girmiti software™

Introduction to C++

- Array of class objects
- Pointer to a class
- Access Modifiers
    - private
    - public
    - protected
- Constructors
    - Default constructor
    - Parameterized constructor
    - Copy constructor

Classes & Objects

Array of class objects:
Like array of other user-defined data types, an array of type class can also be created. The array of type class contains the objects of the class as its individual elements. Thus, an array of a class type is also known as an array of objects. An array of objects is declared in the same way as an array of any built-in data type.

The syntax for declaring an array of objects is:
class_name array_name [size] ;

Example:

```
#include<iostream>
#include<conio.h>
using namespace std;
class Student
{
 int rollno;
 char name[20];
 public:
 void setStudent();
 void getStudent();
 };
```

Classes & Objects

```
void Student::setStudent()
{
cout<<"enter rollno";
cin>>rollno;
cout<<"enter name";
cin>>name;
}
void Student::getStudent()
{
 cout<<"rollno is "<<rollno<<endl;
 cout<<"name is "<<name<<endl;
 }
 int main()
 {
 Student s[3];      //Array of objects
 for(int i=0;i<=2;i++)
 {
 s[i].setStudent();
 }
 for(int i=0;i<=2;i++)
 {
 s[i].getStudent();
 }
 getch();
 }
```

Pointer & Classes

Pointer & Classes:

A pointer to a C++ class is done exactly the same way as a pointer to a structure. To access members using pointer to a class we use the member access operator **->** operator, just as we do with pointers to structures. Also as with all pointers, we must initialize the pointer before using it.

Syntax to access members of a class using pointer to an object variable:

```
ptr->data_member;
ptr->member_fuction();
```

Example:

```
#include<iostream>
#include<conio.h>
using namespace std;
class Student
{
int rollno;
char name[20];
public:
void setStudent();
void getStudent();
};
```

Pointer & Classes

```
void Student::setStudent()
{
cout<<"enter rollno";
cin>>rollno;
cout<<"enter name";
cin>>name;
}
void Student::getStudent()
{
cout<<"rollno is "<<rollno<<endl;
cout<<"name is "<<name<<endl;
}
int main()
{
Student s1;
Student *s;     //Pointer to an object variable
s= &s1;
s->setStudent();
s->getStudent();
getch();
}
```

Pointer & Classes

Passing class pointer as an argument:

```
#include<iostream>
#include<conio.h>
using namespace std;
class A
{
        int x,y;
        public:
        void set()
        {
                cout<<"enter x=";
                cin>>x;
                cout<<"enter y=";
                cin>>y;
         }
         void sum(A *p)
         {
                p->x=p->x+10;
                p->y=p->y+10;
         }
         void display()
         {
                cout<<"x="<<x<<endl;
                cout<<"y="<<y<<endl;
         }
};
```

Pointer & Classes

```
int main()
 {
          A a1;
          A *ptr;
          a1.set();
          ptr=&a1;
          a1.sum(ptr);
          a1.display();
 }
```

Reference variable & Classes

Example of reference as an argument:

```
#include<iostream>
 #include<conio.h>
 using namespace std;
 class A
 {
            int x,y;
            public:
            void set()
            {
                        cout<<"enter x=";
                        cin>>x;
                        cout<<"enter y=";
                        cin>>y;
             }
             void sum(A &p)
             {
                        p.x=p.x+10;
                        p.y=p.y+10;
             }
             void display()
             {
                        cout<<"x="<<x<<endl;
                        cout<<"y="<<y<<endl;
             }
 };
```

Reference Variables & Classes

```
int main()
 {
          A a1;
          a1.set();
          a1.sum(a1);
          a1.display();
 }
```

Access Modifiers

The access modifiers in C++ specifies accessibility (scope) of a data member, member functions, constructor or class.

C++ offers possibility to control access to class members and functions by using access specifiers.

Access specifiers are used to protect data from misuse.

Types of Access Modifiers:

- private
- protected
- public

The default access for members is private.

Access Modifiers

1. public access modifier:

public class members and functions can be used from outside of a class by any function or other classes. We can access public data members or function directly by using dot operator (.)

