Unit II

2 Inheritance and Pointers

2.1 : Basic Concept of Inheritance

Q.1 Define the term inheritance.

Ans. : Inheritance is a property in which data members and member functions of some class are used by some other class.

2.2 : Base Class and Derived Class

Q.2 Explain the term base class and derived class.

Ans. : • The class from which the data members and member functions are used by another class is called the **base class.**

The class which uses the properties of base class and at the same time can add its own properties is called **derived class.**

2.3 : Public and Private Inheritance

Q.3 Write a C++ program to inherit a base class in public mode.

Ans. :

```
#include <iostream>
using namespace std;
class Base
{
 int x;
 public:
 void set_x(int n)
 {
    x = n;
 }
```

```
void show_x( )
{
   cout <<"\n x= "<<x;
}
};
// Inherit as public
class derived : public Base
{
 int y;
 public:
 void set_y(int n)
 {
   y = n;
 }
 void show_y()
 {
   cout <<"\n y= "<<y;
 }
};

int main()
{
 derived obj;//object of derived class
 int x, y;
 cout<<"\n Enter the value of x";
 cin>>x;
 cout<<"\n Enter the value of y";
 cin>>y;
//using obj of derived class base class member is accessed
 obj.set_x(x);
 obj.set_y(y); // access member of derived class
 obj.show_x(); // access member of base class
 obj.show_y(); // access member of derived class
 return 0;
```

```
}
```

Output

```
Enter the value of x30

Enter the value of y70

        x= 30
        y= 70
```

In above program the *obj* is an object of derived class. Using *obj* we are accessing the member function of base class. The derived class inherits base class using an access specifier public.

Q.4 Write a C++ program to inherit base class in private mode.

Ans. :

```
#include <iostream>
using namespace std;
class Base
{
 int x;
 public:
 void set_x(int n)
 {
    x = n;
 }
 void show_x( )
 {
    cout <<"\n x= "<<x;
 }
};
// Inherit as private
class derived : private Base
{
 int y;
 public:
 void set_y(int n)
```

```
{
   y = n;
}
void show_y()
{
   cout <<"\n y= "<<y;
}
};


int main()
{
derived obj;//object of derived class
int x, y;
cout<<"\n Enter the value of x";
cin>>x;
cout<<"\n Enter the value of y";
cin>>y;
obj.set_x(x); // error: not accessible
obj.set_y(y);
obj.show_x(); // access member of base class
obj.show_y(); // error:not accessible
return 0;
}
```

As indicated by the comments the above program will generate error messages "*not accessible*". This is because the derived class inherits the base class privately. Hence the public members of base class become private to derived class.

2.4 : Protected Members

Q.5 Explain why and when do we use protected instead of private ?

☞ **[SPPU : Dec.-15, Marks 4]**

Ans. : The *protected* access specifier is equivalent to the *private* specifier with the sole exception that protected members of a base class are accessible to members of any class derived from that base. Outside the base or derived classes, protected members are not accessible.

Thus normally protected access specifier is used in the situations when the immediate derived class members want to access the base class members but the derived-derived class members are prohibited to access the base class members.

2.5 : Relationship between Base Class and Derived Class

Q.6 What are the advantages of inheritance ?

Ans. : One of the key benefits of inheritance is to minimize the amount of duplicate code in an application by sharing common code amongst several subclasses.

1. **Reusability :** The base class code can be used by derived class without any need to rewrite the code.
2. **Extensibility :** The base class logic can be extended in the derived classes.
3. **Data hiding :** Base class can decide to keep some data private so that it cannot be altered by the derived class.
4. **Overriding :** With inheritance, we will be able to override the methods of the base class so that meaningful implementation of the base class method can be designed in the derived class.

2.6 : Constructor and Destructor in Derived Class

Q.7 Explain the execution process of constructors and destructors in derived class.

Ans. : • When we create an object for derived class then first of all the Base class constructor is called and after that the Derived class constructor is called.

- When the main function finishes running, the derived class's destructor will get called first and after that the Base class destructor will be called.
- This is also called as **chain of constructor** calls.

```
class Base {
public:
 Base()
 {   cout << "Base constructor" << endl;  }
  ~Base()
 {   cout << "Base destructor" << endl;  }
};
class Derived:public Base {
public:
 Derived()
 {   cout << "Derived constructor" << endl;  }
 ~Derived ( )
 {   cout << "Derived destructor" << endl;  }
};
void main()
{
 Derived obj;
}
```

The output of above code will be invoking of base class constructor, then derived class constructor, then derived class destructor and finally base destructor.

2.7 : Overriding Member Functions

Q.8 Explain the function overriding concept with suitable example

Ans. : Definition : Redefining a function in a derived class is called function overriding.

