Constructor with arguments is parameterized constructor

2. Here, we pass arguments while defining object of the class which are assigned to data members of the class

3. It is used to initialize data members of different objects with different initial values when they are created

```
#include<iostream>
using namespace std;
class Construct
{
        int x,y;
        public:
        Construct() //default constructor
        {
                x=0;
                y=0;
        }
        Construct(int a,int b) //parameterized constructor
        {
                x=a;
                y=b;
        }
        void display()
        {
                cout<<“Value of x=“<<x<<endl;
                cout<<“Value of y =“<<y<<endl;
        }
};
int main() {
        Construct obj1(100,200), obj2(500,600);
        obj1.display();
        obj2.display();

        }
```

Practice

Q1. Write a program to define a class Rational. Use constructor to initialize the values of data members of object (i.e num and denum) and display the rational number. Use appropriate member functions as required.

Constructor with default arguments

- Parameterized constructor with default values in arguments

Example:

```
class Construct {
        int a,b,c;
        public:
        Construct();
        Construct(int x,int y=20, int z=30);
        void display();
        };
```

```
Construct:: Construct()
{
        a=0;
        b=0;
        c=0;
}
Construct::Construct(int x,int y, intz)
{
        a=x;
        b=y;
        c=z;
}
void Construct:: display()
{
        cout<<endl<<“Value of a=“<<a;
        cout<<endl<<“Value of b=“<<b;
        cout<<endl<<“Value of c=“<<c;
}
```

```
int main() {
        Construct obj1(1,2,3);
        Construct obj2(120,89);
        obj1.display();
        obj2.display();
         }
```

Practice:

Q1. Define a class Bank. Calculate simple interest using default value of rate=1.5%. Use constructor to initialize the object and appropriate member functions to display and calculate the result.

Q2. Define a class Complex. Write a program to add two complex numbers. Use constructor to initialize the objects.

Q3. . Define a class Complex. Write a program to add two complex numbers using friend function. Use constructor to initialize the objects.

3. Copy Constructor:

• Copy constructor is a constructor which creates an object as a copy of existing object of same class

• Copy constructor is used to create an object by initializing it with an object of the same class, which has been already created

• It uses reference to an object of same type (class) as argument

Copy Constructor:

Syntax for declaration of copy constructor:

```
class Construct {
    …
    public:
    …
    Construct(Construct &); //copy constructor
    };
```

```
#include<iostream>
class Construct {
        int a,b,c;
        char name[20];
        public:
        Construct(); //default constructor
        Construct(int x,int y, int z); //parameterized
        constructor
        Construct(Construct &); //Copy constructor
        void display();

 };
Construct:: Construct() {
        a=0; b=0; c=0;
        }
```

```
Construct::Construct(int x,inty, intz)
{
a=x;
b=y;
c=z;
}
Construct::Construct(Construct &obj)
{
a=obj.a;
b=obj.b;
c=obj.c;
}
void Construct:: display()
{
cout<<endl<<"Value of a="<<a;
cout<<endl<<"Value of b="<<b;
cout<<endl<<"Value of c="<<c;
}
```

```
int main()
{
Construct obj1(10,20,30); //parameterized constructor
called
Construct obj2; //Default constructor called
cout<<"Object 1 :"<<endl;
obj1.display();
cout<<endl<<endl<<endl;
cout<<"Object 2: "<<endl;
obj2.display();
cout<<endl<<endl<<endl;
cout<<"Object 3 : "<<endl;
Construct obj3(obj1); //copy Constructor called
obj3.display();
}
```

Can Constructor be Overloaded?

Yes, Constructor can be overloaded in same way as functions can be overloaded

A class can have more than one constructor where each constructor take different set of parameters

Constructor can be overloaded as class can have more than one constructor having same name but different signature

When we define an object of a class, correct version of the constructor is called on the basis of passed arguments

Practice:

- Q1. Create a class called Mountain with data members name, height and location. Use constructor that initialize the members to the values passed to it as parameters. A function cmpHeight() to compare height of two objects and function displayInfo() to display information of mountain. In main create two objects of the class mountain and display the information of mountain with greatest height.

