OBJECT ORIENTED PROGRAMMING

PAGE NO
DATE

Books

1) E Balaguruswamy "Object Oriented Programming with C++" (4th edition)
2) Bjarne stroustrup, "The C++ Programming Language" (3rd ed.)

Program - set of instructions.
Software - set of programs.

Software crisis-

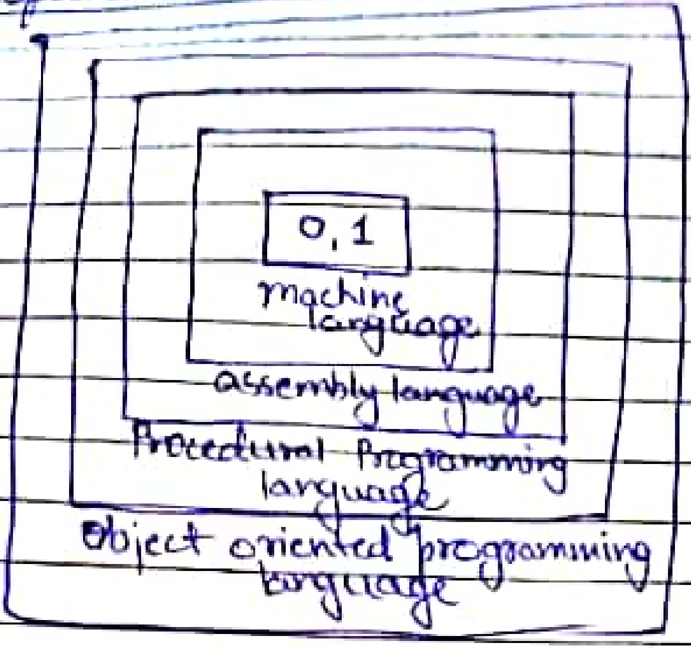
- How to represent real time problem in system design. e.g.- billing, e-commerce, product list, etc.
- To design system with open interface.
- How to ensure reusability and extensibility.
- How to develop the system that can tolerate any changes in future.
- How to increase s/w productivity and decrease cost.
- How to manage time schedule.
- How to industrialise s/w products.

Software quality measures

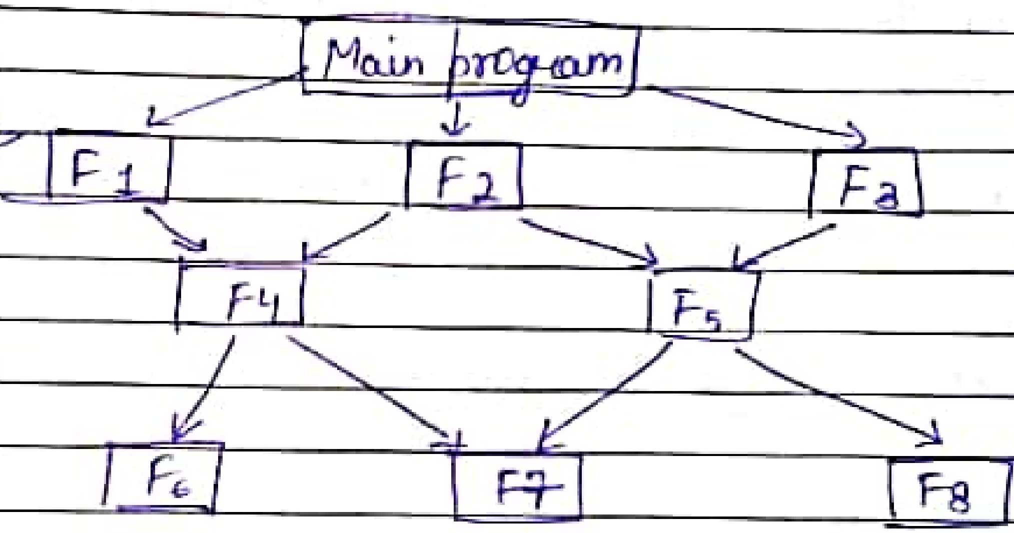
- Correctness
- Maintainability - errors/bugs can easily be ~~find~~ found out & then corrected.
- Reusability
- Openess and interoperability
  contact b/w different modules/functions.
- Portability - can run on any hardware
- Security - breaching should be difficult or impossible.
- Integrity - all modules should be integrated easily / integration of software.