For example - Consider following C++ program that uses the same function name i.e. **print_msg** in base class and derived class. The

function print_msg in derived class overrides the base class function print_msg

```
class A
{
    private:
        int a,b;
    public:
        void get_msg()
        {
            a=10;
            b=20;
        }
        void print_msg()
        {
            int c;
            c=a+b;//performing addition
            cout<<"\n C(10+20)= "<<c;
            cout<<"\n I'm print_msg() in class A";
        }
};
class B : public A
{
    private:
        int a,b;
    public:
        void set_msg()
        {
            a=100;
            b=10;
        }
        void print_msg()
        {
            int c;
```

```
            c=a-b;//performing subtraction in this function
             cout<<"\n\n C(100-10) = "<<c;
             cout<<"\n I'm print_msg() in class B ";
        }
};
void main()
 {
     A obj_base;
     B obj_derived;
     obj_base.get_msg();
     obj_base.print_msg();//same function name
     obj_derived.set_msg();
     obj_derived.print_msg();//but different tasks
 }
```

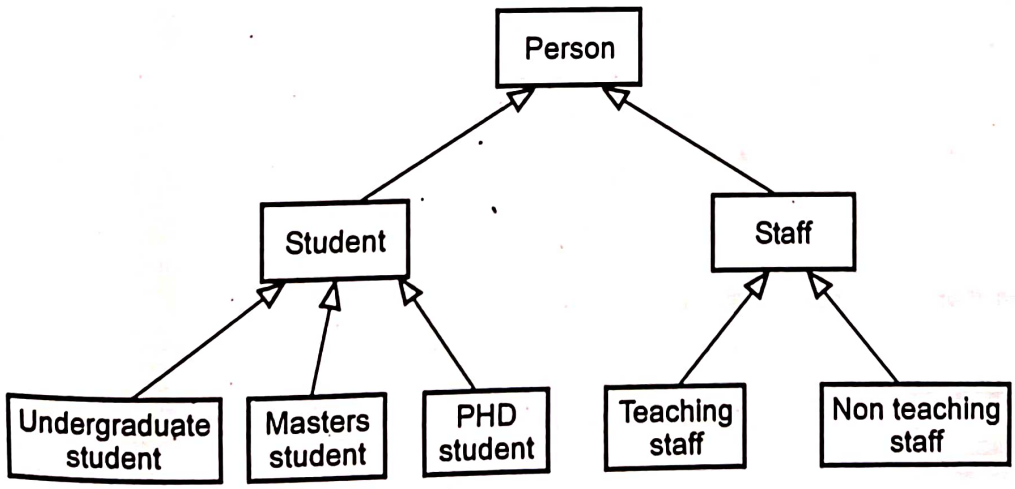
The above program will display 30 due to execution of **print_msg** function of base class and 90 due to execution of **print_msg** function of derived class.

2.8 : Class Hierarchies

Q.9 Write short note on class hierarchies.

Ans. : Inheritance is a mechanism in which using base class, various classes can be derived. Following diagram represents the class hierarchy. The class hierarchy is normally represented by **class diagram**

Example :



The implementation of class hierarchy is possible using **hierarchical inheritance.**

2.9 : Types of Inheritance

Q.10 With suitable examples, explain different types of inheritance.
☞ [SPPU : Dec.-15, Marks 4]

Or Write a note on type - single, multiple and hierarchical.
☞ [SPPU : May-16, Dec.-19, Marks 6]

Ans. : Various types of inheritance are - 1. Single inheritance 2. Multilevel inheritance 3. Multiple inheritance 4. Hybrid inheritance.

1. Single inheritance

In single inheritance one child class is derived from one base class.

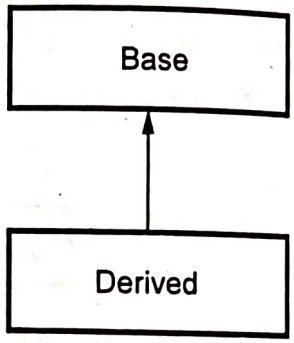


Fig. Q.10.1 Single inheritance

```
#include <iostream>
using namespace std;
class Base
{
public:
    int x;
    void set_x(int n)
    {
        x = n;
    }
    void show_x()
    {
        cout << "\n\t x = " << x;
    }
};
class derived : public Base
{
    int y;
public:
```

```
    void set_y(int n)
    {
        y = n;
    }
    void show_xy()
    {
        cout < "\n\t x = " << x;
        cout < "\n\t y = " << y;
    }
};
int main()
{
    derived obj;
    int x, y;
    cout << "\n Enter the value of x";
    cin >> x;
    cout << "\n Enter the value of y";
    cin >> y;
    obj.set_x(x);//inherits base class
    obj.set_y(y); // access member of derived class
    obj.show_x();//inherits base class
    obj.show_xy(); // access member of derived class
    return 0;
}
```

