Subject :- object Oriented Programming Using C++
Unit IV :- Pointers and Polymorphism in C++.

- Concept Of Pointer :-

- In object oriented Programming Pointer is an special type of variable that stores the memory address of another variable Instead of storing a value directly, a pointer stores the address where the value is kept in memory.

In C++ Pointer has two main operations :-

- Address-of (&) : Gets the memory address of a variable.
- Dereference (*) :- Accesses the value stored at the address the Pointer holds.

For. e.g.

```
int x = 10;
```

Here x stores value 10

```
int *p;
```

Here p is pointer variable.

```
p = &x;
```

p store address of variable x.

Pointer Declaration :-

- Pointer variable is a special variable It Can Store address of another variable.
- Pointer variable can be declare by using '*' operator

To declare a Pointer by using two step -

- To give Pointer name.
- the type of data who address it will be store.

Syntax of Pointer Declaration :-

- datatype *pointer_variable_name ;

datatype - type of variable whose address will be stored.

* - indicates that the given variable is a pointer variable.

Pointer_variable_name :- name of the Pointer variable.

For. e.g. - (Integer pointer declaration).

```
int *p; //P is a Pointer variable it can be store add of integer variable.
```

Float Pointer Declaration :-

```
Float *P;
```

This Pointer store address of Float variable.

Character Pointer Declaration :-

```
char *P;
```

This Pointer store address of character variable.

Pointer Initialization

To declare pointer then After declaration of Pointer variable, the pointer is usually initialized with the address of variable.

```
int x = 10;
int *P = &x; // Initialize Pointer variable P.
```

Pointer Operator :-

There are mainly two types of pointer operators are used:

1) Address of operator (&).
2) Dereference operator (*).

Address of operator (&) :-

The address-of operator (&) is a unary operator that returns the memory address of its of its operand. This operator can be used to store address of variable

For. e.g.

```
#include<iostream.h>
using namespace std;
int main()
{ int x = 20;
  int *p = &x; // address of operator.
  cout << " The address of the variable x is =" ptr;
  return 0;
}
```

Dereference Operator (*)

This operator is an unary operator it can be store and return value of variable.

This operator can be used to declare pointer variable.

For e.g.

```
#include<iostream.h>
using namespace std;
int main()
{ int x = 20
   int *p; // Dereference operator use to delare pointer var
   p = &x;
cout <<" value of x = " <<(*p);
  return 0;
}
```

pointer Arithmetic :-

In c++ pointer arithmetic can be performing operations such as increment, decrement, addition, and substraction.

In c++ to perform pointer arithmetic operation on pointer variable to move and access memory locations efficiently.

common operation include in pointer arithmetic :-

p++ :- Move to next memory location of the same data type.

p-- :- Move to previous memory location of the same data type.

p+n :- Move forward by n elements.

p-n :- Move backword by n elements.

Incrementing a pointer :- The value of pointer is incremented depending on datatype of variable.

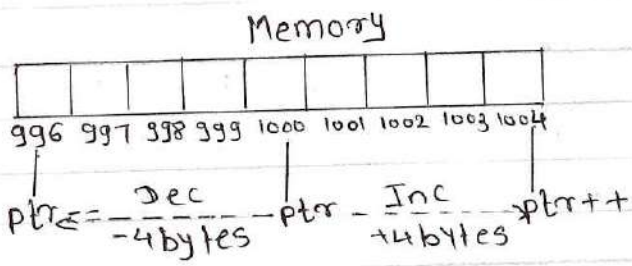
For. e.g.

If an integer pointer ptr holds the address 1000 andwe increment the pointer, then the pointer will be incremented by 4 or 8 bytes, and pointer hold the address 1004 or 1008

Decrementing a pointer :-

The value of pointer is decreased according to the size of the data type of variable.

If an integer pointer holds the address 1000 and we decrease the pointer, then the pointer will be decremented by 4 or 8 bytes. and pointer hold the address 996 or 992



pointer Increment and Decrement.

Addition of constant to pointers :-

We can add integer values to pointers and the pointer is adjusted based on the size of the datatype it points to.

For. e.g.-

if an integer pointer ptr stores the address 1000 and we add the value 5 to the pointer.

ptr +5

It will calculate -

1000 + (5 * 4 (size of int)) = 1020.

