C++ Notes on Unit-IV

Array & Pointers, Operator Overloading & Inheritance

By

Prof. P. J. Mokashi

Unit :- 4

Array & pointers

Introduction

An array is a group related data item which share a common name.

The application in array is similar to that in C - programing. The only different is the way where character array are initialise, in c the complier will allows us to declare the array size as the exact length of a string constant for Example.

```
char c[3] = "sky";
```

is valid in c programing it assume that the programmer can not placed the null character '\0' in the defination. But in c++ the size should be one leger than the number of character in the string for Example :-

```
char c[4] = "sky";
```

is valid in c++

* Array as a class member *

An array can be used as a member variable in a class the array variable can be declare as a private member of the class array which can be used in the member function, likeany other array variables. we can perform any operator on it. similarly, we may used other member function to perform any other operation on the array values.

Example :-

```
const int size = 10;
class array
{
    int a[size];
public:
    void getdata();
    void display();
};
```

* Array as Object *

Like a member of class an array can be used as an object ae can used array of the variable that are of type class. such variable are called as arrays of object.

For Example :-

```
class employee
{
    char name[30];
    float age;
public:
    void getdata();
    void putdata();
};
```

in the above example employee is the name of the class and can be used to create objects that relate to different catorigire of the employee

employee manager[3];

the array manager contains mainly three object namely manager[0], manager[1],

manager[2] of employee class.
since an array of object behaves like any other array, we can used as usual array accessing method to access indiual element and the dot operator access the member function. Note that only the space for the data items of the objects is created.
following program illustrase the use of object array.

```
// Pgm for Array Objects
#include<iostream.h>
#include<conio.h>
class Employee
{
private: char name[30];
          float age;
public: void getdata();
         void display();
};
void Employee::getdata()
{
cout<<"\n Enter name:";
cin>>name;
cout<<"\n Enter age:";
cin>>age;
}
void Employee::display()
{
cout<<"\n Name:"<<name;
cout<<"\n Age:"<<age;
}
void main()
{
clrscr();
Employee e[3];
cout<< “\n\t\t\t ***** OUTPUT *****”;
cout<<"\n Enter Details of Three Employees:";
for(int i=0;i<=2;i++)
{
e[i].getdata();
}
cout<<"\n Details of Three Employee's are:";
for(i=0;i<=2;i++)
{
e[i].display();
}
getch();
}
```

******* OUTPUT *******

```
Enter Details of Three Employees:
Enter name:Shrikant
Enter age:35
```

Enter name:Amol
Enter age:38
Enter name:Prasad
Enter age:40

Details of Three Employee's are:
Name:Shrikant
Age:35
Name:Amol
Age:38
Name:Prasad
Age:40

Pointers to Objects :-

Pointer :- A pointer is an variable which contains an address which is the location of another variable in the memory.
Pointers are also used to access the class members. Basically, a pointer can point to an object created by a class.
Consider the following statement;

```
Item x;
```

where, Item is a class and x is an object defined for class Item. Similarly we can define a pointer x_ptr of class Item as;

```
Item *x_ptr;
```

Object pointers are useful in creating objects at run time. We can also use an object pointer to access the public members of an object.
Consider a class Item defined as follows.

```
class Item
{
private:  int code;
          float price;
public:   void getdata(int a, float b)
          {
              code = a;
              price = b;
          }
```

```
    void display()
    {
    cout<<" Code = "<< Code;
    cout<<" Price = "<< price;
    }
};
```

Let us declare object of Item class as x and a pointer ptr to x as follows:

```
Item x;
Item *ptr = &x;
```

The pointer ptr is initialized with the address x.

We can refer the member functions of Item class in two ways, one by using dot operator and the object and another by using the arrow operator and the Object pointer.