2. Multilevel inheritance :

It is a kind of inheritance in which the derived class is derived from a single base class which itself is a derived class.

```
class A
{
    protected:
        int x;
```

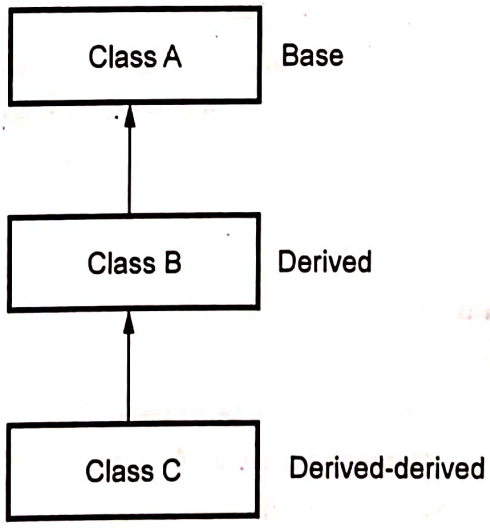


Fig. Q.10.2 Multilevel Inheritance

```
    public:
    void get_a(int a)
    { x=a; }
    void put_a()
    { cout<< "\n The value of x is "<< x; }
};
class B:public A
{
    protected:
        int y;
    public:
        void get_b(int b)
        {  y=b;   }
        void put_b()
        { cout<<"\n The value of y is "<<y;  }
};
class C:public B
{
    int z;
    public:
        void display()
        {
            z=y+10;
            put_a();//member of class A
            put_b();//member of class B
            cout<<"\n The value of z is "<<z;
        }
};
int main()
{
    C obj;//object of class C
    //accessing class A member via object of class C
    obj.get_a(10);
    //accessing class B member via object of class C
```

Output

The value of x is 10
The value of y is 20
The value of z is 30

```
    obj.get_b(20);
    ////accessing class C member via object of class C
    obj.display();
    cout<<endl;
    return 0;
}
```

3. Multiple Inheritance :

In multiple inheritance the derived class is derived from more than one base class.

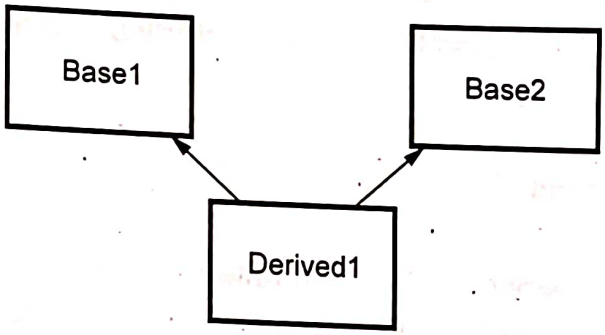


Fig. Q.10.3 Multiple inheritance

The implementation of multiple inheritance is as shown in Fig. Q.10.3

```
class Base1
{
   protected:
      int a;
   public:
      void set_a (int x)
      { a=x;  }
};
class Base2
{
   protected:
      int b;
   public:
      void set_b (int y)
      { b=y;  }
};
class Derived: public Base1, public Base2
{
```

```
    public:
        void addition()
        {
            int c;
            //reading value of a from Base1 class
            //reading value of b from Base2 class
            //perfoming addition of two numbers in Derived class
            c=a+b;
            cout<<"\n The Addition of two numbers = "<<c;
        }
};
int main ()
{
    Derived obj;//using object of derived class
    obj.set_a(10);//accesing Base1 class member function
    obj.set_b(20);//accessing Base2 class member function
    obj.addition();//accessing Derived class member function
    return 0;
}
```

The above program will use value of **a** from **Base1** class, value of **b** from **Base2** class and in **Derived** class the addition function for adding these two values is written. The output of above program will be 30.

4. Hierarchical Inheritance :

In this type of inheritance, there is single base class from which more than one derived classes can be derived. Fig. Q.10.4. Following program illustrates this idea.

```
class Base
{
    protected:
        int a;
    public:
        void set_a (int x)
        { a=x; }
```