- User-friendliness:

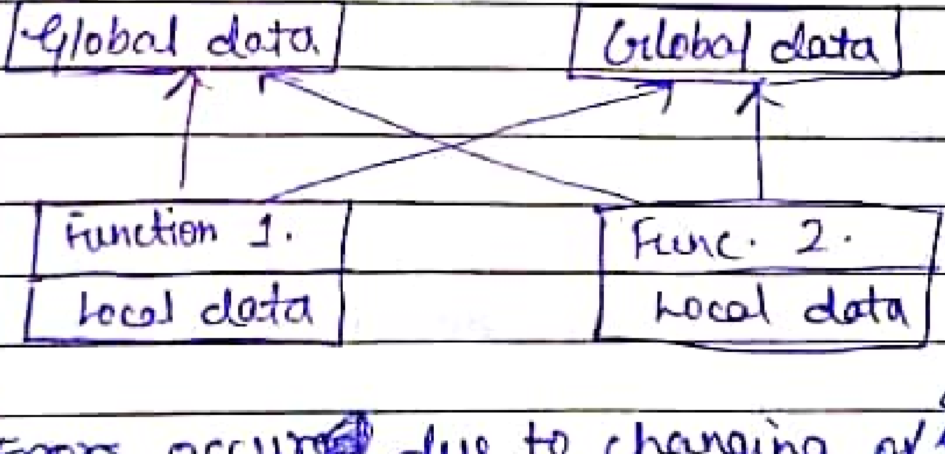
Software Evolution


0,1
machine language
assembly language
Procedural Programming language
Object oriented programming language

In procedural language



* Drawbacks of POP -



- Errors occur due to changing of global data used by diff. functions.

2. Real world issues to solve them.

1. Diff. to identify errors if the programs have so many func using same variable (Global [illegible]
2. The POP does not handle the real world problem very well.
3. Difficult to create new data type.

Drawback of procedural approach – C, Basic, Pascal, FORTRAN

Some OOP features:

1. Emphasis is on data rather than procedure or function
2. Programs – divided in objects.
3. Data structure are designed such that they characterize the objects.
4. Functions & operators on the data of an object are tied together in data structure.
5. Object may communicate through functions.
7. New data & functions can be easily added whenever necessary.
8. It follows bottom – up approach.

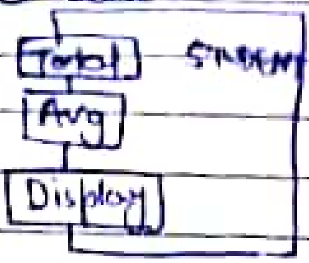
Basic Concepts of OOP

1. Objects:
2. Classes
3. Data abstraction.
4. Inheritance
5. Polymorphism
6. Dynamic Binding
7. Message Passing, etc

Object: They are the basic run-time entity in an object oriented system. It may represent a person, a bank accnt, a thing. Any physical or logical unit having specific characteristics which match to the

real world are called object.
Each object contains data and code.
Object : STUDENT
DATA . Name , DOB , Marks
FUNC : Total , Avg . , Display

or



* Class : Collection of similar object.
Ram, Sita, Hari are monitors of class- (Person)

Example:
```
class person
{ char name [30];
    int age;
    get data ( );
    display ( );
  };
int main ( )
{ person p;
   p. get data ( );
   p. display ( );
}
```

* Data Abstraction and Encapsulation

→ The wrapping up of data and function in a single unit (called class) is known as encapsulation.

→ The data is not accessible to the outside world only those functions wrapped inside them can access it.

