Smalltalk — OOP — Allan Kay is inventor
Purely OOP — 1970s

OOP is a methodology to design a program using class and objects.

Data types: Data type specifies different sizes and values that can be stored in variable.

(i) Primitive/primary :=> built in or predefined.

- integer — int
- character — char
- Boolean — bool
- Floating point — float
- Double — double
- void
- Wide character

(ii) Derived data type: Derived from primitive data type

- Function
- Array
- pointer (java have no pointer)
- Reference

(iii) user defined (Abstract): Defined by user itself

- class
- structure
- union
- Enumeration
- Typedef

Modifiers — Modifier are used in built in data type to modify length of data

- signed (+/-)
- unsigned (+)
- short (lower the length)
- long (higher the length)

(*) (*) If else if else, loops from Java-01 notebook

Function :=> Function is a group of statement that together perform a task.

- Every c++ program has at least one function that is main().

Defination —

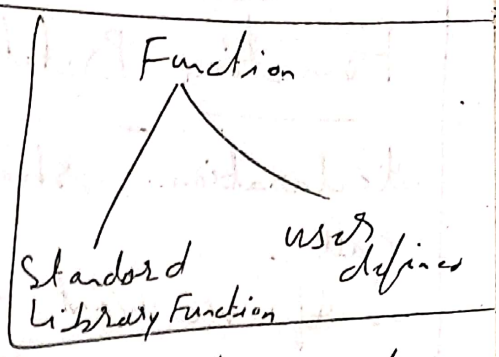
return_type function_name (parameter list)
{
body
}

Formal parameter

eg int max (int num1, int num2)

Calling a function:

to call a function you have to just write the function name with Parameters (if applicable). Else if the function return any value you have to store it in other var.



like result = max(10,20); actual parameter

- The parameter passed to the function is called actual parameter. (10,20)
- The parameter received by the function are called formal parameter (num1, num2)

Pass by Value :⇒ Value of actual parameter are copied to formal parameter. By default C++ use it.

Pass by Reference :⇒ Both actual & formal parameter refer to the same memory location.

Function Prototype :=> In C++ function declaration should be done before its call.

If we want to define a function after call then we must use function Prototype

```
Void add(int, int);
```

Function overloading :=> More than one function with same name but with different functionality.

- The return type different
- The number of paramter different
- The type of paramter different.

```
class Add
{
  public:
    int add(int a, int b)
    { return a+b;
    }
    int add(int a, int b, int c)
    { return a+b+c;
    }
};  // end of class
int main()
{
```

```
Add obj;
int res1, res2;
res1 = obj.add(2,3);
res2 = obj.add(2,3,4);
cout << res1 << " " << res2 << endl;
return 0;
}
```

Op - 5 9

- It is also example of compile time Polymorphism.

(*) pointer, Structure, array, String ... (note)

Strings :

Cin extraction always considered tabs, whitespaces, newline as terminating value

To get entire line we use getline ()

⇓

getline (cin, stringname)

---

break - it leaves loop even the condition for its end is not fulfilled.

Continue - Cause the program to skip the rest of the loop in current iteration.

---

inline function : insert definition in the code itself for short function instead of calling it.

- calling a function generally cause certain overhead (stacking arguments, jump)

⇒ inline string concatenate (const string a, const string b)
{ return a+b;
}

Function template :-

generalize the return type, data type

C++ has the ability to define function with generic types known as function template.

Define

keyword

placeholder

given by us

```
template <class SomeType>
SomeType Sum(SomeType a, SomeType b)
{
    return a+b;
}
```

call

```
X = Sum<int>(10,20);
```

Scope of Variable :=>

outside any block is global space. The name is valid anywhere in the code.

inside a block/function is local scope. valid only in that specific block or function.

ex

```
int a; // global var
int Sum(int a, int b)
{
    int c; // local var of Sum
    c = a+b;
    return c;
}
```

* There can not be two var with same name in same scope.

global var is visible on all over the program
But printf give priority to local var over global var

Namespace :-> A declarative region that provide a scope to the identifiers inside it.

- the name of function, variables, types etc.

- They are used to organize code into logical group and to prevent name collision that can occur especially when the code base include multiple libraries.

std::cout — it says cout is in the scope of std.

Suppose there are many cout in different libraries. so we explicitely said compiler to use cout of std,

std :: — Scope resolution operator
name of namespace

Ex — Syntax to declare namespace

```
namespace name
{
    named entities
};
```

Using :- introduce a name into current declarative region.

Using namespace std ;

↓

Here in all over program we use std as namespace.

Typedef :- typedef used to give a different name by which a type can be identified

Syntax - typedef existingtype newtypename;

ex- typedef int integer;

after that we can use

integer a = 5;

↑ it means int

We can also use

using ~~existingtype~~ =

newtype = existtype;

ex -

using integer = int;

- used to reduced the length of long type names -

Class → a group of similar entities.
collection of objects.

- It has no memory until object is created.

Object :=> An object can be defined as an instance of a class.
There can be multiple instances of a class in a program.
- It takes up some space in memory.

ex - if expensive car is a class then its object are Mercedis, BMW, Toyota etc.
Their attributes are price, speed, color.

- class is a group of objects which have common properties. It is a templates / blueprint from which objects are created.
It is a logical entity.

A class contain -
(a) Fields (c) Constructors
(b) Methods (d) Blocks
(e) Nested class & interface.

Syntax - class classname
{
field.
method.
}; // end of class

Access Modifier

in C++ :⇒

public → available to everyone

private → available only to that class

protected → available to that class and the class that derived from it

When we dont maintain any modifier then by default it is in private (C++)

In Java there is one more & rest all same

default → available in same package

In java if no modifier is maintioned that it is in default modifier.

Class Constructor :⇒

function default return type is int

a block of code that initialize ready created object.

- It does not have a return type
- name same as class name.