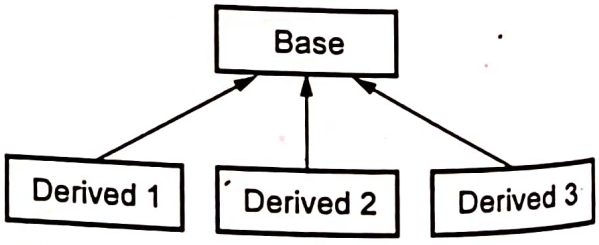


Fig. Q.10.4 Hierorchical Inheritance

```
};
class Derived1:public Base
{
    public:
        void add_ten()
        {
            int ans;
            ans=a+10;
            cout<<"\n After adding 10, result = "<<ans;

        }
};
class Derived2:public Base
{
    public:
        void Subtract_ten ()
        {
            int ans;
            ans=a-10;
            cout<<"\n After subtracting 10, result = "<<ans;

        }
};
class Derived3:public Base
{
    public:
        void mul_ten()
        {
            int ans;
            ans=a*10;
            cout<<"\n After multiplying by 10, result = "<<ans;
        }
};
int main ()
```

```
{
    Derived1 obj1;
    Derived2 obj2;
    Derived3 obj3;
    obj1.set_a(10); //using base class functionality
    obj1.add_ten();//using derived class functionality
    obj2.set_a(10); //using base class functionality
    obj2.Subtract_ten();//using derived class functionality
    obj3.set_a(10); //using base class functionality
    obj3.mul_ten();  //using derived class functionality
    return 0;
}
```

The above program passes value 10 to the base class functionality and performs addition by 10, subtraction by 10 and multiplication by 10 using three derived classes respectively.

2.10 : Ambiguity in Multiple Inheritance

Q.11 Explain the ambiguity problem in inheritance with suitable example.

Ans. : • Ambiguity is the problem that arise in multiple inheritance. It is also called as **diamond problem.**

Consider the following Fig. Q.11.1. (Refer Q.11.1 on next page)

• Now the code for the above design can be written as

```
class A
{
 public:
     read();
     write();
};
class B:public A
{
 public:
```

```
    read();
    write();
};
class C:public A
{
 public:
    write();
};
class D:public B,public C
{
 public:
        write();
};
```

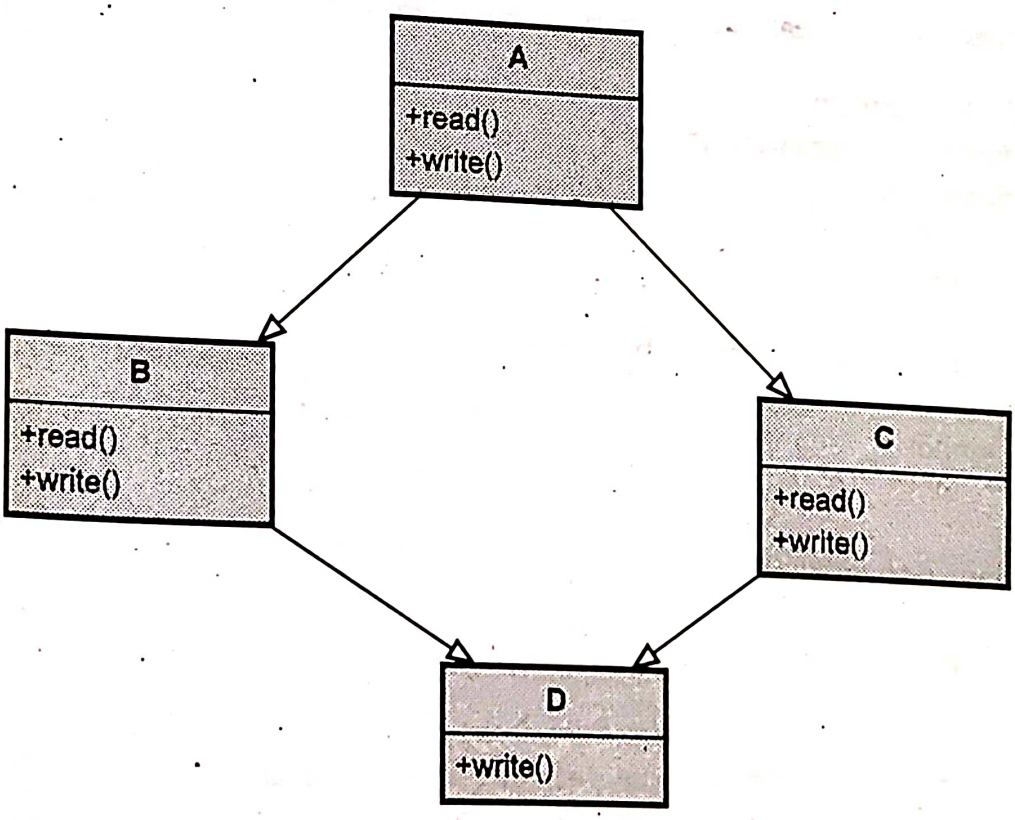


Fig. Q.11.1 Ambiguity in Inheritance

- We try to inherit the **write()** function of base class A in class B and C, which will be alright. But if try to use the **write()** function in class D then the compiler will generate error, because it is ambiguous to know which **write()** function to choose whether of class B or class C. This ambiguity occurs because the compiler understands that the class D is derived from both class B and class C and both of these classes have the versions of **write()** function. So the **A** class gets duplicated inside the class D object.