Destructor

- A special member function of the class that is called when an object of the class is destroyed
- A constructor initializes the object, whereas a destructor destroys the object
- Destructor de allocates the memory allocated by object
- It is invoked automatically when the object goes out of the scope

Characteristics of destructor

1. Destructor has same name as class and is preceded by a tilde(~)
2. Destructor doesn’t have any argument
3. Destructor doesn’t have any return type, not even void
4. It is called automatically whenever an instance of the class to which it belongs , goes out of scope
5. It is defined in public section of class

Syntax of Destructor

Syntax:

```
class className


{
public:
~className () {
        destructor code
... };
```

```
#include<iostream>
using namespace std;
class ABC{
public:
        ABC(){
                cout<<"This is a constructor"<<endl; }
        ~ABC(){
                cout<<"This is a destructor"<<endl; }
};
int main() {
    ABC obj1, obj2; }
```

```
#include<iostream>
using namespace std;
class ABC
{
    int id;
    public:
    ABC(){
        id=0;
    }
    ABC(int a)
{
        id=a;
cout<<endl<<"Constructor called for object : "<<id<<endl;
}

 ~ABC() {
    cout<<endl<<"Destructor called for object :"<<id<<endl;

 }

};
```

```
int main(){
    ABC obj1(1),obj2(2);

cout<<endl<<endl<<"***************Block
2***************"<<endl;
ABC obj3(3),obj4(4);
cout<<endl<<endl<<endl<<endl;
cout<<endl<<endl<<"**********End of block
2**********"<<endl;

}
```

Can destructor be overloaded?

- Destructor cannot be overloaded
- There can be only one destructor in a class
- Destructor doesn’t take any argument nor has a return type
- So multiple destructors with different signatures are not possible in a class
- Hence, destructor overloading is not possible

Memory Allocation

- Static Memory Allocation
- Dynamic Memory Allocation

Static Memory Allocation: In static memory allocation, memory is allocated in stack and the memory allocated is automatically freed or de allocated by the compiler itself when the scope of the allocated memory finishes.

Example:

```
void main()
{
        int num;          // variable
        int arrayEg[5];   // array
                          ...
}
```

```
class Student
{
        ...
};
void main()
{
        Student obj;      // object
        Student objArray[10]; //array of object
        ...
}
```

Dynamic Memory Allocation

- In dynamic memory allocation, the memory is allocated in heap
- While allocation of memory on heap, we need to delete the memory manually as memory is not de allocated or freed by the compiler itself even if the scope of allocated memory finishes
- We allocate and deallocate memory dynamically using new and delete operator respectively.

new Operator

- The new operator is used for allocating memory dynamically
- new operator requests for the memory allocation in heap

Syntax to allocate memory using new operator:

pointer_variable = new datatype ;

Example: int *ptr = new int;

new Operator

- We can also initialize the memory using new operator.

Syntax:

pointer_variable = new datatype (value);

Example: int *ptr = new int (50);

Delete operator

• delete operator is used to de allocate the memory allocated by new operator
Syntax: delete pointer_variable;

Syntax: To free dynamically allocated array pointed by pointer variable

delete [] pointer_variable:

Create a class named Student with data members name, roll and member function get and display output

```
#include<iostream>
#include<cstring>
using namespace std;
class Student{
    string name;
    int roll;
    public:
    void get(){
        cout<<"Enter details:";
        cin>>name>>roll;
    }
    void display(){
        cout<<"Name: "<<name<<endl<<"Roll:
"<<roll<<endl;
    }
};
```

```
int main()
{
    Student *ptr;
    ptr = new Student;

    ptr->get();
    ptr->display();

    delete ptr;
    return 0;
}
```

```
#include<iostream>
#include<cstring>
using namespace std;
class Student{
    string name;
    int roll;
    public:
    void get(){
        cout<<"Enter details:";
        cin>>name>>roll;
    }
    void display(){
        cout<<"Name: "<<name<<endl<<"Roll:
"<<roll<<endl;
    }
};
int main()
{
    int n;
    Student *ptr;
    cout<<"Enter no. of students:"<<endl;
    cin>>n;
    ptr = new Student[n];

    for(int i=0;i<n;i++)
    {
        (ptr+i)->get();
    }
    cout<<endl;
    cout<<"Display information"<<endl;
    for(int i=0;i<n;i++)
    {
        (ptr+i)->display();
    }
    delete[] ptr;
    return 0;
}
```