Subtraction of constant from pointer :-

We can also subtract a constant from pointers and it is the same as the addition of a constant to a pointers. For e.g. if an integer pointer ptr stores the address 1000 and we subtract the value 5 from the pointer.

ptr - 5

It will calculate -

1000 - (5 * 4 (size of int)) = 980

For. e.g.

```
#include<iostream.h>
using namespace std;
int main()
{
    int n = 10;
    int * ptr = &n
    cout <<"Before Arithmetic operation : "<<ptr<<endl;
    ptr++;
    cout<<" After increment : <<ptr <<endl;
     ptr--;
```

```
    cout<<" After Decrement " <<ptr<<endl;
     ptr= ptr+1;
    cout <<"Adding 1 to ptr" << ptr<<endl;
     Ptr= ptr-2;
    cout <<" Subtract 2 from ptr"<<ptr<< endl;
   return 0;
}
```

Pointer to object :-

- It is an pointer variable that stores the address of an object in memory.
- In c++ programming object pointers are declared by using the * symbol.
- object pointers are used to access members of an object indirectly by using the arrow operator (→).

Syntax :-

```
class_name *ptr
Ptr = &object;
ptr→ member_function();
```

dynamically allocating memory for a new object.

```
class_name *Ptr = new class_name();
```

Accessing Members :-

```
ptr→ memberfunction(); // call member function
ptr→ datamember = value; // call data member.
```

For. e.g.

```
class  myclass{
 public:
  void display()
  { std:: cout<<"Hello";
  }
};
int main()
{ myclass obj;
  myclass *ptr =&obj
    ptr→ display();
    return 0;
 }
```

This pointer :-

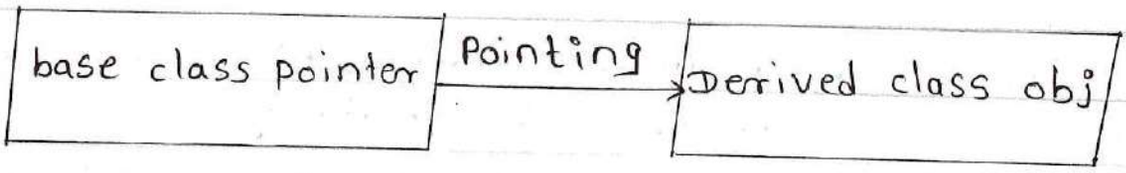
In c++ programming to use unique keyword call 'this' to represent an object that invokes a member fun^n. This unique pointers is automatically passed to member function when it is invoked 'this' is a pointer that member function when it is invoked. 'this' is a pointer that always point to an object for which the member function was called e.g. the function call A.max() will set the pointer 'this' to the address of the object A. Then suppose we call B.max() the pointer this will store address of object B.

```
#include<iostream.h>
class sample
{ int a;
  public:
  void setdata(int x)
  { this → a=x;
  }
  void putdata()
  { cout << this→a;
  }
};
void main()
{
    Sample s;
    s.setdata(100)
    s.putdata();
}
```

pointer to derived class :-

- In c++ pointer can be used for base objects as well as objects of derived class.
- A pointer to the object of derived class & a pointer to the object of base class. are type compatible. (may be used in different way).

The pointer of Base class. pointing different objects of the derived class.



In c++ programming can be allow base class pointer can be point to the derived class objects.
for e.g.

```
class base          class derived: public base
{ .....             { .....
  .....               .....
};                  };
```

The to write.

```
base *p1;
derived d_object;
p1 = &d_object;
```

```
#include<iostream.h>
using namespace std;
class base{
public
  void show()
  { cout<<"base in";
  }
};

class derived : public base{
public:
    void show(){
     cout<<" derived in";
  }
};
void main()
{ derived d1;
   base *ptr;
   Ptr= &d1;
   Ptr→ show();
  return 0;
}
```

polymorphism :-

The word "polymorphism" means having many forms. In OOP Polymorphism as the ability of a message to be displayed in more than one form. A real-life e.g. of polymorphism is a person who at the same time can have different characteristics.