2.11 : Virtual Base Class

Q.12 What is virtual base class ? Explain it with suitable example.

Ans. : In order to prevent the compiler from giving an error due to ambiguity in multiple path or multiple inheritance, we use the keyword virtual. That means when we inherit from the base class in both derived classes, the base class is made virtual. The code that illustrates the concept of virtual base class is as given below -

C++ Program

```
#include<iostream.h>
class base {
    public:
     int i;
};
class derived1:virtual public base
{
 public:
     int j;
};
class derived2:virtual public base
{
     public:
     int k;
};
//derived3 is inherited from derived1 and derived2
//but only one copy of base class is inherited.
class derived3:public derived1,public derived2
{
```

```
public:
    int sum()
    {
        return i+j+k;
    }
};
void main()
{
 derived3 obj;
    obj.i=10;
    obj.j=20;
    obj.k=30;
    cout<" The sum is  = "<obj.sum();
}
```

Output

The sum is = 60

Program Explanation

In above program if we do not write the keyword virtual while deriving the classes derived1 and derived2 then compiler would have generated error messages stating the ambiguity in accessing the member of base class.

The sum is a function which can access the variables i, j and k of parent classes. In main function we have created an object obj of derived3 class and using this object i, j and k can be accessed.

2.12 : Abstract Class

Q.13 What is abstraction ? Why it is important ? Describe abstract base class. Illustrate an example to explain it.

☞ [SPPU : May-15, Marks 8]

Ans. : • Abstraction is a mechanism by which only essential information is presented to the outside world and background details are hidden.

• Due to abstraction only essential information is presented to the user.

- Abstract base class is class which contains at least one pure virtual function. The pure virtual function has a keyword **virtual** and it is assigned with 0.
- The abstract class is used to specify interface which is implemented by all the subclasses.
- Following C++ code represents the abstract base class.

```
class area
{
    double dim1, dim2;
    public:
      void setarea(double d1, double d2)
      {
          dim1 = d1;
          dim2 = d2;
      }
      void getdim(double &d1, double &d2)
      {
          d1 = dim1;
          d2 = dim2;
      }
      virtual double getarea() = 0; // pure virtual function
};
class square : public area
{
    public:
        double getarea()
        {
          double d1, d2;
          getdim(d1, d2);
          return d1 * d2;
        }
};
class triangle : public area
```

```
{
    public:
        double getarea()
        {
            double d1, d2;
            getdim(d1, d2);
            return 0.5 * d1 * d2;
        }
};
int main()
{
    area *p;
    square s;
    triangle t;
    int num1=10;
    int num2=20;
    cout<<"\n Calculating area of square";
    s.setarea(num1,num2);
    p = &s;
    cout <<"\nArea of square is : " << p->getarea();
    cout<<"\n Calculating area of triangle";
    t.setarea(num1,num2);
    p = &t;
    cout << "\nArea of Triangle is: "<< p->getarea() ;
    return 0;
}
```

The above program will display area of square as 200 and area of triangle is 100.

Q.14 What are abstract classes ? Write a program having student as an abstract class and create many derived classes such as engineering, science, medical etc. from the student class. Create their object and process them.

Ans. :

```
#include<iostream>
#include<cstring>
```

```
using namespace std;
class Student
{
 char name[10];
public:
    void SetName(char n[10])
    {
        strcpy(name,n);
    }
    void GetName(char n[10])
    {
        strcpy(n,name);
    }
    virtual void qualification()=0;
};

class Engg:public Student
{
public:
    void qualification()
    {
        char n[10];
        GetName(n);
        cout<<n<<" is a an engineering student"<<endl;
    }
};
class Medical:public Student
{
public:
    void qualification()
    {
        char n[10];
        GetName(n);
        cout<<n<<" is a medical student"<<endl;
```

```
    }
};
int main()
{
    Student *s;
    Engg e;
    Medical m;
    char nm[10];
    cout<<"\n Enter the name: "<<endl;
    cin>>nm;
    e.SetName(nm);
    s=&e;
    s->qualification();
    cout<<"\n Enter the name: "<<endl;
    cin>>nm;
    m.SetName(nm);
    s=&m;
    s->qualification();
    return 0;
}
```

Output

```
Enter the name:
Ramesh
Ramesh is a an engineering student

Enter the name:
Suresh
Suresh is a medical student
```

2.13 : Friend Class

Q.15 Explain the friend class concept.

Ans. : Similar to a friend function one can declare a class as a friend to another class. This allows the friend class to access the private data members of the another class.