Problem:

Create a class named test in which there are two data members a,b with member functions display(). Use constructor for initializing the value and create two objects dynamically.

```
#include<iostream>
using namespace std;
class Test{
    int a;
    int b;
    public:
    Test(){
        a=0;
        b=0;
    }
    Test(int x, int y){
        a=x;
        b=y;
    }
    ~Test(){
        cout<<"Calling destructor"<<endl;
    }
    void display(){
        cout<<"Value of a: "<<a<<endl<<"Value of b:
"<<b<<endl;
    }
};
```

```
int main()
{
    Test *ptr1 = new Test();
    Test *ptr2 = new Test(5,6);

    ptr1->display();
    ptr2->display();


    delete ptr1;
    delete ptr2;

    return 0;
}
```

Class Question Solution

```
#include <iostream>

using namespace std;
class Theory;
class Practical;
class Marks{
    int tot_marks;
    public:
    void add(Theory t, Practical p);
};
class Theory{
    int mark;
    public:
    Theory(){
        mark=5;
    }
    friend void Marks:: add(Theory t, Practical p);
};
class Practical{
    int marks;
    public:
    Practical(){
        marks=10;
    }
    friend void Marks::add(Theory t, Practical p);

};
void Marks::add(Theory t, Practical p){

        tot_marks=t.mark+p.marks;
        cout<<"Displaying total:"<<t.mark+p.marks;
    }
int main()
{
    Practical p;
    Theory t;
    Marks m;
    m.add(t,p);

    return 0;
}
```

Static Data member and function

In C++ static member function and member variables are declared by using keyword static.

The main rule of static variable is to maintain values common to entire class.

Following are some important features of static variable and function.

1. Static data members are initialized to zero when the object of that class is created.

2. Only one copy of static data member created and shared by all class.

3. It is only visible within class.

4. Static function can have access to only other static member of same class.

5. Static member function can be called using class name.

As mentioned above, a method may be declared as static, meaning that it acts at the class level. Therefore, a static method cannot refer to a specific instance of the class (i.e. it cannot refer to this, Me, etc.), unless such references are made through a parameter referencing an instance of the class, although in such cases they must be accessed through the parameter's identifier instead of this.

```
#include<iostream>
using namespace std;
class tryy{
int id;
static int tot;//static data member
public :
tryy(){
tot++;
id=tot;
}
void print(){
cout<<"ID is:"<<id<<endl; }
static void printcount(){
cout<<"\n number of instance are "<<tot; }
 };
int tryy::tot=0;
int main(){
tryy t,tt;
t.print();
tt.print();
tryy::printcount();
}
```

```
#include <iostream>
using namespace std;
class base {
   public:
   virtual void print(){
      cout << "print base class" << endl;
   }
   void show(){
      cout << "show base class" << endl;
   }
};
class derived : public base {
   public:
   void print(){
      cout << "print derived class" << endl;
   }
   void show(){
      cout << "show derived class" << endl;
   }
};
```

```
int main(){
   base* bptr;
   derived d;
   bptr = &d;
   //calling virtual function
   bptr->print();
   //calling non-virtual function
   bptr->show();
}
```

Virtual destructors

```
#include<iostream>
using namespace std;
class base {
   public:
      base() {
         cout<<"Constructing base \n";
      }
       ~base() {
         cout<<"Destructing base \n";
      }
};
class derived: public base {
   public:
      derived() {
         cout<<"Constructing derived \n";
      }
      ~derived() {
         cout<<"Destructing derived \n";
      }
};
int main() {
   derived *d = new derived();
   base *bptr = d;
   delete bptr;
   return 0;
}
```