- It can never static (it is a instance member function, means it can be accessed by objects)

```
class @Hello
{
    string name;
    Hello() // — constructor
    {
        thisname = "Arijit";
        cout << "Hello constructor";
    }
};  — end of class
```

* it is implicitly invoked when an object is created.
* we use it to initialize the data members (variables)

```
class Complex
{
private:
    int a, b;
public:
    Complex()
    {
        cout << "Hello constructor";
    }
};
```

```
void main()
{
    Complex c1;
}
```

o/p — Hello constructor

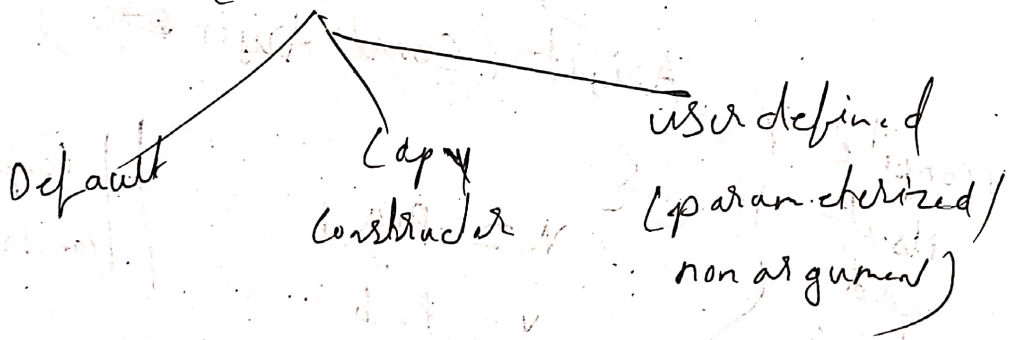
- Constructor makes object a object by initializing the values of variable
(insan ko bolta insan bano)

Those values which need to initialize at the time of object creation, ~~need~~ is initialized by constructor.

- Constructor must be placed in public section of class
- We can also define constructor outside the class.

Ⓧ if there is no constructor then compiler implicitly add default constructor & copy constructor.

if there is any type of constructor added in program then compiler doesn't add default constructor. (So we can't create any object without specifying parameter Arijit a; ✗)



Default ⇒ implicitly added by compiler, it is added in the object code file.

Parameterized ⇒ When we pass value to initialize variables/data in the object

Copy Constructor :⇒ When we pass a already created object as a parameter in the argument, we pass reference of object.

```
class Arijit
{
    private: int x,y;
    public:
        Arijit()
        {
        }
        Arijit(int a, int b)
        {
            x=a;
            y=b;
        }
        Arijit(const Arijit &*p)
        {
            x=p->a;
            y=p->b;
        }
};
void main()
{
    Arijit o1,o2(3,4);
    Arijit o3=o2; //Copy Constructor
```

if we itself write it then it is no argu- constructor

default that is added by compiler itself

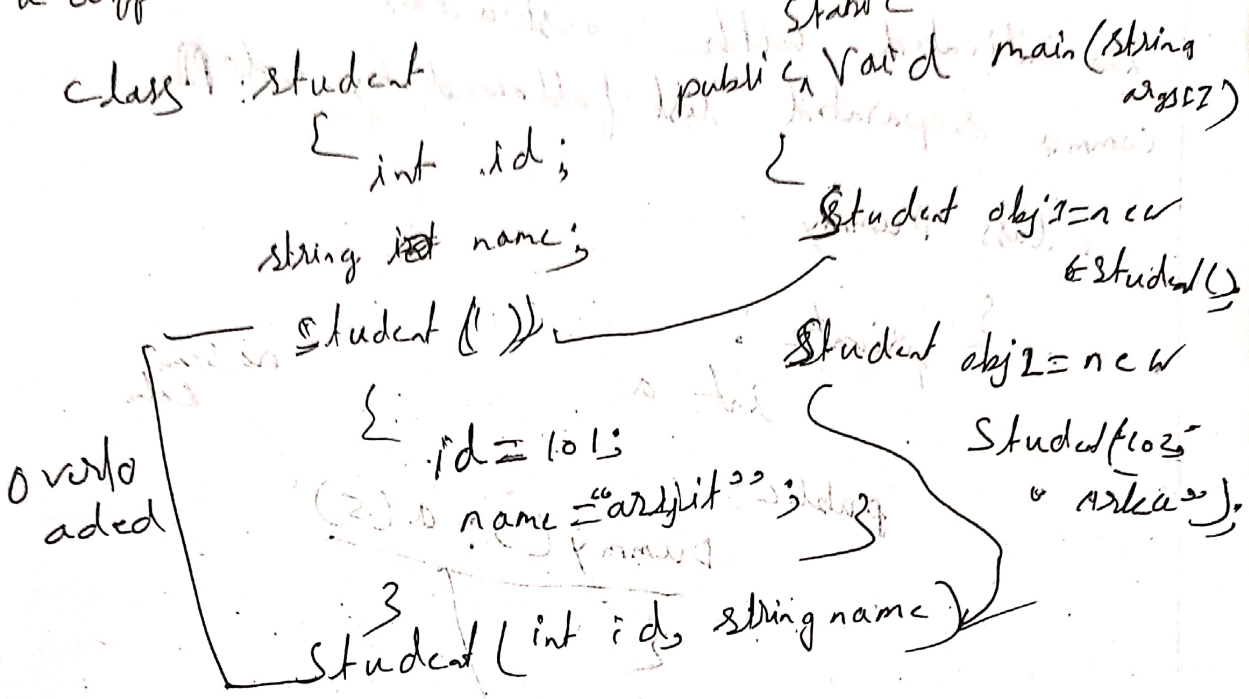
parameterized constructor

compile itself add if we not write

copy constructor

Constructor overloading :=>

if we have more than one constructor with different parameter list / type in a such way that each constructor perform a different task.



Every constructor should differ in their parameter list / type.

Member initialization in constructor :=>

. initializer list is used to initialize data members of a class.