The statements.

```
x.getdata(100, 75.50);
x.display();
```

are equivalent to

```
ptr -> getdata(100, 75.50);
ptr -> display();
```

Since *ptr is an alias of x, we can also use the following method.

```
(*ptr).display();
```

The parentheses are necessary because th
Operator has higher precedence than th
indirection operator *.
In pointer to objects concept, we can al
create the objects using pointers and ne
operator as follows:

```
Item *ptr = new Item;
```

This statement allocates enough memory
for the data members in the object struct
and assigns the address of the memor
space to ptr. Then ptr can be used to
refer to the members as shown below:

```
ptr -> display();
```

We can create an array of objects u
pointers as

```
Item *ptr = new Item[10];
                // array of 10 objects.
```

Creates memory space for an array of
10 objects of Item. Remember, in such
cases, if the class contains constructor, it
must also contain an empty constructor.
If a class has constructor with argumen
and doesn't include an empty constructor, th
we must provide the arguments when object
created. Following program illustrates the use

```
// Pgm for Pointers to Objects
#include<iostream.h>
#include<conio.h>
class Item
{
private:int code;
        float price;
public:
        void getdata(int a,float b)
    {
        code=a;
        price=b;
        }
        void display()
        {
        cout<<"\n Code= "<<code;
        cout<<"\n Price= "<<price;
        }
};
void main()
{
clrscr();
cout<< “\n\t\t\t ***** OUTPUT *****”;
Item *p=new Item[3];
Item *d=p;
int x,i;
float y;
cout<<"\n Enter three times following details:";
for(i=0;i<=2;i++)
{
cout<<"\n Enter Code & Price:";
cin>>x>>y;
p->getdata(x,y);
p++;
}
cout<<"\n Three details for Item are:";
for(i=0;i<=2;i++)
{
cout<<"\n Item:"<<i+1;
d->display();
d++;
}
getch();
}
```

******* OUTPUT *******

Enter three times following details:
Enter Code & Price:1
32.5

Enter Code & Price:2
56.7

Enter Code & Price:3
98.32

Three details for Item are:
Item:1
Code= 1
Price= 32.5
Item:2
Code= 2
Price= 56.700001
Item:3
Code= 3
Price= 98.32

Uses of New, delete and this keyword.

* New keyword :-

c++ strongly supports 2 keyword for allocating memory & relacing the memory in better & easerier way. These keyword are new & delete these are also know or free stored operator In c++ an object can be created by using new operator. after creating an object the memory space is deallocated or destroyed by using delete operator. A object created inside a block with new aill remain in existance Untill, It explicity distoryed by using delete Thus, the life time of an object is directly under control on is unrelated to the block structure of the program, the new operatore can be used to create object in any time. The general syntax for new operatore is syntax :-

pointer variable = new datatype;

eg. int *p = new int;

where, *p - pointer variable of int type.

Here, pointer variable is a pointer the new operator allocated sufficient memory hold a data object a data & returns the address of a object.

The memory which are allocated by using the new operator can be de-allocate by using delete operator. The delete operator is also as memory realize operator in c++.

When a data object which is known longer needed it is distroyed to relase the memory space for rease.

The general syntax for delete keyword is :-

Syntax :-

Delete pointer variable;

i.e. delete *p;

or

delete p;

This keyword :- (this pointer)

C++ uses a unique keyword called this which is uses to represent An object that is hoke the member function. These is a pointere that point to the object foe which these function was called.

This Unique pointer is automatically passed to a member function when it is called the pointer there as an implicit argument do all the membere function.

Generally, we can implicitly used these pointer when we can overload unary or binary operatores by using member function.

Operator Overloading

Operator overloading is a main feature of c++ language. It is an important technique that inhance the power of extensibility of c++. while adding two variable of user define type with the same syntax Applied to the basic type.

The machinisam of giving a special meaning to an operator is know as operator overloading.

Operator overloading provides a flexible option for the creation of new defination For most of the c++ operator we can create a new language of our own by the creative use of the function and operator overloading technique.

we can overload all the c++ operator except the following.

1) class member Access operator (. , .*)
2) scope Resulution operator (::)

3) size operator (sizeof)
4) conditional operator (? :)

Remainber that when an operator is overloaded, it original meaning is not loss.

For Example :-

The operator '+' which has been overloaded add two vectors basically used to add two integers.