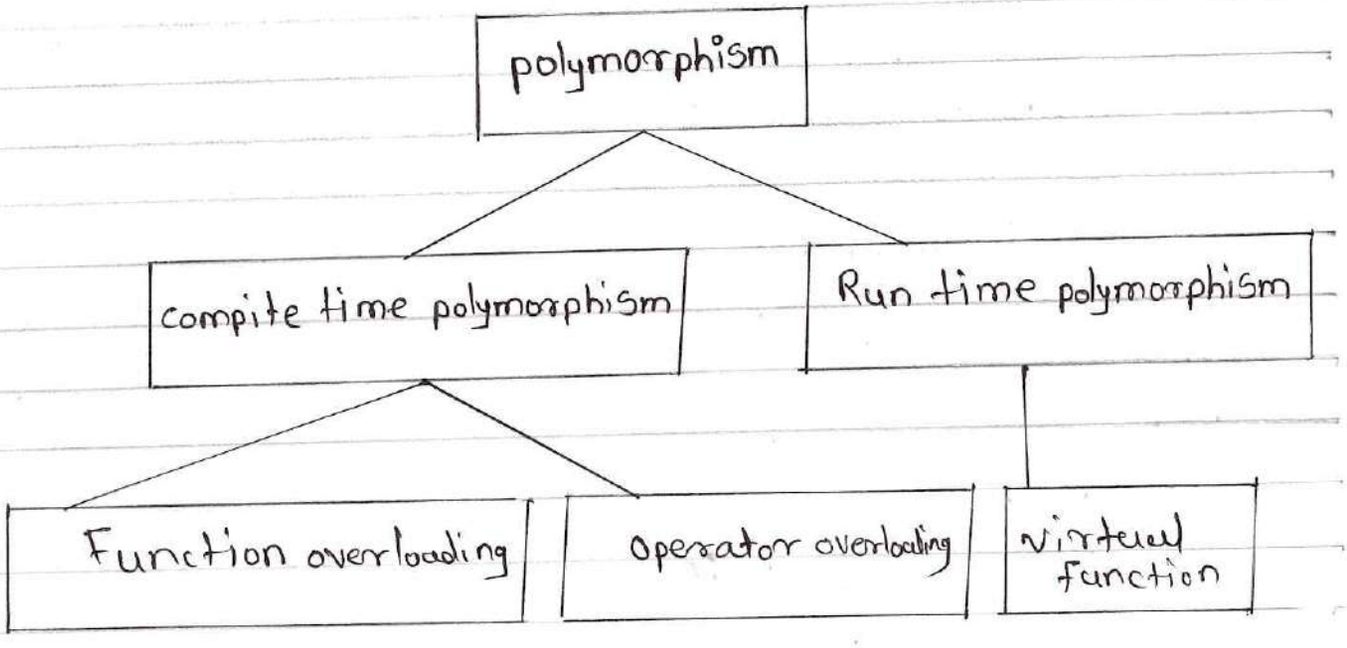
- polymorphism is considered one of the important feature of object-Oriented Programming.

Types of polymorphism :-

There are two types of polymorphism:

compile time polymorphism.

Run time polymorphism.



compile time polymorphism : This type of polymorphism is achieved by function overloading or operator overloading.

Function Overloading :- When there are multiple functions with same name but different parameter then these functions are said to be overloaded. Functions can be overloaded by change in number of arguments or change in type of arguments.

For. e.g.

```
#include<iostream.h>
class A
{ public
    void function(int x)  //First Function with 1 int parameter
    { cout<<" value of x is = "<<x<<endl;
    }
    Void function (int x, float y)
    { cout<<" value of x = "<<x<<endl;
      coret << " value of y = " <<y<<endl;
    }
    void function (Float x)
    { cout<<" value of x = "<<x<<endl;
    }
};
int main()
{ a obj;
  obj.function(5);
  obj.function(5,5.5);
  obj.function(5.7);
  return 0;
}
```

Operator overloading

Operator overloading in c++ has ability to provide the operator with a special meaning for particular data type the ability is known as operator overloading.

E.x. addition (+) operator. For use to concatinate of two string & for integer to add. two integers. The << add >> operator are binary shift operator but are also used with input & output streams. this is possible due to operator overloading

Rules For Operator Overloading.

Only existing operators can be overloaded :- No new operators can be created.

At least one operand must be a user-defined type. (class or structure)

operator The overloaded operator must have at least one operand that is of user defined data type.
- We can't change the basic meaning of an operator. That is to say, we can't redefine the plus(+) operator to subtract one value from other.
- overloaded operators follow the syntax rules of the original operators. They can't be overridden.
- There are some operators that can't be overloaded.
- We can't use friend functions to overloaded centains operators. However, member function can be used to oveload them
- Unary operators overloaded by means of member function, take no explicit arguments and return no explicit values, but, those overloaded by means of the friend function, take one referrence argument.
- Binary operators overloaded through a member function take one explicit argument and those which are overloaded through a friend function take two explicit argument.
- When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.
- Binary arithmetic operators such as +, -, *, and / must explicitly return avalue. They must not attempt to change their own arguments.

Overloading Unary Operator :-

Unary operator overloading allows you to redefine how unary operators behave four objects of a class.

Let us consider to overload (-) unary operator. In unary operator function, no arguments should be passed. It works only with one class objects. It is a overloading of an operator operating on a single operand.

For. e.g.

```
#include<iostream.h>
class distance
{ public:
      int feet, inch;
   distance (int f, int i)
   { this→feet = f;
   this→inch = i;
   }
   void operator - ()
   { feet --;
     inch --;
     cout <<" In feet & Inches = "<< feet <<" " <<inch ;
   }
};
int main()
{
   distance d1(8,9);
    -d1;
   return 0;
}
```

Overloading Binary Operators :-

The binary operators take two arguments and following are the eocamples. of Binary operator

Binary operator Overloading is the process of defining the behavior of a binary operator for object of a user-defined class.