. The list of members to be initialized is indicated with constructor as a comma separated list followed by colon

```
class Dummy
{ private:
      int a
public:
      Dummy() : a(5)
      {
      }
};
```

no semi colon

(*) a=5; //this is assignment

this is initialization

Both do same

it is mandatory in some case like

(i) const member
(ii) Reference member } both need value in initialization

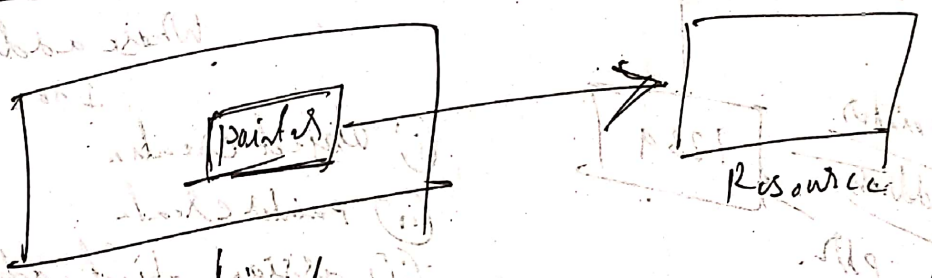
. They dont take assign value

in both case we have to use initializer list.

Destructor

- instance member function (accessed only by object)
- name same as class but preceded by tilde (~) symbol.
- never static
- has no return type
- takes no argument (so no overloading possible)

⊛ It is invoked implicitly when object is going to destroy (just before destroy) to release resources allocated an object

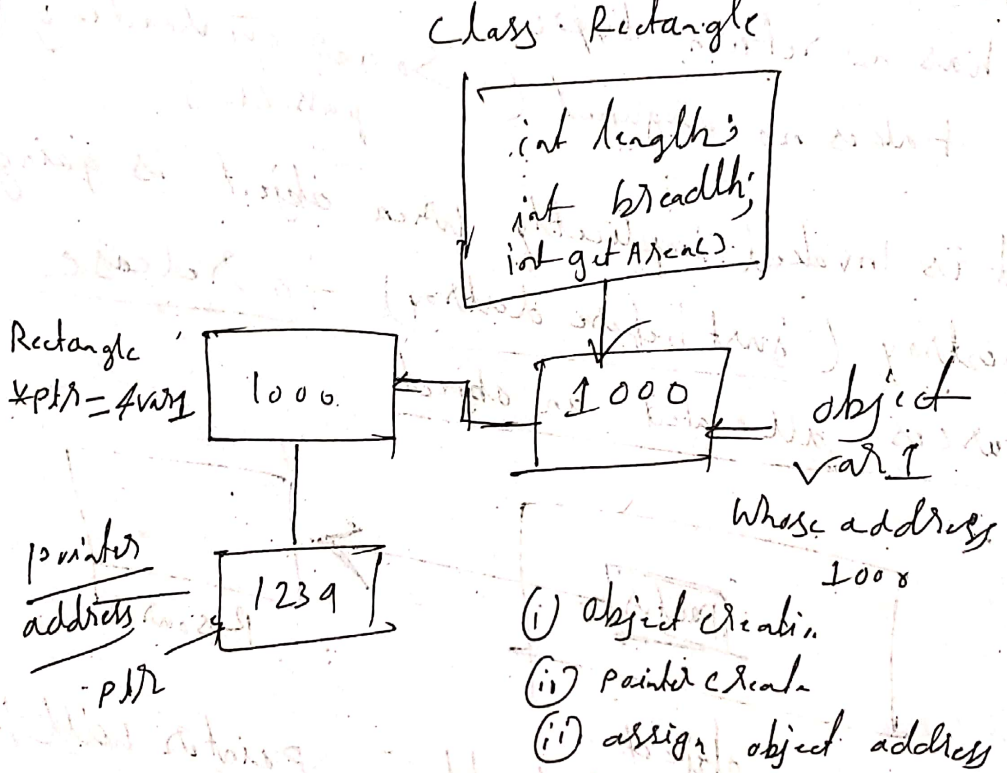


if object destroyed then pointer will be destructed.

Pointer to class :=>

Pointer are variable that store address of variables.

A class pointer is a pointer variable that store address of a object.



We can access value by using (→) member access operator

pointer to object

Same just

Rectangle *ptr = new Rectangle ();

instead of create object before we create in time of declaring pointer.

Operator overloading :⇒ To assign more than one operation on an same operator.

- We have to write a special function known as operator

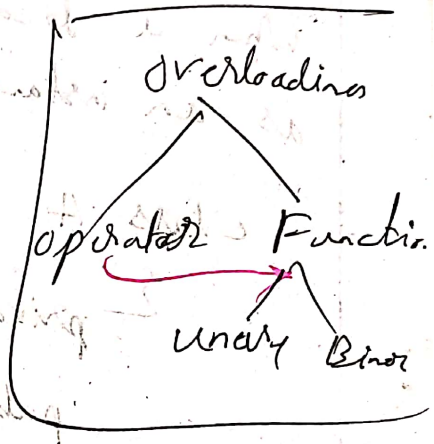
Syntax-

```
return type operator operator sign (arg list)
{
  // body
}
```

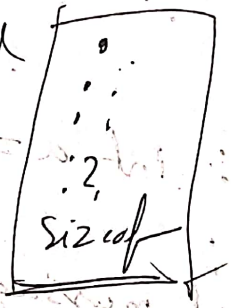
There is two way to define this

(i) class function

(ii) Friend Function



(x) We can't overload



Two types-

(i) unary

(ii) Binary

```
obj1.operator+(obj2)
=
obj1+obj2
```

This keyword ⇒

- Access the currently executing object of a class
- Access the data members of the currently executing object.
- Calling the member function associated with the currently executing object.
- To resolve the shadowing issue

When a local variable has a same name as an instance variable.

```
class A
{
    private: int a = 10;
    public:
        void message()
        {
            cout << this->a;
        }
};
void setdata (int a, int b)
{
    this->a = a;
    this->b = b;
}
```

naming conflict.

Static Members :=>

Static keyword is used in

- Data member
  - inside a class
  - outside a class (static variable)
- Member function

it gets memory at the very first before object creation.

Static data members : (class variable)

if any variable is declared static then

- only one single copy exist in all objects
- it is always initialized as zero after the class declaration (no additional memory allocation)
- It retains value (always take updated value)
- if no object created then also we can access static variable

classname :: staticvar_name;

Static member function :=> member function with static keyword.