* Defining operator overloading

To define an additional task to an operator, we must specifie the meaning to the class to which the operator is applied. This is done with the help of a special function called operator function, which describe the task.

The general form of an operator function is,

13

return type classname ::
operator (op_arglist)
{
function body
}

where, return type is the type of value returned by the specified operation and Op is the operator which has been overload. Some common example are,

vector operator + (vector);
vector operator - ();
int operator == ();

Overloading Unary operator (-) minus.

A unary operator minus (-) has only one operand as we know that this '-' operator changes the sign of an operand when applied to a basic data items. it can be applied to a basic data items., it can be

applied to ~~a basic data items~~ an object should changes the sign of each of its data items. it can also be applied to an object in the same way as it applied to an int or float variable.

The Following program demonstrate the overloading of unary '-' operator.

program:-

```
#include<iostream.h>
class space
{
  int x,y,z;
public:
  void getdata();
  void display();
  void operator -();
};
void space :: getdata()
{
cout<<"Enter value of x,y,z"
  cin>>x>>y>>z;
}
void space :: display()
```

```
{
 cout<<" x="<<x;
 cout<<" y="<<y;
 cout<<" z="<<z;
}
void space :: operator -()
{
    x = -x;
    y = -y;
    z = -z;
}
   void main()
   {
     space s;
     s. getdata();
     s. display();
      -s;
     s. display();
   }
```

The above program will generate an output as

```
// Pgm for Overloading Unary Minus Operator
#include<iostream.h>
#include<conio.h>
class Substract
{
private:intx,y,z;
public: void getdata(int a,intb,int c);
        void display();
        void operator-();
};
void Substract::getdata(int a,intb,int c)
{
x=a;
y=b;
z=c;
}
void Substract::display()
{
cout<<x <<" ";
cout<<y <<" ";
cout<<z <<"\n";
}
void Substract::operator-()
{
x=-x;
y=-y;
z=-z;
}
void main()
{
clrscr();
Substract s;
cout<< “\n\t\t\t ***** OUTPUT *****”;
s.getdata(10,-20,30);
cout<<"\n Original Values are:";
cout<<"\n s: ";
s.display();
cout<<"\n After applying Unary Minus operator:";
-s;
cout<<"\n s: ";
s.display();
getch();
}
```

******* OUTPUT *******

Original Values are:
s: 10 -20 30

After applying Unary Minus operator:
s: -10 20 -30

as well as subclasses but also the derived classes.

* Overloading Unary '+' operator.

Like unary '-' operator, the same mechanism can be use to overload binary '+' operator. The binary '+' operator are two complex no. by using some feature of a friend function. The general representation for binary operator '+' is,

```
[ Operator +() ]
```

It has some main features they are,

1. It takes only one complex type of argument.
2. It return's complex types of value.
3. It is a member function of complex value.

Following program demonstrate the concept of overloading (+) operator.

```
#include <iostream.h>
class complex
{
  int x,y;
 public:
        complex()
      {
     cout<<"This is constructor";
      }
      complex (int r, int s)
      {
         x = r;
         y = s;
      }
     complex operator +();
        void display();
};
```

```
// Pgm for Overloading Binary Plus Operator
#include<iostream.h>
#include<conio.h>
class Addition
{
private: float x,y;
public: Addition()
{
}
Addition(float a,float b)
{
x=a;
y=b;
}
Addition operator+(Addition);
void display();
};
Addition Addition::operator+(Addition a)
{
Addition temp;
temp.x=x+a.x;
temp.y=y+a.y;
return(temp);
}
void Addition::display()
{
cout<<x<<"i"<<"+"<<y<<"j"<<"\n";
}
void main()
{
clrscr();
Addition a1,a2,a3;
cout<<"\n\t\t\t ***** OUTPUT *****";
a1=Addition(2.5,3.5);
a2=Addition(1.6,2.7);
a3=a1+a2;
cout<<"\n a1= ";
a1.display();
cout<<"\n a2= ";
a2.display();
cout<<"\n ----------------------";
cout<<"\n a3= ";
a3.display();
getch();  }
```

***** OUTPUT *****

```
a1= 2.5i+3.5j
a2= 1.6i+2.7j
----------------------
a3= 4.1i+6.2j
```

Rules & pitfall of Overloading

following are some rule for operator overloading.