For example

```
class A
      {
        private:
          int data;
          friend class B;//class B is friend of class A
        public:
        A()//constructor
        {
          data = 5;
         }
      };
  class B
     {
       public:
         int sub(int x)
         {
          A obj1; //object of class A
          //the private data of class A is accessed in class B
          // data contains 5 and x contains 2
          return obj1.data – x;
         }
    };
```

2.14 : Nested Class

Q.16 What is nested class ? Write a C++ program to demonstrate the concept of nested class.

Ans. : When one class is defined inside the other class then it is called the nested class. The nested class can access the data member of the outside class. Similarly the data member of the nested can be accessed from the main. Following is a simple C++ program that illustrates the use of nested class.

```
/**************************************************************************
The program for demonstration of nested class
**************************************************************************
******************/
#include<iostream.h>
class outer
{
 public:
    int a;     // Note that this member is public
    class inner
    {
     public:
       void fun(outer *o,int val)
     {
         o->a = val;
         cout<<"a= "<<o->a;
       }
    };//end of inner class
};//end of outer class
void main()
{
   outer obj1;
   outer::inner obj2;
   obj2.fun(&obj1,10); //invoking the function of inner class

}
```

Output

```
a= 10
```

2.15 : Pointers : Declaration and Initialization

Q.17 Define pointer and explain its initialization.

Ans. : Definition : A pointer is a variable that represents the memory location of some other variable. The purpose of pointer is to hold the memory location and not the actual value. For example -

```
a=10; /*storing some value in a*/
ptr=&a;/*storing address of a in ptr*/
b=*ptr;/*getting value from address in ptr and storing it in b*/
```

2.16 : Memory Management : New and Delete

Q.18 Explain the purpose of new and delete operators with suitable examples.

Ans. : The dynamic memory allocation is done using an operator **new.**

For example :

```
int *p;
p=new int;
```

We can allocate the memory for more than one element. For instance if we want to allocate memory of size in for 5 elements we can declare.

```
int *p;
p=new int[5];
```

The memory can be deallocated using the delete operator.

For example : `delete p;`

C++ Program using new and delete operators

```
int main ( )
{
 int i,n;
 int *p;
 cout << "How many numbers would you like to type? ";
 cin >> i;
 p= new int[i];//dynamic memory allocation
 if (p == 0)
```

```
    cout << "Error: memory could not be allocated";
  else
  {
   for (n=0; n<i; n++)
   {
    cout << "Enter The Number: ";
    cin >> p[n];
   }
   cout << "You have entered: ";
   for (n=0; n<i; n++)
    cout <<" " <<p[n];
   delete[] p;//dynamic memory deallocation
  }
  return 0;
}
```

2.17 : Pointers to Object

Q.19 Is it possible to use pointer to object ? If yes, explain how.

Ans. : Yes, it is possible to use pointer to object. Following C++ code shows how to use pointer to object

```
class Test
{
 int a;
public:
 Test(int b)
 {
   a = b;
 }
 int getVal()
 {
   return a;
 }
};
int main()
```

```
{
 Test obj(100), *ptr_obj;
 ptr_obj = &obj;

 cout <<"Value obtained using pointer to object is ..."<<endl;
 cout<<ptr_obj->getVal()<<endl;
 return 0;
}
```

Address of **obj** is stored in pointer variable.

2.18 : this Pointers

Q.20 Write short note on - this pointer. ☞ [SPPU : Dec.-14, Marks 3]

OR Explain the term - this pointer. ☞ [SPPU : Dec.-15, Marks 2]

OR What is the use of this pointer ? ☞ [SPPU : Dec.-19, Marks 2]

Ans. : The this pointer is a special type of pointer which is used to have the access to the address of its object. This pointer is passed as a hidden parameter to the member function call. For example -

```
class test
   {
    private:
       int num;
    public:
       void get_val(int num)
       {
      //this pointer retreives the value of obj.num
      //this pointer is hidden by automatic variable num
         this ->num=num;
       }
       void print_val()
       {
        cout<<"\n The value is "<<num;
       }
   };
```

2.19 : Pointers Vs. Arrays

Q.21 What is the difference between Pointers and Arrays ?
Ans. :

Sr.No.	Array	Pointer
1	Array is a collection of similar data type elements.	Pointer is a variable that can store an address of another variable.
2	Arrays are static in nature that means once the size of array is declared we can not resize it.	Pointers are dynamic in nature, that means the memory allocation and deallocation can be done using new and delete operators .
3	Arrays are allocated at compile time	Pointers are allocated at run time.
4	Syntax: type var_name[size];	Syntax: type *var_name;

2.20 : Accessing Arrays using Pointers

Q.22 Write a program using pointer for searching the desired element from the array.
Ans. :

```
#include<iostream>
using namespace std;
void main()
{
   int a[10], i, n, *ptr, key;
   cout<<"\n How Many elements are there in an array ? ";
   cin>>n;
   cout<<"\n Enter the elements in an array ";
   for (i = 0; i<n; i++)
      cin>>a[i];
```

```
    ptr = &a[0];/*copying the base address in ptr */
    cout<<"\n Enter the Key element ";
    cin>>key;
    for (i = 0; i<n; i++)
    {
        if (*ptr == key)
        {
            cout<<"\n The element is present ";
            break;
        }
        else
            ptr++; /*pointing to next element in the array*/
                /*Or write ptr=ptr+i*/
    }
}
```

2.21 : Pointer Arithmetic

Q.23 Explain various pointer arithmetic operations.

