(C++ structure can have fn too.)

opp: ~~by default public~~ can be made private

(Backward compatibility of C++ : whatever C supports, C++ will also support)

Date: / / Page no:

⑨ Polymorphism :-

Polymorphism is a greek term means the ability to take ~~many~~ more than one forms

(i) An operation may exhibit different behaviour in different instances.

(ii) The behaviour depends upon the type of data used in the operations.

one name = many forms

at different instance perform differently.

[+ operator] → int a + int b = int c

→ complex a + complex b = complex c
No! No!

diff arg. → Operator overloading

diff. Type of arg. → Function Overloading → func()
→ func(int)
→ ~~func(char)~~
→ func(int, int)

[one name, diff. call.]

⑩ Why Class when we have Structures ?

① struct complex
{ float x; float y; } c1, c2, c3;
c3 = c1 + c2 ; // Not allowed ✗

To give more feel of inbuilt data type ~~use~~, created class

structures does not support arithmetic operations of struct variables

② Do not permit data variables.

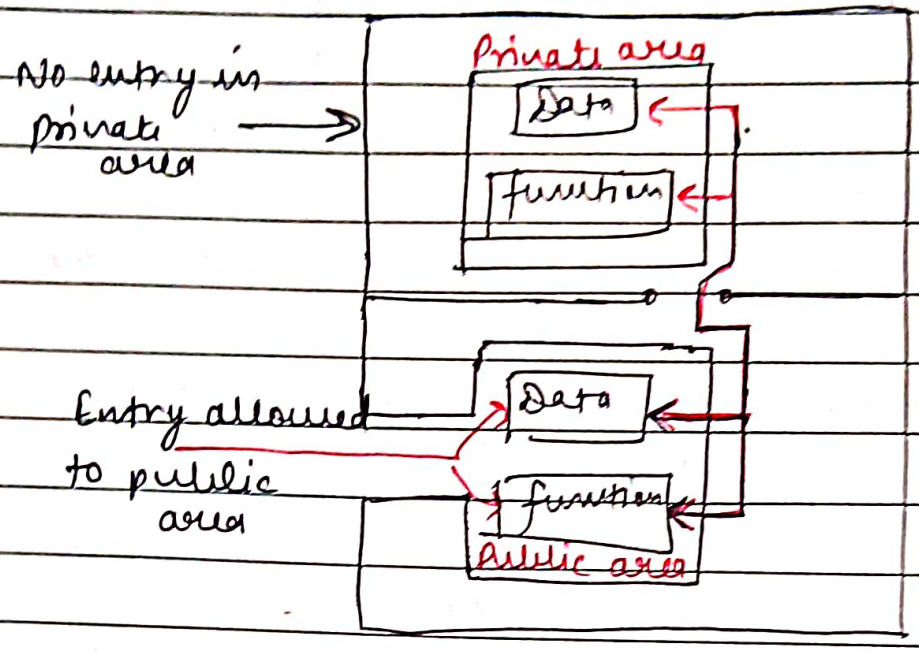
(i) So C++ / OOPs try to bring user defined data types as close as possible to the builtin data types.

(ii) By default member of the class are Private while member of structure are public.

06/

Date / / Page no.

Class



Class Private & Public

```
class item.
{ int number;          // private.
  float cost; void display();   ] // By default private
                                  ∴ we don't use private keyword
  public: int c;
      void Getdata (int a, float b);
      void Putdata ();
  };
```

In General we keep data part as private and member fns as public but not necessary.

Class item
Data number cost, name
function putdata() get data display

07.03.20
Wednesday

→ While object declaration in C++, we do not need to use "struct keyword"

Date: / / Page no:

For creating memory we have to declare object of a class.

define → does not create memory
declare → to create memory

creating object → item x

Comma separated multiple objects can be created
item x, y, z;

Accessing Class Members :-

① member function can be invoked only by using an object

Getdata (10, 15.7); ✗
object is necessary // Cant accessed directly

② Private Data member can be accessed by member function not by object directly

x.number = 10; ✗ private

x.Getdata (75, 105.7) ✓
↳ public

③ Public Data member can be accessed

Defining the member function :-

We can define member function in 2 ways
① Inside the Class (defination)
② Outside the Class (definations)

② Outside the Class → Class

void item :: getdata (int a, float y) {
number = a
cost = b
}

return type — void; scope — item; scope resolution — ::

Date: / / Page no:

Scope

function of class item of return type void.