- It can access only static data members
- It is also accessible without object creation

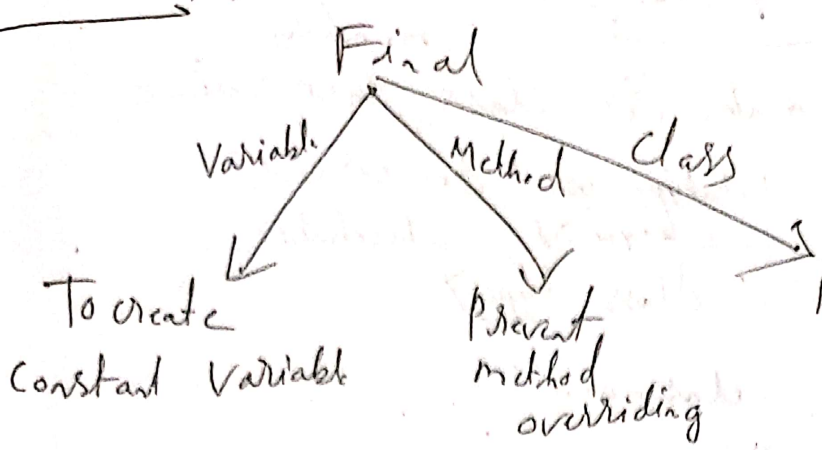
classname :: staticfunctionname

```
class A
{
    int a;
    static int b;
    public:
    A(int x, int y)
    { a=x; b=y; }
    void disp()
    { cout<<a<<b; }
    static void show()
    { cout<<b; // a is not accessible
    }
};
int A::b=20; // initialization.
```

Data type

```
void main()
{
    A obj(10,20);
    A obj2(100,200);
    obj.disp(); //10,200
    A::show(); //200
}
```

Final :=>



Constant Member function :=>

Constant variable :

- can not be left uninitialized at the time of creation

```
const int PI=3.14;
```

```
const int PI;
PI=5;
```

- It cant get value from anywhere.

Constant Method :=>

- it does not / can not change any data member (foo.x=20 // error)
- It can only call const functions
- It can only read data member. (cout<<foo.x)
- Constructor will be called always to initialize data
- Const member function can only be called. (int get() const { , })

Class Template :->

members

Way to make our class generalize as far as datatype is concerned

keyword placeholder

```
template < class type >
class classname
{
  .
  .
  .
};
```

ex

```
template < class T >
class Arijit
{
  T a, b;
public:
  Arijit ( T first, T second )
  { a = first;
    b = second;
  }
  T getmax ();
};
template < class T >
T Arijit < T > :: getmax ()
{
  T max;
  max = a > b ? a : b;
  return max;
}
```

placeholder

function template = class Template

return type

```
int main()
{
    Arijit <int> object(100, 75)
    cout << object.getx();
    return 0;
}
```

Template Specialization =>

Define different implementation for a template when specific type is passed as template argument.

Suppose we have a class that increment a value by 1 and have only one function increase but if the value is character then it is difficult to generalize. So we use template specialization.

ex
```
// template class
template <class T>
class Arijit
{
    T element;
public: Arijit(T arg)
    { element = arg; }
    T increase()
    { return ++element; }
};
```

```
// template class specia
template <>
class Arijit <char>
{
    char element
public:
    -
    -
    -
};
```

Friend Function :=>

- a function which is not member function but can access private/protected member
- is declared in the class but defined outside of class
- can become friend to more than one class

Syntax Keyword

friend returntype functionname (class obj);

// declare function definination does not need any scope resolution.

ex —

```
#include<iostream>
using namespace std
class A
    { int a,b; // by default Private.
      public: void setdata (int a, int b)
              {
              this.a = a ;
              this.b = b ;
              }
        friend void add (A ob);
    };
void add (A ob)
    { int c;
      c = ob.a + ob.b;
      cout << c;
    }
```

```
int main()
{
    A obj(1,2);   // create object
    add(obj);     // calling friend function
```

- We give object as parameter as friend function is not a member function. So if we simply call friend function then it can't directly access data member (as it has garbage value)

Uses — (i) in operator overloading

Java does not support friend

- friend function can be declared in both private + public section
- it can not be called with object: it will be called directly like function
- It can't access member name directly. It need objectname and dot operator
- A friend function in a class can be a member function of other class

```
void firstclass::func()
{
}
```

Friend class :=>

A friend class can access both private and protected members of the class in which they are declared as friend.

```
class A
{
    int x;
    friend class B;
};

class B
{
public:
    void display(A &a)
    {
        cout << a.x;
    }
};

int main()
{
    A a;
    B b;
    b.display(a);
}
```

Inheritance :=>

derive / child class

- A mechanism in which one class inherit the properties of other class.

parent class / base class

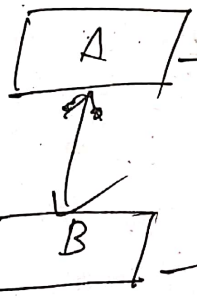
Benifits - • For code Reusibility
• For method overriding

Syntax :=> class child classname : accessmodifier parentclassname.
{
}

ex- class dog : public animal
{
}

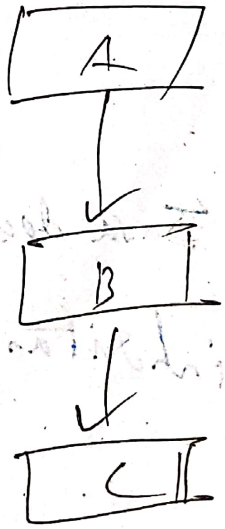
Types —

(i) single inheritance :=>



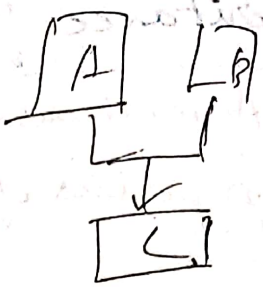
class B : public A { }

(ii) multilevel



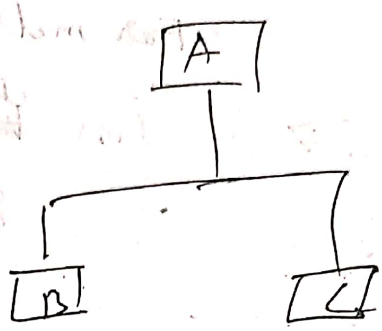
class A { };
class B : public A { };
class C : public B { };