(1) only existing operator can be overloaded new operators can not be created.

(2) The overloaded operator must have atleast one operands of user defined type.

(3) we can not change the basic meaning of operators.

(4) overloaded operator follows the syntax rule of the original operator. They can't be over riden.

(5) Their some operators which can not be overloaded they are,

1) [. .*] class member access operator

2) Scope Resolution operator [::]

3) size of operator (sizeof)

4) conditional operator (?:)

(6) we can't use friend function to overload certain operator Those operator are,

= Assignment operator.
( ) Function call operator
[ ] subscripting operator
→ class member access operator.

(7) Unary operator overloaded by means of a member function which it's no explicit argument & returns no explicit value overloaded by means of a friend function.

(8) Binary operator overloaded through a member function take one explicit argument which are overloaded through a friend function by using explicit argument.

(9) When binary operators overloaded through a member function, the left hand operant must be a object of the relivant class.

20

20

(10) Arithmatic operators such has +, −, ×, / must explicitly return value, they must not attempt to change their own arguments.

Inheritance.

Introduction :-

Reusability is the another feature of OOP's. due to this C++ classes can may be used in several ways once's has been written & tested. it can be adopted by other pgm to suit their requirement this is basically done by creating a new classes, reusing the properties of existing once's.

Define :-

Inheritance / derivation :-

The mechnism of deriving of new class from an a existing class is called inheritance or derivation.

The old class is refer to as the base class and the new class is called as derived or subclass.

The derived class inherits some or all of the features of the base class. A class can also inherits

properties from more than one class or from more them.

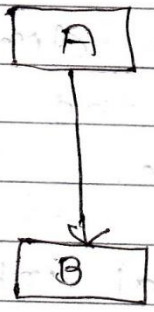
Types of Inheritance / different forms of inheritance.

Basically inheritance has five different forms they are :-

1. single inheritance.
2. multiple inheritance.
3. hireachical Inheritance.
4. multilevel inheritance.
5. hybrid inheritance.

1. single inheritance :- A derived class with only one base class is called single inheritance.

Diagreamatically we can represent single 'Inheritance'



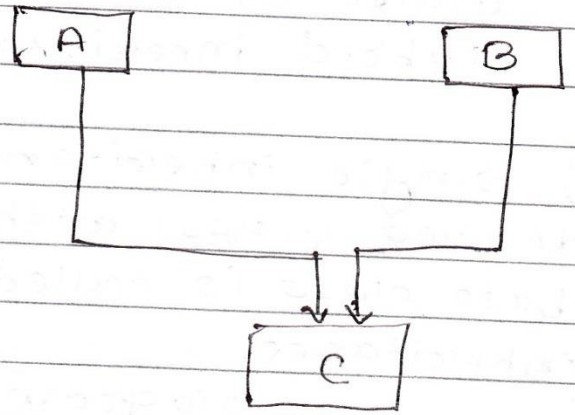
Where, A is base

class & B is derived class. The direction of arrow indicates the direction of inheritance.

(2) Multiple inheritance :-

A derived class the several base classes is called multiple inheritance.

Diagramatically we can represent multiple inheritance



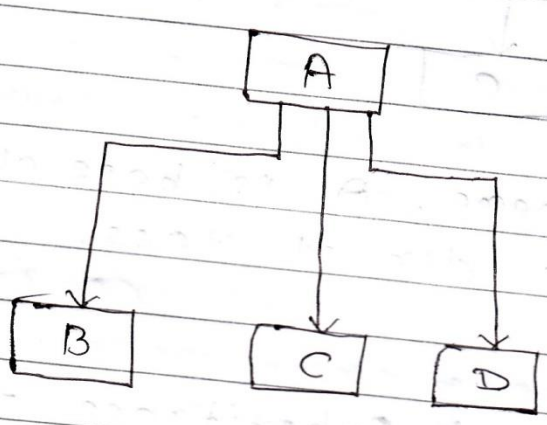
where, A & B is base class & C is derived class, The direction of arrow indicates the direction of inheritance.