Outside the Class

scope resolution operator

returntype Class :: func-name (arg1, arg2);

① Inside the Class :

as usually, No need to define scope like put data ()

NOTES :-

① several different classes can use the same function name. The membership label will resolve their scope.

inside

```
class item {                    class purchase {
void getdata ();                void getdata ();
}                               }
```

outside

```
void item :: getdata ();        void purchase :: getdata ()
```

Same name

but clears the scope

② Member function can access the private data of the class.

A Non Member function can't do so.

③ A member fn can call another member fn without using dot (.) operator.

```
void item :: getdata () {       // x.getdata ();
:                               ⟹ directly we can do putdata()
putdata ();
}
```

Outside vs Inside

Date: / / Page no.:

(3) If f^n is inside function → then it will be treated as inline function.

Inline Function:- → depends on compiler afterwards
It is request to compiler
depends on compiler

(1) compiler replace the f^n call with the corresponding function code in the case of inline f^n

(2) We can make f^n inline by prefix the keyword "inline".

Big function will take memory
∴ inline function should be small as possible.
else it will be disadvantage rather than advantage.

Execution must no change in both.
do not use when too ~~too~~ there is loop, static var.

(*) Nesting of Member Functions

point no. (3) of previous

```
void item :: getdata () {
    — " —
    — " —
    putdata ();
}

int main () {
    n. getdata (2, 3);    // automatically put data
}.
```

```
class item {
    int a;
    int b;
public:
    putdata ();
}.
```

(*) Memory Allocation of for Objects

(Q) How member function's memory is allocated?

Member f^n are created and placed into the memory space only once when they are defined as a part of a class specifications.

06

Space for storing function statements

Date / / Page no:

Since all the objects belonging to that class use the same member functions, No separate space is allocated for member functions when the object are created.

{ At the time of defination → only memory space for member f^n will be created before creating object. }

Only space for member variables is created/ allocated separately for each object.

Separate memory allocations for the objects are essential because the member variables will hold different data values for different objects.

In Short

	Memory
functions	~~multi~~ once only
data	= No. of objects defined.

common for all objects

get data ()

put data ()

Memory is created when funct^n is defined

Object 1	Object 2	Object n
number	number	number
cost	cost	cost

Memory is created when Object defined.

Memory diagram after Object creation

13.09.2022
Tuesday

Date: / / Page no:

Q) Why only one for member function?

Reason :- Object, will go to memory location & run that code return ans (if required) but no change in code of f^n, ∴ only one it is done "Space efficient".

Private Member f^n :- A private member f^n can be called by another f^n that is the member of its class. Even an object can't invoke a private f^n using a (.) dot operator.

Static Data Members :- (i) It is similar to C static variables
(ii) It is initialized to zero (0) when first object of its class is created. No other initialization is permitted.
(iii) Only one copy of that member is created for entire class and is shared by all the objects of that class, no matter how many objects are created.
(iv) It is visible only within the class but its lifetime is the entire program.

static data member needs to declared outside of the class

```
class item
{   static int count;
    int number;
    public:
        void getdata(int a){
            number=a;
            count++;}
        void get count(){
            cout<<count;}
};
```

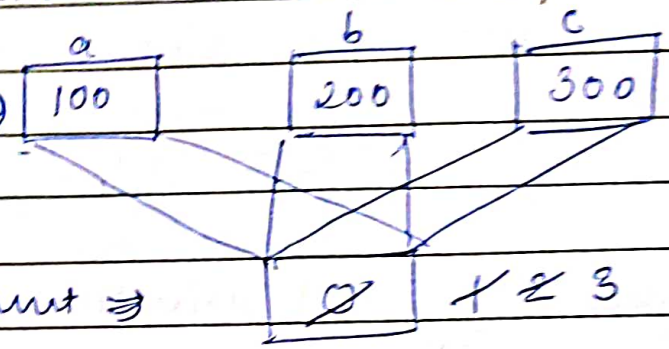
```
int item::count;
int main(){
int a,b,c;
a.getCount();   => 0
b.getCount();   => 0
c.getCount();   => 0
a.getdata(100);
b.getdata(200);
c.getdata(300)
a.get count();  => 3
b.get count();  => 3
c.get count();  => 3
```

Date / / Page no.