→ Insulation of data from direct access by the program is called data hiding or information hiding.

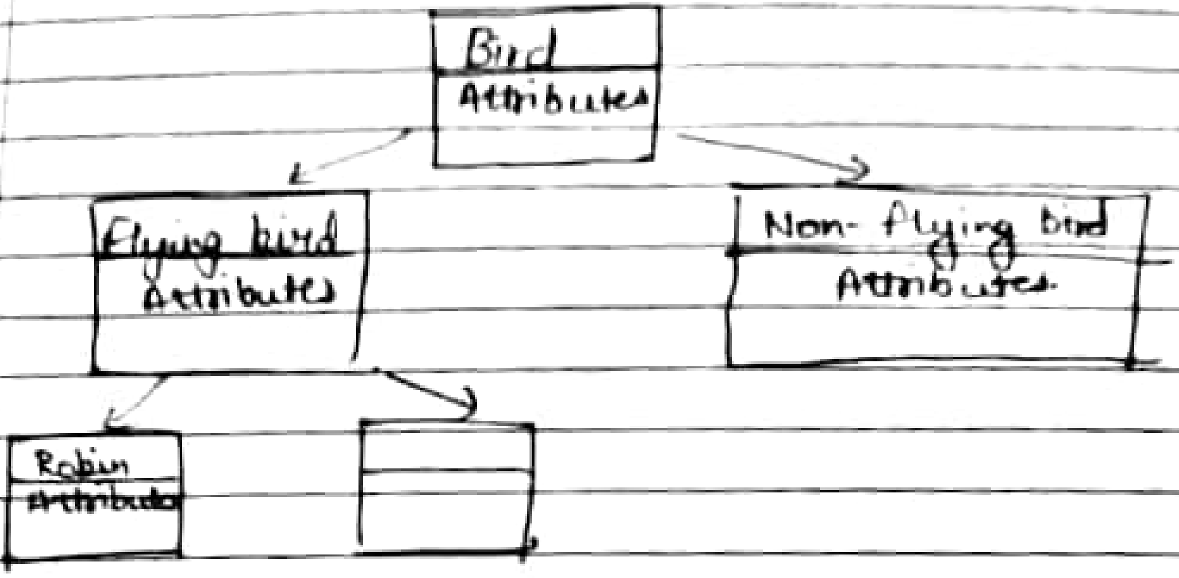
→ Abstraction refers to the act of representing essential features without including the background details or explanation.

→ Since classes use the concept of data abstraction they are known as abstract data type (ADT)

# Inheritance

→ It is the process by which objects of one class acquires the properties of object of another class

→ Ex Bird Robin is a part of class flying bird
& flying bird is a " " " bird

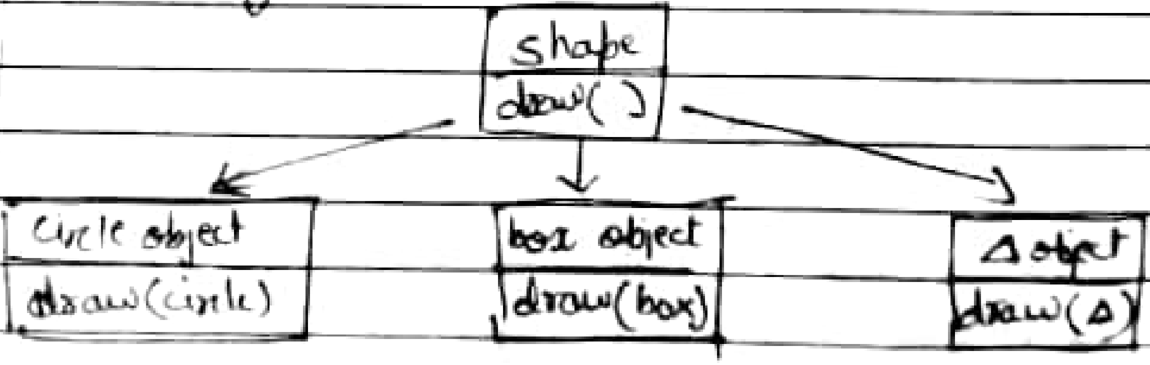


Polymorphism Ability to take more than one forms

a = 2 } a+b = 5
b = 3 }

a = "clear" } a+b= clearsky
b= "sky" }

The process of making an operator to exhibit diff. behaviours in different instance is known as operator overloading.