☞ [SPPU : Dec.-18, Marks 4]

Ans. : Pointer arithmetic means performing arithmetic operations on pointers. Following are various pointer arithmetic operations -

Operation	Meaning
x=*ptr1 * *ptr2	Multiplication of two pointer variables is possible in this way.
x=ptr1–ptr2	The subtraction of two pointer variables.
ptr1– – or ptr1 ++	The pointer variable can be incremented or decremented.
x=ptr1+10	We can add some constant to pointer variable.
x=ptr1–20	We can subtract a constant value from the pointer.

ptr1<ptr2	The relational operations are possible on pointer variable while comparing two pointers.
ptr1>ptr2	
ptr1==ptr2	
ptr1<=ptr2	
ptr1>=ptr2	
ptr1!=ptr2	

2.22 : Arrays of Pointers

Q.24 What is array of pointers ? Explain with pseudo code.

Ans. : The array of pointers means the array locations are containing the address of another variable which is holding some value. For example,

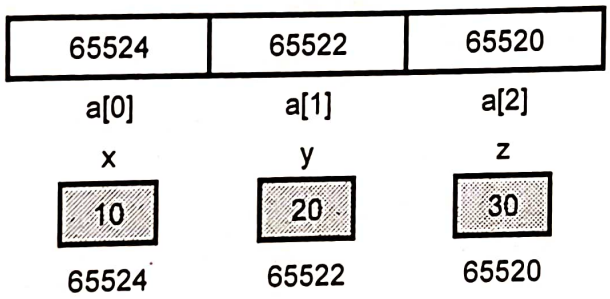


Fig. Q.24.1 Array of pointers

The array of pointers is the concept which is mainly used when we want to store the multiple strings in an array. Here we have simply taken the integer values in three different variables x, y and z. The addresses of x, y and z are stored in the array a. This concept is implemented by following simple C++ program.

```
#include<iostream>
using namespace std;
void main()
{
   int *a[10];/*array is declared as of pointer type*/
   int i, x, y, z;
```

```
    cout<<"\n Enter The Array Elements ";
    cin > x >> y >> z;

    a[0] = &x;/*storing the address of each variable in array location */
    a[1] = &y;
    a[2] = &z;

    for (i = 0; i<3; i++)
    {
        cout<<"\nThe element "<<*a[i]<<" is at location "<<a[i];
    }
}
```

Output

```
Enter The Array Elements
    10
    20
    30
    The element 10 is at location 65524
    The element 20 is at location 65522
    The element 30 is at location 65520
```

2.23 : Function Pointers

Q.25 Explain the concept of function pointers with suitable example. ☞ [SPPU : May-19, Marks 5]

Ans. : • The pointer to the function means a pointer variable that stores the address of function.

- The function has an address in the memory same like variable. As address of function name is a memory location, we can have a pointer variable which will hold the address of function.

- **Syntax**

```
Return_Type *pointer_variable (data_type);
```

- **For example,**

```
float (*fptr)(float);
```

Here **fptr** is a pointer to the function which has float parameter and returns the value float.

The following program illustrates the use of pointer to the function

```
void main()
{
   void display(float(*)(int), int);
   float area(int);
   int r;
   cout<<"\n Enter the radius ";
   cin>>r;
   display(area, r);/*function is passed as a parameter to another
function*/
}
void display(float(*fptr)(int), int r)
{
   /*call to pointer to function*/
   cout<<"\n The area of circle is "<(*fptr)(r);
}
float area(int r)
{
   return (3.14*r*r);
}
```

Q.26 Write a program to find the sum of an array Arr by passing an array to a function using pointer. ☞ [SPPU : Dec.-17, Marks 4]

Ans. :

```
#include <iostream>
using namespace std;
int fun(const int *arr, int size)
{
  int sum = *arr;
  for (int i = 1; i < size; ++i)
```

```
  {
    sum = sum+ *(arr+i);
  }
  return sum;
}

int main()
 {
  const int SIZE = 5;
  int numbers[SIZE] = {10, 20, 90, 76, 22};
  cout << "The sum of array is: "<<fun(numbers, SIZE) << endl;
  return 0;
}
```

Q.27 Explain pointer to a variable and pointer to a function. Use suitable example. ☞ [SPPU : May-18, Marks 4]

Ans. : • **Pointer to variable :** A pointer is a variable that represents the memory location of some other variable. The purpose of pointer is to hold the memory location and not the actual value.