(v) Type and scope of each static ~~data~~ member outside the class defination.

(vi) Static data members are stored separately.

Memory diagram



Static Member fn :- (i) A static fn can have access to only other static member whether its fn ~~and~~ or variables declared in same class. No need to declare it outside the class with data

(ii) A static member fn can be called using class name instead of object.

```
class-Name :: static-function (-) ;
```

get data can't be a static fn as it has 'number' which is not static

```
—n—                          —n—
—n—                          —n—
{                            :
item :: get count();  ⇒ 0    Static void get count()
item :: get count();  ⇒ 0      count
  :                   ⇒ 0    }   ↑
a. getdata(100);               Static
  :
item :: get count();  ⇒ 3
  :                   ⇒ 3
  :                   ⇒ 3
```

14.09.2022
Wednesday

Date / / Page no:

Array of objects :-

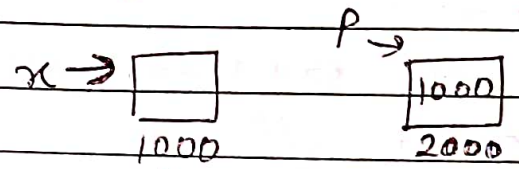
ClassName arr[10];

arr of objects which can hold 10 objects of class = Class.Name

Accessing data of 6th object. ⇒ arr[5].getdata(50);

Object as a function argument :-

int y;
int *p = &y;
int &x = y;



C++ ~~Call~~

Aliasing (True) "Evaluation Matrix"

int &x; // Error → but where it points

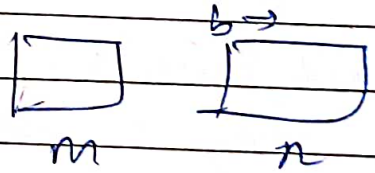
int *q; // Can be done

getting same name for different memory locations.

// Call By Reference

swap(int &a, int &b) {
int t; t = a;
a = b;
b = t; }

new name a →
new name b →



swap(m, n);

If variable is required by & then I have given ~~same~~ another name for same variable.

* Home Name & School Name : Analogy.

OOPs use this concept while C does not allows this

Date: / / Page no:

(i) object as a function argument can be done in 2 ways

(a) a copy of entire object can be pass to the function
Call / Pass by value

(b) only the address of the object as transferred to the function

```
Class Time {
    int Hr;
    int Min;
  public:
    void getTime (int h, int m) {
        Hr = h;
        Min = m; }
    void putTime ( ) {
        cout << Hr;
        cout << Min; }
    void sum (Time, Time); } ;

void Time :: sum (Time t1, Time t2)
    Min = t1.Min + t2.Min;
    Hr = Min/60;
    Min = Min % 60;
    Hr = Hr + t1.Hr + t2.Hr
}
int main ( ) {
    Time t1, t2, t3;
    t1.getTime (2, 45);
    t2.getTime (3, 45);
    t3.sum (t1, t2)
    t1.putTime ( );
    t2.putTime ( );
    t3.putTime ( );
}
```

//Call By value:

we can write name also

call By value

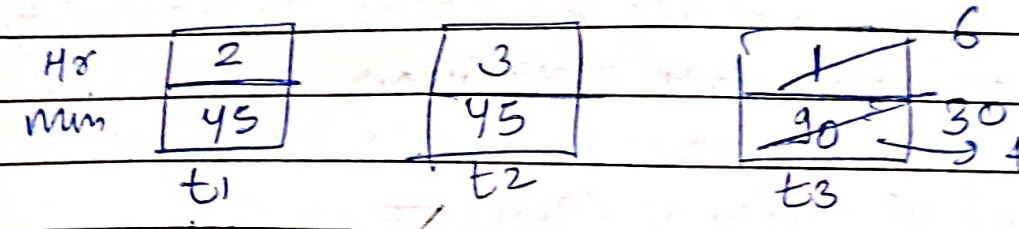
without object then automatically it will update object which called