* Structure of C++

Include files

↓

Class declaration

↓

member function definition

↓

Main function program

Ques WAP in C++ ~~which~~ to calculate the billing ~~cost of~~ amt ~~product~~ & display the product name, no of product and price product and bill amt.

9|8|17

Creating source file

C++ → .C, .c, .cc, .CPP and .CXX

Turbo C++ & Borland → .C for C & .CPP for C++

Zortech C++ → .Cxx

UNIX AT&T → .C and .cc.

Tokens: The smallest individual unit in a program is known as tokens.

Keywords, Identifiers, Constant, strings, Operators

Identifiers and constants

variables, array, class, etc.

int 3abc; ✗ int A; ✓

abc ✓ a; ✓

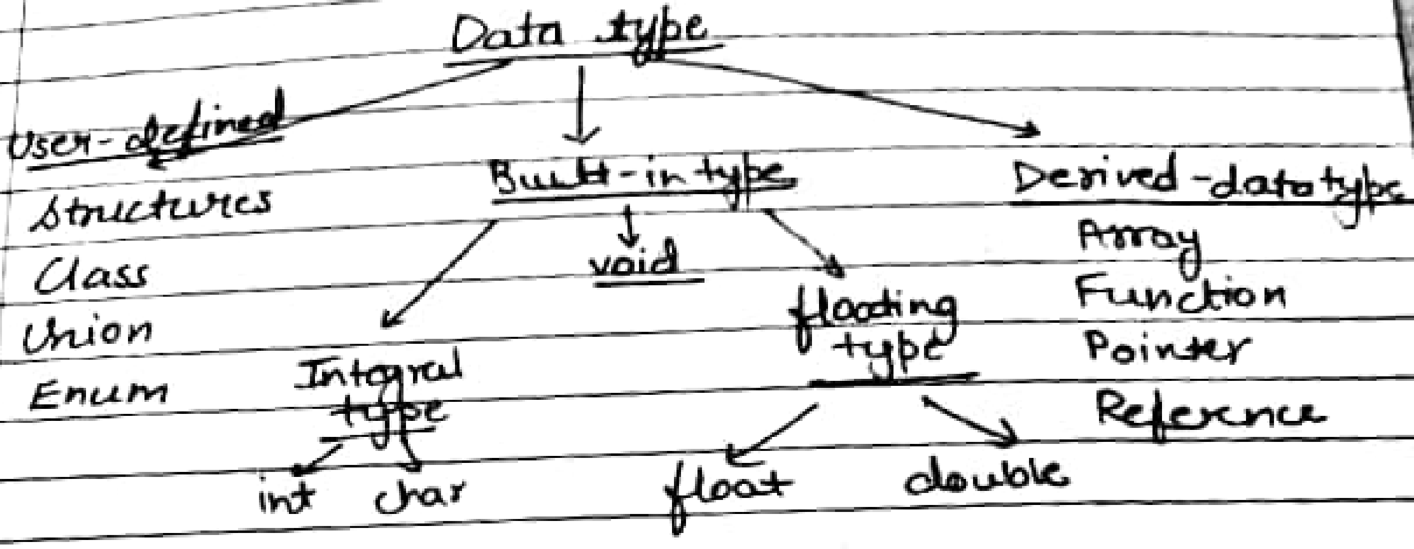
abc1 ✓

char ✗

```
[illegible] t ;   cout << [illegible] ;
int * ptr ;
t = (int *) ptr ;
```

const refers to fixed value that do not change during the execution

Like C, C++ supports literal constants like integers, characters, floating pts, string etc.



void : 1) to specify the return type of a function when it is not returning any value.