(iii) Multiple



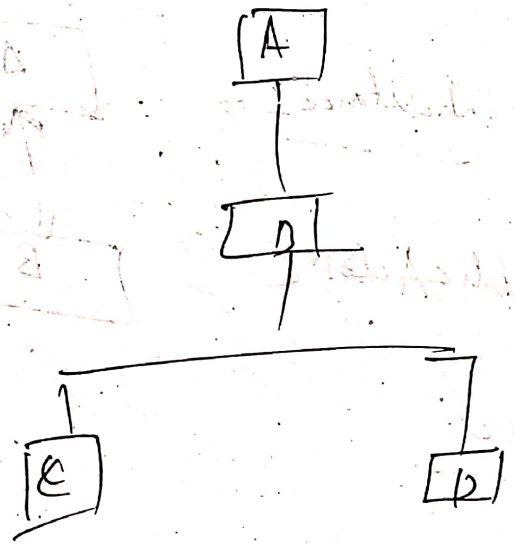

```
class A { };
class B { };
class C : public A, public B
{ };
```

(iv) Hierarchical




```
class A { };
class B : public A { };
class C : public A { };
```

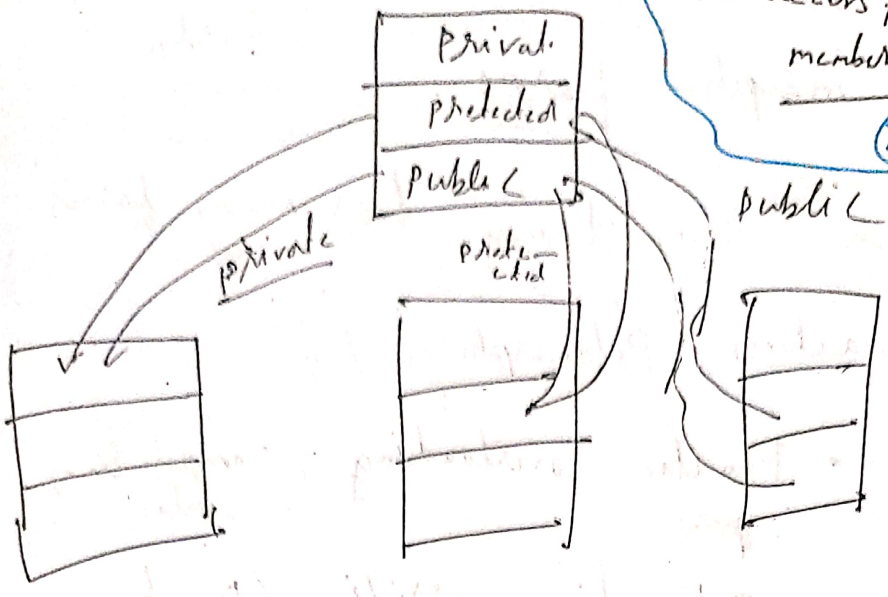
(v) Hybrid — mixture of more than one.



(x) Java does not support multiple inheritance.

Visibility mode :=>

creating object only can access public members



Visibility mode determines that How object of derived class treat the members of parent class

if visibility mode private then both protected & public member of parent class will be treated as private and can not accessed by its derived class object

| is a Relationship always use public

(X) derived class does not inherit
(i) Constructor and its destructor
(ii) friends
(iii) private members

(X) When we create object of base class constructor or destructor also called automatically of parent.

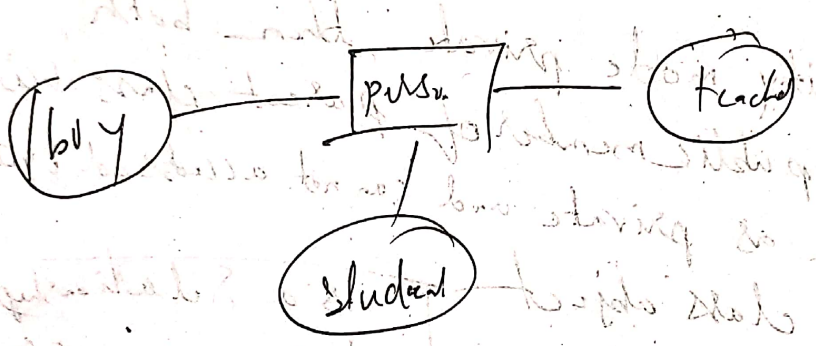
Polymorphism :⇒

Poly- many

morph- forms.

types
- compile tim.
- Runtim.

- When one thing has many forms we achieve Polymorphism by
  - Function overloading (compile time) static
  - Function overriding (Runtime)
  - virtual functio.



Method overriding :⇒

if sub/derive class has same method with same signature as declared in the parent class then it is called method overriding.

if derived class give specific implementation of a method then it happens

- We can't override static method as it is not a member function of a class.

~~child class method is called~~

(X) if we create object of child class and call the override function then function of child class will be executed.

Super key word is used to call overridden method in child class.

```
class car
{ public:
    void shiftGear() { } // overridden method.
    void f2() { }
};
class sportscar : public car
{ void shiftGear() { } // Method overriding
  void f2(int x) { } // Method hiding as argument different
};
void main()
{ sportscar obj;
  obj.shiftgear();
  obj.f2(); // error
}
```

(*) The problem is method overriding is if

```
Car *ptr;
sportscar obj;
ptr = &obj;
```

As we can make a ~~object~~ pointer (*ptr) which can point to the object of any of its descendants class.

So we can assign address of child class in parent class pointer.

```
ptr -> shiftgear();  // if we use virtual then derived will called.
```

according to early binding (compile time) it will bind that method which is ptr. As ptr is of class car then it will bind to the function of parent class. This is the problem we want child class function.