It allows expressions involving objects to appear similar to expressions involving built-in data types.

```
#include<iostream.h>

class complex
{ private:
   int real, imag
   public:
    complex(int r=0, int i=0)
    { real = r;
      imag = i;
    }
```

```
complex operator +(const complex &c)
{   complex temp;
    temp.real = real + c.real;
    temp.imag = imag + c.imag;
     return temp;
}
 void display()
  { cout<<real<<"+"<<imag<<"i"<<endl;
};}
int main()
{  complex c1(10,20);
   Complex c2(5,8);
    Complex c3;
      c3 = c1 + c2;
      cout<<"First Complex no=";
      c1.display();
     cout<<"second complex no=")
      c2.display();
     cout<<"sum=";
     c3.display();
      return 0;
   }
```

Run-Time Polymorphism in C++ :-

Runtime polymorphism is achieved with the help of virtual function.

Runtime polymorphism is called as dynamic polymorphism. it is the ability of a program to decide which function to call during execution rather than at compile time.

Virtual Function -

A virtual function is a member function declared with the keyword virtual in the base class. and redefined in the derived class.

It allows the compiler to perform dynamic binding meaning the function call is resolved at run time.

Rules of Virtual Function :-

- The virtual function must be member of the same class.
- They cannot be static members.
- They are accessed by using object pointers.

- A virtual function can be a friend of another class.
- A virtual function in a base class must be defined even though it may not be used.
- The prototypes of the base class virsion of a virtual function & all the derived class version must be identical.
- We cannot have virtual constructors, but we can have virtual destructors.
- while a base pointers can point to any type of the derived object, the revers is not true. When a base pointers points to a derived class, incrementing or decrementing it will not make it to point to the next object. of the derived class.

If a virtual function is defined in the base class it need not be necessary redefined in the derived class.

virtual function ensure that the correct function is called for an object, regardless of the type of reference used for the function call.

- They are mainly used to achive runtime polymorphism.
- Functions are declared with a virtual keywords in a base class.
- The resolving of a function call is done at run time.

Syntax:-

```
class Base
{ public:
    virtual void function_name()
    {
        // Base class implementation;
    }
};
```

For Example

```
#include<iostream.h>
class base
{ public:
   virtual void display()
  { cout<<" Base class display <<endl;
  }
};
```

```
class derived : public Base
{ public:
    void display(){
    cout<<"Derived class display "<<endl;
}
};
 int main()
{  base   *b_ptr;
   derived dobj;
   b_ptr = &dobj;
   b_ptr -> display() // calling virtual function
   return 0;
}
```

pure virtual function:-

- This function can be declared with 0 and has no implementation in the base class. Any concrete derived class must override it.
- A class with at least one pure virtual function is an abstract class.
- A pure virtual function in c++ is a virtual function that has no implementation in the ~~best~~ base class and must be overridden by derived classes.

Syntax:-

```
class Base
{ public:
    virtual void display() =0;
};
```

The 0 makes display() a pure virtual function.

Example:-

```
#include<iostream.h>
using namespace std;
class shape{
 public:
   virtual void draw() =0;
};
class Circle : public shape{
  public:
    void draw(){
     cout<<" Drawing circle ="<<end;
  };
```

```
int main()
{   circle c;
    c.draw();
   return 0;
}
```

- A class containing at least one pure virtual function is called as abstract class.
- Object of an abstract class cannot be created.
- Derived classes must override all pure virtual functions otherwise they also become abstract.
- pure virtual functions are used to define interface that derived classes must implement.

2-Marks

1) State function overloading. (W-25)
2) write a Process to create and initiative Pointer. (W-25)
3) Define virtual base class with suitable e.g. (S-25)
4) Define pure virtual function with syntax. (S-25)
5) Explain in brief abstract base class with example. (W-24)
6) State any two rules of operator overloading. (W-24)

4-Marks

1) Explain virtual base class with suitable example (W-24)
2) Explain virtual function with suitable example (W-24)
3) write a Program to create a class product with data-members Price & Product ID. Accept & display data for one object using Pointer to object. (6 marks)
4) State the four rules for virtual function
5) Explain runtime Polymorphism with suitable example.
6) write a program to overload the '—' unary operator to negate the value. 6 Marks.
7) Differentiate between compile-time and run time Polymorphism with examples.
8) Explain the concept of virtual functions with suitable example.
9) write a Program to demonstrate unary operator overloading using member function
10) write a program to demonstrate the use of this Pointer.

6-Marks

1) write a Program to declare class book containing data member as title outhor name, Publication Price Accept & display the information for one object using Pointer to that object.
2) Explain operator overloading, write a Program to overload binary + + operator.