2 45

3 45

6 30

When we want to pass object as argument write Class-Name in f^n declaration



$t1.min + t2.Min$
$= 45 + 45 = 90$

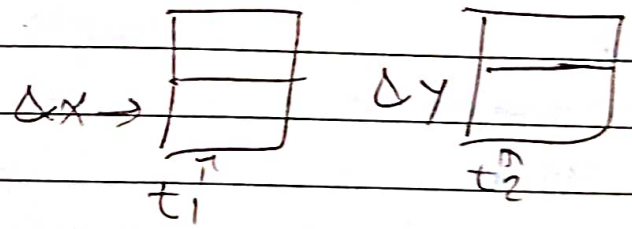
$Hr = Hr + t1.Hr + t2.Hr$
$= 2 + 3 + 1$
$= 6$

$Hr = Min / 60 = 1$
$Min = Min \% 60 = 30$

2 copies will be made
→ one at time of getHour()
→ one at time of sum() fun^n:

Call By Reference:

```
void sum (Time t, Time t);   //decl

void Time :: sum (Time &X, Time &Y) {

}
```



Changes in any one can change value of both.

Only one copy of ~~ob~~ object is made.

Date. / / Page no:

Friend function :- Can ~~take~~ access private members

(i) An ordinary function that is not the member function of a class has no priviledges to access the private data member. But the friend function does have the capability to access any private data members.

(ii) We declare the friend function using the friend keyword inside the Body of Class ~~so fu~~

Characteristics :-

(i) It is not in the scope of the class to which it has been declared as friend.

(ii) Since it is not in the scope of the class it can't be called using the object of that class.

(iii) It can be invoked like normal function without the help of any object.

(iv) It can be declared either in private ~~or~~ and public part of the class without affecting its meaning.

(v) Usually it has object as argument.

```
class item {
    int number;
    int cost;
  public:
    void set values(x, y)
    { number = 10;
      cost = 100; }
    friend float mean (item s); };
float mean (item s) {
    return float (s.number + s.cost) / 20;
  }
```

19.09.2022
Monday

Date: / / Page no:

```
main() {
    item x;
    x.setvalue()
    cout << mean(x); }
```

can access directly since it is not the member fn of class but can access private member of class

friend fn

(vi) The friend function accesses the class variable number & cost by using the (.) operator and object pass to it

The function mean(x) passes object x as its values to the friend function,

Constructors

must be in public

(special member function of class)

(i) Constructor is a special member function for automatic initialization of an object.

(ii) Whenever an object is created the Constructor will be executed automatically.

(iii) A constructor function can be overloaded to accomondate many different form of initialization.

Syntax of constructor function:-

(i) A constructor name must be same as that of its class name

(ii) It is declared with No Return type.

```
class abc {
    public:
        abc() {
            cout << 'Constructor Call';
        } }
```

Same name

```
void main() {
    abc obj1;
}
```

Types of Constructor

① Default constructor ② Parametrised Constructor
③ Copy Constructor
(Call By Reference)

① Default constructor :- (Implicit constructor)

(i) It is a constructor with no argument or all arguments should have default values.

(ii) It can be explicitly written in a program in case if constructor is not defined in a class. The C++ compiler automatically generates it in the program.

```
abc() {      }
class Student {
    private:  char name[20];
              int roll no;
    public:  ① Student() {      }
}
void main () { student obj1; }
```

② (Implicit)
If not constructed then automatically

③ Default initial value
Student(int x=10)
{ roll no. = x; }
Same value

(① & ③ are explicit)

② Parametrised Constructor :-

```
public:  Student (int x) { roll no = x; } }
void main() { Student object; //(Once explicitly constructor is created then default constructor is not there)
                ✗
              Student object (10);
                ✓
```

(i) It is a constructor with parameters defined in the class.

(ii) If a constructor with parameter is defined in the class, then C++ will not add any default constructor in a class.

Date / / Page no:

Overloading Constructors :-

same name but different no. of arguments / type of arguments

① student () { cout << "Without argument"; }
② student (int x) { roll no = x; }
③ student(~~int~~ char x, int y) { cout << "two arguments"; }
④ student (char x) { cout << "char argument"; }

```
void main() {
        Student obj1 ;          //  —— ①  (Explicit constructor)
        Student obj2(20);       //  —— ②
```

Q) WAP to generate series of fibonacci number using constructor.