Example:

```
#include <iostream>
using namespace std;
class A
{
public:
int length;
void display( );
};
// Member functions definitions
void A::display()
{
cout<<“hello”;
}
```

Access Modifiers

```
int main( )
{
A a1;
a1.length = 10;     // OK: because length is public
cout << "Length of line : " <<a1.length <<endl;
a1.display();        //OK: because display() is public
return 0;
}
```

Access Modifiers

2. private access modifier:
A private member variable or function cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members.
By default all the members of a class are of private type.

Example:

```
#include <iostream>
using namespace std;
class A
{
int length;
public:
void display( );
};
// Member functions definitions
void A::display()
{
cout<<“hello”;
}
```

Access Modifiers

```
int main( )
{
A a1;
a1.length = 10;    // error: because length is private by default
a1.display();       //Ok: because display is public now.
return 0;
}
```

Access Modifiers

3. protected access specifier:

A protected member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes.

Example:

```
#include<iostream>
using namespace std;
class A
{
protected:
int length;
public:
void display( );
};
// Member functions definitions
void A::display()
{
cout<<“hello”;
}
```

Access Modifiers

```
int main( )
{
A a1;
a1.length = 10;    // error: because length is protected
a1.display();      //Ok: because display is public now.
return 0;
}
```

Constructor

Constructor
Constructor is a special type of method that is used to initialize the object.
Whenever an object is created, the special member function i.e. constructor will be executed automatically. It constructs the values i.e. provides data for the object that is why it is known as constructor.

Rules for defining the constructor.

- Constructor name must be same as its class name
- Constructor must have no explicit return type

Types of constructors

1. Default constructor (no-arg constructor)
2. Parameterized constructor
3. Copy constructor

Constructor

Default Constructor:

A constructor that have no parameter is known as default constructor.

Syntax of default constructor:
<class_name>(){}

- If there is no constructor in a class, compiler automatically creates a default constructor.

Example of default constructor:

```
#include<iostream>
using namespace std;
class A
{
public:
A()          //default constructor
{
cout<<"hello";
}
};
int main()
{
A a1;
}
```

Constructor

Parameterized constructor:
A constructor that have parameters is known as parameterized constructor.
Parameterized constructor is used to provide different values to the distinct objects.

Syntax of Parameterized constructor:
<class_name>(arg1,arg2,...,arg n){}

Example:

```
#include<iostream>
using namespace std;
class Student
{
        public:
        int rollno;
        string name;
        Student(int x,string y);      //Parameterized constructor
        void display();
};
```

Constructor

```
Student::Student(int x,string y)
        {
                rollno=x;
                name=y;
        }
        void Student::display()
        {
                cout<<"rollno no. is "<<rollno<<endl;
                cout<<"name is "<<name<<endl;
        }
int main()
{
        Student s1(1,"Aryan");
        Student s2(2,"Vihan");
        s1.display();
        s2.display();
}
```

Constructor

Copy constructor:
Copy constructor are always used when the compiler has to create a temporary object of a class object.
The copy constructors are always used in the following situations:

- The initialization of an object by another object of the same class.
- Return of objects as function value.
- Stating the object as by value parameters of a function.

The copy constructor can accept a single argument of reference to same class type.The purpose of the copy constructor is copy objects of the class type.

Syntax of the copy constructor is:

```
class_name::class_name(class_name &ptr)
```

The symbolic representation of the above format is:

```
X::X(X &ptr)
```

Constructor

Example:

```
#include<iostream>
using namespace std;
class Point
{
   private:
   int x, y;
   public:
   Point(int x1, int y1)
   {
        x = x1;
        y = y1;
   }
   // Copy constructor
   Point(Point &p2)
   {
         x = p2.x;
         y = p2.y;
    }
```

Constructor

```
int getX()
{
      return x;
}
int getY()
{
       return y;
 }
 };
 int main()
 {
      Point p1(10, 15);
      Point p2 = p1;    // Copy constructor is called here
      cout << "p1.x = " << p1.getX() << ", p1.y = " << p1.getY();
      cout << "\np2.x = " << p2.getX() << ", p2.y = " << p2.getY();
 }
```

Discussions