2) to indicate an empty argument list to a function

void func(void)

3) void is used for declaring generic pointer.

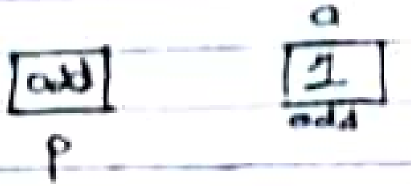
Ex. void * gp - can be assigned a pt. value of basic data type.

```
int * ip
gp = ip; ✓
ip = gp; ×
```

```
void * ptr1 ;
char * ptr2;
ptr1 = ptr2; ✓
ptr2 = ptr1 ; ×
ptr2 = (char *) ptr1;
```

Pointer

```
int a ;
int * p ;
p = &a
```



```
int const *ptr = & m
```

Here constant of m cannot be changed.

Declaration of a variable

C++ allows the of declaration of a variable any where in scope.

```
int main()
{ float x ;
   float sum = 0;
   for (int i=1 , i<5 , i++)
    {   cin >> x ;
        sum = sum + x ;
    }
   float avg ;
   avg = sum/i-1
   cout << avg ;
   return 0;
}
```

Dynamic initilization

```
float avg;
 avg = sum /(i-1);
```

float avg = sum/(i-1);

declaration & initilization at same time.

Reference variable

Provides a alias (alternative name) for a previously defined variable, data type & reference-variable-name

```
float total=100;
float & sum= total;
 cout<< total;
 cout << sum;
  total +=10;
⇒ both 110
 sum=0 ⇒ both 0
```

```
int x [10];
 int & y= x[14];
```

```
void f (int & x)
 {  x = x +10; }
int main ()
{ int m =10;
 f (m)
 cout << m;
}
```

Operators in C++ :

<< - insertion operator

>> - extraction operator

:: - scope resolution operator

Functions in C++

```
void show (); ⟶ func. decl
main ()
{ ----
  ----
  show (); ⟵ func. call
  ----
}
void show () ⟶ func. defn.
{ ----
  ----
}
```

Func. prototype - Prototype describes the fn interface to the compiler by giving details such as:

- no. of arg.
- type of arg.
- type of return value.

type func-name (argument list)

Ex. float vol (int x, int y, int z); ✓
float volume (int x, int y, z); ✗
" " (int, int, int); ✓

Default Assignments

C++ allows us to call a func without specifying all its arguments.

float amnt (float p, int t, int r=0.15)
int = amt (5000, 7)

float amt (float p=1000; int t; float r=0.15)

Destructor

Characteristics

destroys the va

1. Destructor functions are invoked automatically when the objects are destroyed.
2. Destructors cannot be overloaded.
3. If a class has destructors. each object of a class will be deinitialised before the object goes out of scope.
4. It obeys usual access rules. Private & protected accessed only by a member & friend functions when will accessed by all.
5. It returns no value and no arguments can be provided to destructor.
6. It cannot be inherited.
7. It may not be static.
8. It is not possible for a destructor to take the address
9. Member function may be called from within the destructor.
10. An object of a class with destructor cannot be member of a union.
11. It is required because when the pointer to object go out of scope a destructor is not called implicitly.

```
    matrix :: ~matrix ()
    { for (int i=0; i<d1; i++)
       delete p[i];
       delete p;
    }
```

A constructor is called in the following situations.