This problem is solved by virtual function.

by using virtual keyword we are saying compiler to bind this at run time

Virtual Function =>

- a member function declared in base (parent class and is redefined in child classes
- we use virtual keyword only in base class. (automatically child class function als. get it)

Abstract base class =>

A function will be do nothing if there is no body inside or outside class.

```
void func()=0;
```

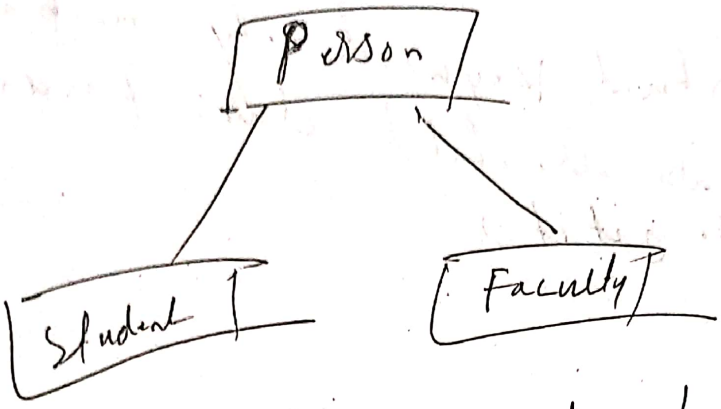
if this function is in any class then we cant create object as it can try to access fun which is illegal.

```
virtual void fun()=0;
```

by using virtual we are saying that the class cannot make any object as well as need redefination of function in child class.

(*) if their is any pure virtual member then this class is abstract class

- Abstract class that can only be used as base class and have atleast one pure virtual member function (virtual member function without definition)



~~Person object should not be~~ any object should not created in person.

- In Java we use abstract keyword.

(*) Java does not support all type of inheritance (multiple inheritance X)

(*) OOP can be used without using any header file

(*) Encaptulation & abstraction → Similar feature

(*) This pointer — allows an object to call data & method of itself. This is like Recursion

(*) classes are pass by Reference & Structure are pass by copy. Does not depend on Program

(*) int a;
public: float a; } — same var name can not be used in single class.

(*) Template class = generic class

(*) classes does not have any size at all.

(*) Abstract class have member function with no implementation & inheriting sub class have to implement these function must.

(*) Scope of nested class depends on access specifier & inheritance used.

(*) class defination must be ended with ; (Semi colon) not : colon ✗

(X) Instance can not be created in Abstract class object as it has no constructor but has destructor.

(*) You can create any number of class as well as object

(X) class student{ }s; ✓
class student{ }s[5]; ✗ no size
class student{ }; student s[5]; ✓ object array

(*) Function can return object if return type same.

(*) When an object is returned a temporary object is created to take value as will be destroyed after return.

(*) Student S[2] = {S(100,2), S(200,4)};
object array initialize

(*) if we use class then we are using Encapsulation.

(*) 7 basic Feature of OOP
Encap, inheri, Polymorphi, Abstraction
objectmust, Message Passing, Dynamic binding

(*) Exception handling is a feature of OOP.

(*) OOP provide better Security always

(*) Language Support class ✓ but not Polymorph ✗
↳ object based language (Ada) example

(*) Function with highest Priority in compiler is called first in case of abstract class & function overloading.

(*) << can be overloaded
||, &&, += can't be overloaded

(*) Encapsulation helps in writting immutable class in java
↑ making all member private

- The data prone to change in near future is encapsulated so that it does not get changed accidentally.
- Using access specifier Encapsulation is achieved.

⊛ Global variable always violate encapsulation as it is accessed in everywhere.

⊛ If we use pointer then also encapsulation violated, as data can change.

⊛ encapsulation ensured data security to some extent but not full (as global was, pointer may violate).

⊛ Hiding the implementation complexity can make programming easy. (We can use printf without worrying)

⊛ Class is a logical abstraction (Provide logical structure for all of its object.

⊛ object is real abstraction

- Abstraction gives idealized interface
  - can apply to both data & control
  - ~~object~~
  - Object ≡ abstraction of data + code.
    (use data members) (inbuilt class)
  - Use abstraction to avoid duplication
  - Abstraction has 3 level
    - Logical
    - Physical
    - View

- Constructor should always public
- If there is default value parameterized constructor then an object without passing argument can be created with default value.
- While creating child class object, Parent class constructor will be called first

(*) class A { }
class B { }
class C : public A, public B { };

obj of C created then A will first called

(*) If object is passed by value, there will be infinite loop. out of memory error.

- Explicit call to constructor create a temporary instance
- Student S3 = 100; ✓ Object create with arg as 100.

- Constructor → Default, Parameterized, copy
- Explicit keyword can be used to define constructor so that it can be called only by object. constructor name. not implicitly.

(*) Default constructor can be called explicitly

(*) static constructor initialize static member. it is called only one time.

(*) → can not be a Parameterized constructor

it initialize default value of constructor called before first object creation.

- Default constructor initialize all data members 0 & null

- Copy constructor is used when we pass object to a function. It just copy object value
- Copy constructor also used when compiler generate temporary object to copy value
- Copy constructor can be private (dynamic mem allocation)
- Argument to copy constructor must be Const (or it will change values)

- Constructor overloading is used to initialize object with different way.

(X) destructor call is reverse of constructor call (first child then parent)
- it has no argument
- it can be implicit / Explicit
- cant be overloaded ~ classname()
- no return type

- As abstract class does not have any constructor so there is no destructor as well.
- Destructor should be virtual so that in case of inheritance.
- When pointer are involved, user defined destructor should be there

- all the constructor should private inorder to not create object
- Private members are never inherited

(X)
```
class A
{
  private : int marks
  public : A (int x=100)   // default val
          { marks = x
          }
};
A obj1; // — default val
A obj2 (200); ✓
```

(X) Private member can not be overridden as it can not inherited in derived class.