• Consider the variable declaration

```
int *ptr;
```

ptr is the name of our variable. The * informs the compiler that we want a pointer variable, the int says that we are using our pointer variable which will actually store the address of an integer. Such a pointer is said to be **integer pointer.**

• **Pointer to function :** The pointer to the function means a pointer variable that stores the address of function.

• **Syntax**

```
Return_Type *pointer_variable (data_type);
```

For example

```
float (*fptr)(float);
```

Here **fptr** is a pointer to the function which has float parameter and returns the value float. Note that the parenthesis around fptr; otherwise the meaning will be different.

The following program illustrates the use of pointer to the function

```
#include<iostream>
using namespace std;
void main()
{
    void display(float(*)(int), int);
    float area(int);
    int r;
    cout<<"\n Enter the radius ";
    cin>>r;
    display(area, r);/*function is passed as a
        parameter to another function*/
}
void display(float(*fptr)(int), int r)
{
    /*call to pointer to function*/
    cout<<"\n The area of circle is "<(*fptr)(r);
}
float area(int r)
{
    return(3.14*r*r);

}
```

2.24 : Pointers to Pointers

Q.28 What is pointers to pointers ? Explain.

Ans. : A pointer can point to other pointer variables which brings the multiple level of indirection.

```
void main()
{
```

```
    int a;
    int *ptr1, **ptr2;
    a = 10;
    ptr1 = &a;
    ptr2 = &ptr1;
    cout<<"\n a = "<<a;
    cout<<"\n *ptr1 = "<<*ptr1;/*value at address in ptr1*/
    cout<<"\n ptr1 = "<<ptr1;/*storing address of a*/
    cout<<"\n *ptr2 = "<<*ptr2;/*storing address of ptr1*/
    cout<<"\n ptr2 = "<<ptr2;/*address of ptr2*/

}
```

In above program address of variable **a** is stored in a pointer variable **ptr1**. Similarly address of pointer variable **ptr1** is stored in variable **ptr2**. Thus **ptr2** is a pointer to pointer because it stores address of (65524) a variable which is already holding some address (65526). Hence we have declared ptr2 as :

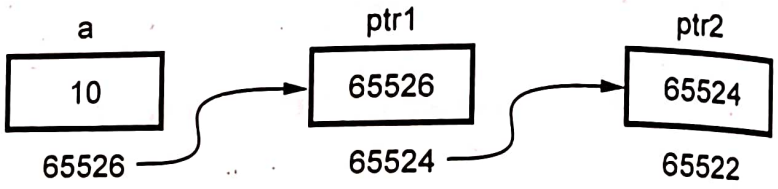


Fig. Q.28.1 Pointer to Pointer

```
int **ptr2;
```

2.25 : Pointers to Derived Classes

Q.29 Is it possible to declare a pointer pointing to a derived class ? If yes, justify.

Ans. : • It is possible to declare a pointer that points to the derived class.

• Using this pointer, one can access the data attributes as well as member functions of derived class.

Example Program

```
#include<iostream>
using namespace std;
class Base
```

```
{
public:
    int a;
};
class Derived :public Base
{
public:
    int b;
    void display()
    {
        cout << "\n a= " << a << "\n b= " << b;
    }
};
int main()
{
    Derived obj;
    Derived *ptr; //Pointer to derived class
    ptr = &obj;
    ptr->a = 100;
    ptr->b = 200;
    ptr->display();
}
```

Output

```
a= 100
b= 200
```

2.26 : Null Pointer

Q.30 What is NULL pointer ? Explain.

Ans. : • Pointer that are not initialized with valid address may cause some substantial damage. For this reason it is important to initialize them. The standard initialization is to the constant NULL.

• Using the NULL value as a pointer will cause an error on almost all systems.

- Literal meaning of NULL pointer is a pointer which is pointing to nothing.
- NULL pointer points the base address of segment.

Following is a simple program that illustrates the problem of Null pointer

```
/*****************************************************************
Program for illustrating null pointer problem
*****************************************************************
***********/
#include <iostream.h>
#include <string.h>
void main()
{
    char *str=NULL;
    strcpy(str,"HelloFriends");
    cout<str;
}
```

In above program we are trying to copy some string in the Null pointer. One can not copy something to Null pointer. Due to which the Null Pointer problem occurs.

2.27 : Void Pointer

Q.31 What is void pointer ?

Ans. : • The void pointer is a special type of pointer that can be used to point to the objects of any data type. It is also known as generic pointer.

- The void pointer is declared like normal pointer declaration, using the keyword void.
- For example :

```
void *ptr;
```

END...