1. When a new operator is used during dynamic initialisation

2. When the associated type is used in a definition

```
void func()
{ Test f ; // Constructor called to allocate f
}
```

3. When a callby value is used to pass an argument to a func

```
void func ( Test f)
{ - - - - ;
   - - - ;
}
func (object); // Constructor called to make a copy
                  of object.
```

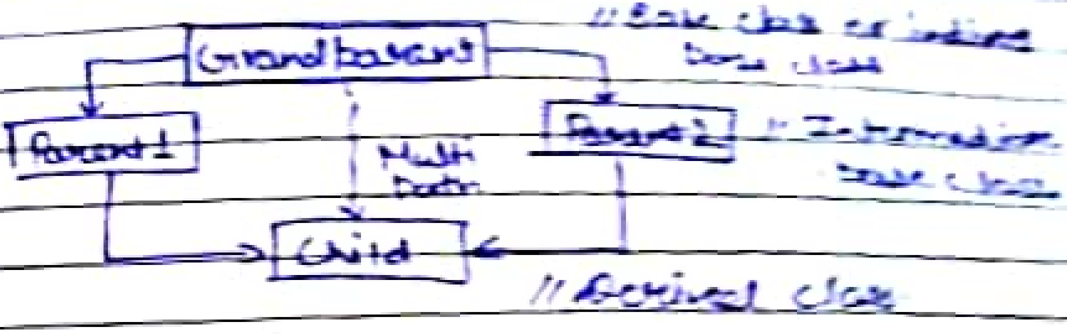
4 When the return type of a function must create value of the associated type

```
Test func()
{ return myobj ; // constructor called to
}                   make a copy of myobj.
```

Virtual Base Classes

Code reusability
↓
Inheritance

Code redundancy
↓
Problem


Grandparent
// Base class or indirect base class
Parent1
Parent2
// Intermediate base class
Multi path
Child
// Derived class

- → All the public & protected members of grandparent are inherited into child twice via parent1 & parent2
- → Child have duplicate sets of members inherited from grandparent.
- → This introduces ambiguity. and hence must be avoided by inheriting the indirect base class grandparent as virtual base class for parent1 & parent2
- → The duplication of inherited members due to multiple paths can be avoided by making the common base class as virtual base class

```
    class Grandparent {
                      ____
                     ____
                 };
    Class Parent1 : virtual public Grandparent
              { ___
              };
    Class Parent2 : public virtual Grandparent
              { ___   // keywords virtual & public may
              };           be used in any order.
    class child : public parent1 , public parent2
            { ___   // only one copy of grandparent
            }             will be inherited.
```

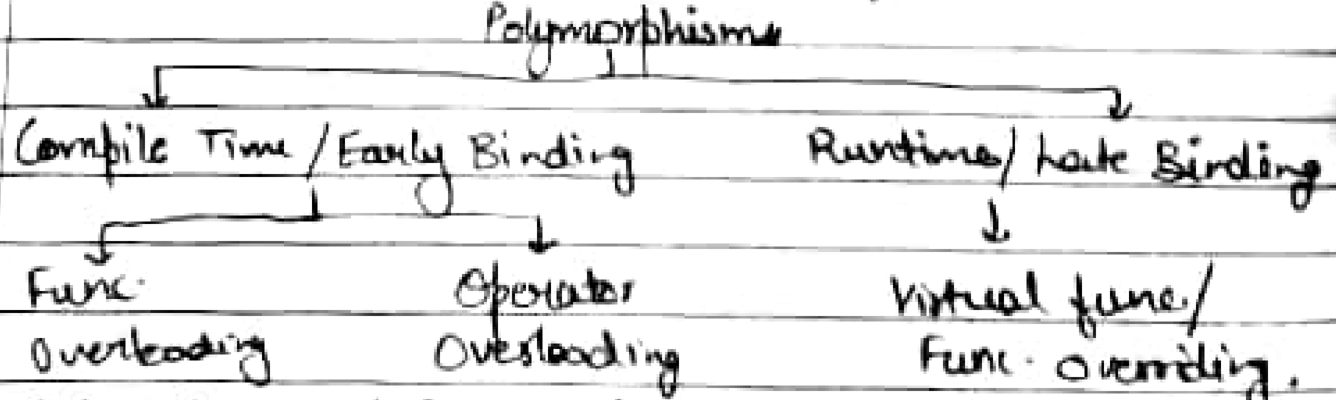
Abstract Class

1. Abstract classes are not used to create objects
2. An abstract class is defined only to be inherited by other classes, i.e. only to act as a base class.
3. It is the designed concept in program development & provide a base upon which other classes may be built.