- In protected access of constructor, object can be created inside the subclasses,

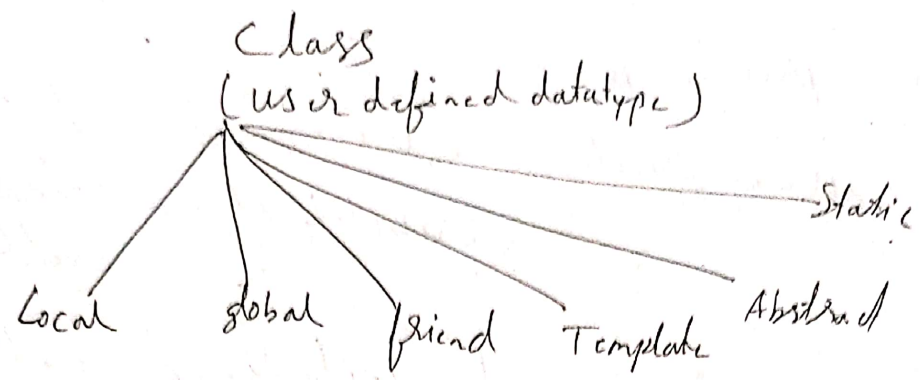
```
class B : protected A()
{
};
B b;
b disp(); // - can access as its protected member
```

- class A : public B
  - Same access modifiers of B
- class A : protected B
  - public will be protected rest same
- class A : private B
  - all will private.
- We can't reduce visibility of inherited methods

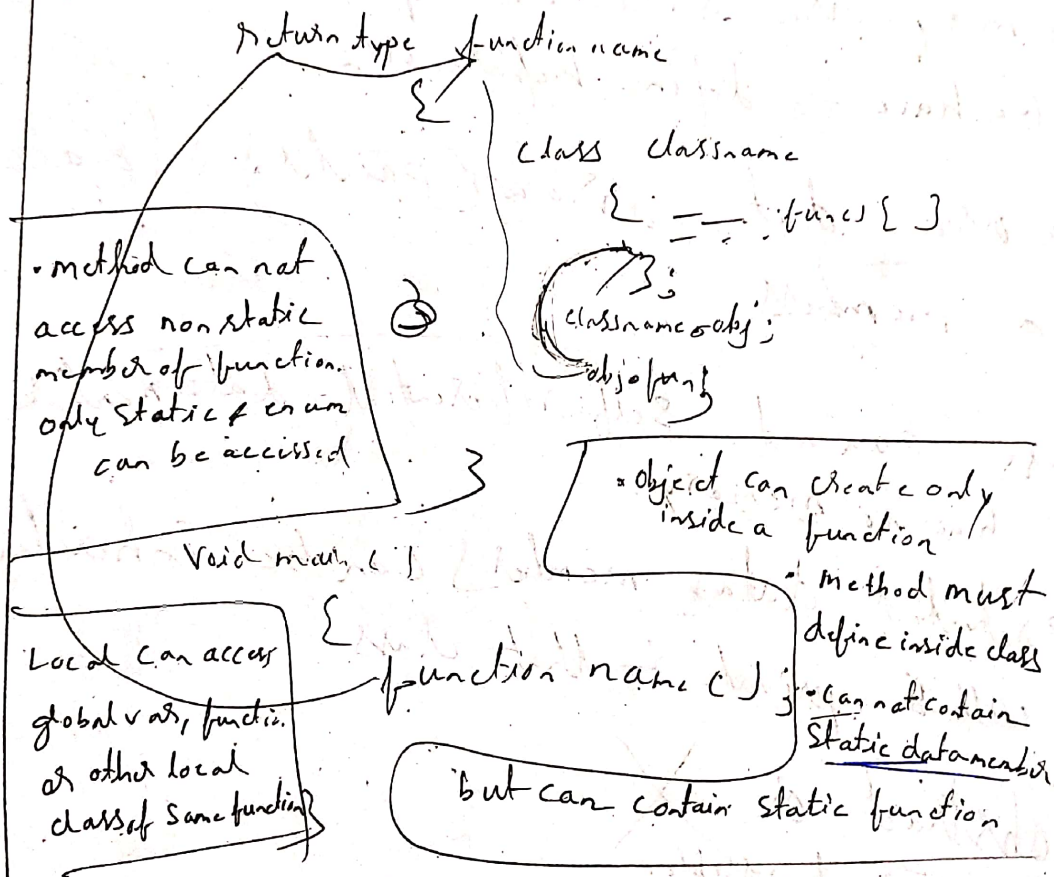
- Data member can be initialized only by constructor (as it is property of object not class)
- We may use structure type in class but first we have to define before use.
- We use dot, arrow (pointer) to access data members

- class cannot self referential - data member as it has no memory.
- protected data members can be inherited but will be private to that class
- abstract d ✗ — Not possible
- 5 Type of member function
  - Simple
  - Static
  - const
  - inline
  - friend

• Member function of a generic class are automatically generic



Local: A class which is declared inside a block or Function



Global :- declared outside of all the functions or block.

- The local class have this feature to access the static & extern variable of the function in which it is declared

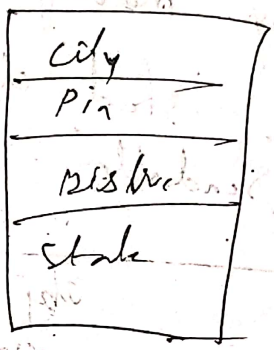
* All member function of local class is are inline by default.

* other function can not access other local class

Nested class

- class inside a class is nested class

class student



Address is associated with only student object so it should be inside student class

Address class

Ex) class student

{

private: int roll

char name[20];

class Address

{ int house no;

char city[20];

public

void setaddress

{

};

Address addr;

- nested class access is same as other member.
  if inner class private we cant access outside class
  if public we can access outside class.
- Accessibility rules will be same.

Nested class

- static (only static members)
- non static (can access all members of outer class)

- nested class can be declared with any modifier.
- nested class make code efficient & readable.

---

object passed to function

- pass by value
- pass by pointer (Explicitly address pass)
- pass by Reference (implicitly address pass)

- only 1 object can be returned
  as many as object can be passed.

- memory allocated of object in RAM
- When object constructor call, memory allocated
- new is type safe (it does not require to know type of data)
- malloc / calloc — dynamic memory allocation
- When object goes out of scope, memory allocated of object gets free (reference of address is delete)
- C++ — delete is used to free the memory

- classname arrname[size] — object.
~~(cannot be init~~ will not be initialized individually.