(3) hireachical inheritance :-

When one class may be inherited by more than

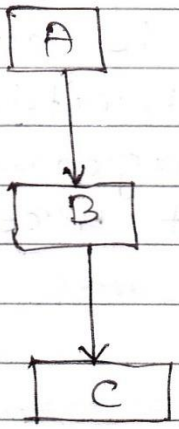
1 classes then such type of inheritance is called as hireachical inheritance.

Diagramatically we can represent hireachical inheritance.



where, A is base class & B, C are derived class. The direction of arrow is indicate the direction of inheritance.

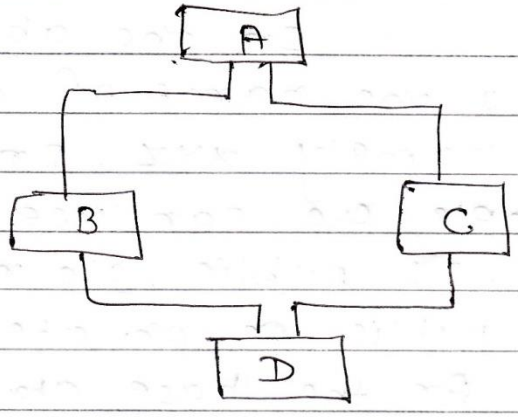
(4) multilevel inheritance :- The mechanism of deriving a class from another derived class is known as multilevel inheritance. Diagramatically we can represent multilevel inheritance as,



where, A is base class & B & C derived class.

(5) Hybrid Inheritance :-

The inheritance which can use the concept of multiple & multilevel inheritance then such type of inheritance is called hybrid inheritance, simply we can say that hybrid inheritance can uses both the feature of multiple as well as multilevel inheritance. Diagramatically we can represent hybrid inheritance.



Defining derived classes

A derived class can be defined by specifing it's relationship with the base class in ~~adding~~ addition to it's own details. The general syntax for defining derived class is,

Syntax :-

```
class derived classname :
        visibility mode base
         classname
{
member of derived class
}
```

i.e. class ABC : public XYZ

```
{
- - - - -
- - - - -
- - - - -
};
```

In the above example abc is the name of derived class namely xyz. In the above example ae can use the visibility mode, as public, ae may either use public or private visibilities mode for the base class.

To mode that the default visibilites mode are used for deriving class is private.

Visibility control or access modifier in c++.

c++ mainly uses 3 visibilities control they are:-

1. public
2. private
3. protected

1. public: Any variable or a method or visible to the entired class in which it is defined. This is possible by simply declaring the variable or method or public for i.e public int number;
   public void sum()

Below that A variable or method declared as public has to voided possible visibilites & accessable anywhere, this could be benificial to privent many programs and this task us to the next level of protection.

(2) private: private access specifiees has the higher degree protection. They are accessible only within their own classes. They can not be inhereited by sub-class and their free hom in sub-classes

A method or member declared as private behaves like a method declared or a final that is it privents the method from being sub classed.

3) protected :-

The protected access specifies lies in betaeen public & private specifies that is the protected modifiee makes the field not only all the classes

Question Bank on Unit- IV

1) **Explain array of objects with suitable program.**
2) **Explain pointers to objects with suitable program.**
3) **Explain how array can be used as a class member with example.**
4) **Explain the use of new & delete keywords.**
5) **Explain this pointer(operator).**
6) **What is operator overloading? Explain rules(pitfalls) of operator overloading?**
7) **Write a program to overload Unary Minus operator.**
8) **Write a program to overload Binary Plus operator.**
9) **What is inheritance? Explain different forms(types) of inheritance.**
10) **Write difference between Single & Multiple inheritance.**
11) **Explain: i) public ii) private iii) protected**

****