Virtual func.

1. A virtual func. is a member func. that is declar as virtual within a base class & redefined by a derived class.
2. To create virtual func., proceed the base vision of func. declarations with the keyword virtual
3. When a class containing virtual func. is inherited the derived class can redefine (override) the virtual func. to suit its own unique needs.
4. The method named and type signature should be sa for both base and derived version of func.

- Hence polymorphism at run time (late binding) is also achieved using virtual functions.



5. Super keyword is a reference variable used to refer parent class object in JAVA and C#.

6 Pure Virtual Functions :

	Overloading	Overriding
Defn:	Methods having same name but each must have diff. no. of parameters or parameters having diff. types & order.	Subclass have method with Same name & exactly the Same no. and type of parameters & same return type As superclass method in JA
Meaning:	More than 1 method shares the same name in the class but having diff signatures.	Method of base class is redefined in the derived class having same signature
Behaviour:	To add/Extend more to methods behaviour.	To change existing behaviour of method.
Polymorphism:	Compile time	Run-time
Inheritance	Not required	Always required
Method Signature)	Must have diff. signature	Must have same signature.

Pure Virtual Func.

1. It is a virtual func. in the base class for whi there exist no implementation in the base class
2. They are only declared inside the base class
- Since objects of abs classes cannot be created
3. It is declared virtual inside the base class & redefined it in the derived classes. It serves only as a place holder
4. Such func are ~~only~~ also called as do-nothing func

```
class vehicle
{ private: datatype d1,
           data type d2;  ()
  public : virtual void spec = 0; // pure virtual func
};                        spec () = 0;
class LMV : public vehicle          declares that it
{ public : void spec()               is a pvf.
     { // LMV defn of spec func}
};
class HMV : public vehicle
{ public : void spec ( )
     { // HMV defn of spec func}
};
```

Constructor In Derived Classes

Method of Inheritance	Order of execution
1. Class B : public A { };	A ( ) : Base constructor B ( ) : Derived constructor
2. class A : public B, public C { };	B ( ) : Base (first) C ( ) : Base (second) A ( ) : Derived
3. Class A : public B, virtual public C { };	C ( ) : Virtual Base B ( ) : Ordinary base A ( ) : Derived

Initialisation list in the constructor fn is the method of initialising class objects.

```
Class XYZ { int a;
            int b;
            public :
            XYZ (int i, int j) : a(i), b(2*j) { }
          };
main ()
{ XYZ z (2,3); //Here a will be initialised to
}                 a to 2 & b to 6.
```

Member Classes Nesting of classes

#
```
class alpha {...};
class beta {...}
class gamma { alpha a ,   // a is the object of alpha class
              beta  b;   // b is the object of beta class
            }
```

→ An object of gamma will contain the objects a and b. This type of relationship is known as containership / nesting