→ class must have default constructor to initialize all objects of array.

(X) objects in an object array, can be created without default constructor with the help of parameterized constructor.

(X) Object array created in Heap

(X) array of characters ≠ string
(it can be changed) (it always remain same)

(X) Function object = object with a single function

factory object = create new object

(Y) Every object must have unique name

Abstract class

* at least one pure virtual function
= abstract class

(X) main function can be there in abstract class

(X) an abstract method = abstract class

- Abstract class A
  - inheritance
  - class B — either it have to be abstract or it have to implement the methods (virtual)
- we cant create object but
- class libraries is an important use of abstract class.
- cant create object of abstract class but we use pointer reference (Runtime poly)

- The abstract class in java can implement constructors (though instance can't create but during constructor chaining, constructor can be called)
- It is not mandatory to have an abstract class to have abstract method.

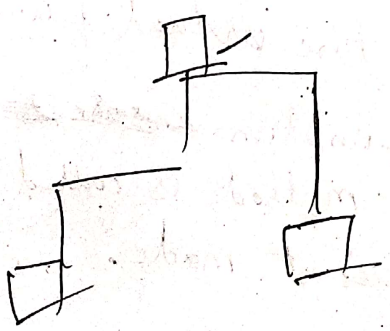
- Template class can have more than one generic data type

Template <class T1, class T2>

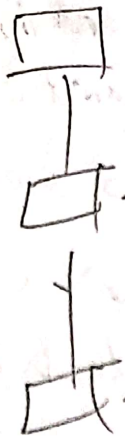
- Explicit class specialization, template <> class myclass <int>
- Template class defines the form of a class without full specification on the data.

(X) For every new type there is an instance of every new type will have their own static instance.

(X)



hierarchical inheritance



parent class
— High abstraction

* Multiple inheritance — diamond problem
Hybrid
- Default inheritance modifier is private

(X) Diamond problem :⇒ Methods with same name create ambiguity & conflict

---

Virtual function :⇒ (Run time Polymorphism)
- it should be public access specifier.
- must be member of same class
- They can not be static (are property of individual objects)
- They are accessed by using object pointer
- It can be friend by other function
- It is not mandatory for derived class to override the virtual function.
- (X) virtual function ~~call is called~~ ensured that correct method is called regardless of reference it made.

* Prototype must be same in base & derived class

* A class may have virtual destructor but not virtual constructor (constructor can not be overridden.

* overriding virtual function in derived class, we may write virtual or compiler can make it virtual implicitly.

* virtual function must be defined in base class but overridden in child not mandatory

* Abstract function

(Dynamic/run polymorphism)

* Abstract method must be declared in base class & definition provided in derived class. (must)

* Compiler check whether all derived class define the abstract method or not.

* Though compiler check the definations but definations are resolved at run time. This is called dynamic polymorphism.

public abstract fun ();

* Abstract method must not be static.

---

* Void fun const ( ) { }

const member function

does not allow changes in the function in the class

* Operator can be overloaded either by member function or friend function.

* In operator overloading, left operand is passed implicitly. other operand (can be called directly) are passed as function arguments.
  * this object

* =/[subscript]/→ operator can be overloaded only with member function (not friend function)

* <<, >> must be overloaded with friend function only.

* We can use overriden method of base class with the help of classname :: (Scope resolution operator) methodname.

* C# does not support method overriding. We use virtual or override keywords. non virtual or static method can't be overridden.

---

* Const member function does not change value of calling object.
  * Non const function can be called only from non const object

* A function can have both const & non const option version.

- const

Java ⇒ private private members
~~C++~~
C#

C++ → private : private members

(X) Access private member of a class using address of member function

(X) main function never be private
always public

(X)
```
class A
{
    class B
    {
        public void show()
        {
        }
    }
}
```

else we can access inside enclosing class using nested class objectpointer / to access we can create object of enclosing class in allover the program

(X) A derive class Object can access public method of base class → False
as if private/protected inheritance is used then it cant access

(X) Static data member must be defined outside class. (as its common to all object, should be created only once)

- Const Member are static by default (as const wont change object to object)
- object then will use dot / arrow } operator
  object pointer -> ?
  to access Method

(*) Static Member can not be virtual (as virtual need redefination but Static will always same)

- Static can't be overloaded

(*) String are immutable (can not changed)

(*) String name = "Abhijit"
length = length of string excluding null character

(*) char charAt(index) - particular index character

(*) Substring takes two parameter
Abhijit -> Substring(1,4)
starting from first index to last.

(*) if only one parameter pass then it will print from that position - last.

(*) toupper = Convert whole string to upper case
compareTo = compare String objects

trim( ) — Remove white space from both the ends

Replace() — takes two parameter (index + changed char)

* indexOf("i") ⇒ 2

Also indexOf("i", 2) ⇒ 4
2nd occurrence

New operator

- It allocate memory for an object and return particular pointer
- Add new vtable slot call
- If new fails either throw an exception or return 0.
- First Memory is allocated then constructor is called.
- new does not need any type name (const, volatile etc)
- new operator — initializer
- Objects allocated using new are not destroyed even it go out of scope

Delete

- Deallocate block of memory
- if object created using new operator it should be deleted using delete operator
- It does not return value. (void)
- object destructor call before deallocation

delete
- 2 type
  - object deletion
    - delete objectname;
    - delete [] objectname;
  - object array deletion

Automatic var

(a) Local variable (inside any block or function)

- allocated & deallocated automatically
- Static are not automatic variable (one copy) (after end call destroyed)

```
void add
{int c; auto
c=5;
}
```

- auto var ⊗ default value garbage
- It is not initialized implicitly.

- Uninitialized auto variable contain undefined / garbage value.

~~Extern~~

- Definition = allocation of Memory of variable it can be done only once
- In extern source file must be included in new file.
- Only one header file contain the declaration of variables that are extern.
- (X) Function are extern by default.
- (X)
- (X) Constructor with default argument

```
A (int x = 0)
{ marks = x }

A obj1
A obj2 (20)
```

- (X) default argument is written in last in argument list

```
A (int x, int y, int z = 0)
```

because

```
A obj1 (2,3)
```