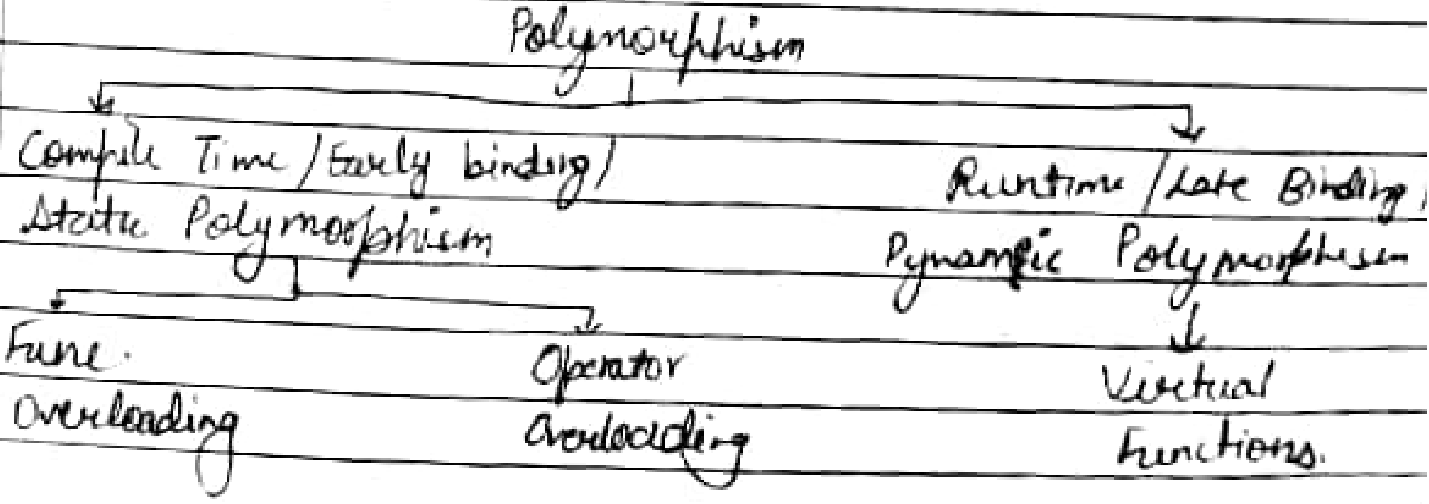
→ Nesting object is created int two stages.

a) Member objects are created using their respective constructors

b) Then the other members are created.

→ Constructor of all the member objects should be called before its own constructor body is executed)

Pointers, Virtual Functions & Polymorphism



Polymorphism

Compile Time / Early binding / Static Polymorphism

Func. Overloading

Operator Overloading

Runtime / Late Binding / Dynamic Polymorphism

Virtual functions.

Binding – For every func. call, compiler binds or links the call to one func. def^n.

Early binding	Late binding
→ At the time of compiling program	→ At run time
→ Eg- func overloading, the decision of binding during control func is taken by considering formal arguments of the func, their datatype and their sequence	→ The compilers decides the type of object at runtime then binds the func. called to a func. def. Eg - virtual func
→ It is achieved with the formal arguments, datatypes and their sequence	→ It is achieved using pointers

POINTERS

→ A ptr is a derived data type that refers to another data variable by storing the variable's memory address rather than the data.

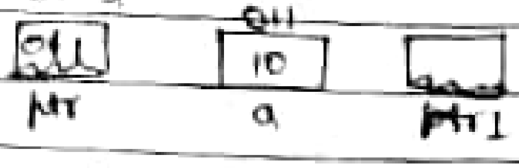
→ A ptr variable defines where to get the value of a specific data variable instead of defining actual data

→ A ptr may also refer to another ptr.

→ Often ptr refer to a data variable

→ ptr provide an alternative approach to access another data object

```
int *ptr, a, //declaration
 ptr = &a ; // initialization of ptr variable containing the
  a = 10;     address of variable a.
int *ptr1 = a;
```



// & also called reference operator is used to retrieve the address of a variable

** This pointer

1. The keyword this is used to represent an object that invokes a member f^n

2 This is a pt that pts to the object for which this f^n was called.

5 The unique ptr is automatically passed to a member f^n when it is called.

4. The pointer this acts as an implicit arguments to all member func

• This ptr is used implicitly when overloading the operators using member function

```
class ABC
{ int a;
  - - -
  - - -
};
```

The private variable 'a' can be directly used inside the member f^n as;

```
a = 123;
(OR)
this → a = 123;  // Does the same job.
```

return this;
// inside a member f^n will return the object that

→ A ptr can pt to an object created ~~by~~ a class
~~item x; // class item & x object of class item.~~
~~item *it ptr; // A ptr itptr of type item.~~

~~class item~~
~~{ int code;~~
~~float price;~~
~~public: void getdata~~
~~(int a, float b)~~
~~{ code = a~~

object ptr are used to creating object at run time. They are also used to access the public members of an object.