Sol:
```
class fib {
    public:     int z; int x; int y;
    fib( ~~0, int x=0, int y=1~~ ) {
            ~~z = x+y;~~ int x=0; int y=0;
    }
int get data ( ~~int z~~ ) {  ~~cout << z << endl;~~ return z; }
};

void main() {
    int x=0, int y=1; a, b   ~~fib arr[10]~~
    for( int i=2; i<10; i++ ) {   ~~arr[0]=x~~ ~~arr[1]=y~~
        ~~fib~~ fib(x, y);
        ~~int x~~ x = y;
            y = get data;
            cout << y << endl;
    }
fib obj1 ();
obj1. get fibo (10);
```

20.09.2022
Tuesday

Date: / / Page no:

```
class Fib {
    int x ; int y;
    public:  Fib() { x=0; y=0; }
                int x=0; int y=0
    void getFibo(int n) {
        for (int i=2 ; i<n-2 ; i++) {
            cout << x << " " << y << " ";
            int t = x+y;
            x = y ;
            y = t
        }
};
void main() {
    Fib obj;
    cout << "0"
    obj.getFibo(10);
```

Q) WAP to find area of circle & rectangle using class & objects where all the data members should be kept in a private section.

Sol:

```
class circle {
    int radius;
    public;
        ~~int area(int r) {~~ circle(int val) {
                                radius = val; }
    int area (float r) {
        return (3.14) * r * r ) ; }
};
class rectangle {
    int l;
    int b;
```

Date: / / Page no:

```
public:     rectangle (int len, int ben){
                l = len;
                b = ben; }
int area (int l, int b){
        return l*b; }
};
int main(){
        circle obj1.(2);
cout << obj1.area (2) << endl; // 3.14*2*2 = 12.56
        rectangle obj2 (2,3);
cout << obj2. area (2,3);          // 2*3=6
    return 0; }
// can be done by fn overloading
```

M.2

```
class Test {
        int r, l, b;
    public:     or Test
        void getdata (int r, int l, int b){
                r = r;   l = l;   b = b; }
        void AreaOfCircle () {
                cout << (3.14*r*r) << endl; }
        void Area of Rectangle (int l, int b){
                cout << l*b << endl; }
    };
int main(){
        Test obj (1,2,3);
        obj. Area Of Circle ();
        obj. Area of Rectangle ();
}
```

③ Copy Constructor :-

(i) CC is a constructor which is used to initialize an object by another object of the same class which has been created previously.

(ii) The general format of copy constructor

```
class_name (const class-name & obj)
{
    // body of const
}
```

Dying

Passing object by Reference. (Aliasing)

ex.

```
class Test
{ int x, y, z;
  public:
      Test (int a, int b) {
          x = a;   y = b; }
  Test (const Test & obj) {
          x = obj.x ;
          y = obj.y ;
      }
};
```

Calling

Catched by this

```
int main ()
{ Test obj1(10, 20);
  Test obj2(obj1);
    // or
  Test obj3 = obj1;
}
```

Copy Constructor Calling

(iii) If we don't define our own copy constructor then C++ compiler creates a default copy constructor for each class which does a member wise copy b/w objects.

(iv) In C++, a copy constructor called in following cases:

(a) When an object is constructed based on another object of the same class.

21.09.2022
Wednesday

Date. / / Page no:

(b) When an object of a class is passed by an argument.

(c) When an object of a class is returned by value.

ex. Relay race video game to know the location of previous players to next player.

Destructor :

① A Destructor is a function that automatically executes when object is destroyed.

② The primary uses of destructor function is to release space occupied by an object.

③ A destructor function may be invoked explicitly. Generally at our level, we don't require it much as there will be no memory problem.

④ The Rules for writing a destructor function.

(i) The Destructor function name is as that class it belongs except that the first character of the name Tilde sign '~'

```
(ii) class ABC {
        public:  ABC() { }
        ~ABC() { }
     };
```

(iii) It is declared with no return type.

(iv) It not takes any argument & therefore cannot be overloaded.

(v) If we do not write our own destructor in class compiler creates destructor for each class.

Returning Object As an Argument

```
class Ex
{ public:
    int a;
    Ex add (Ex X, Ex Y)
    {   Ex Z;
        Z.a = X.a + Y.a;
        return Z;
    }
};
```

return type = object of type class :: class-name

returning object

```
int main()
{
Ex E1, E2, E3;
E1.a = 50
E2.a = 100;
E3.a = 0;
cout << E1.a << E2.a << E3.a << endl;
E3 = E3.add (E1, E1)
cout << E1.a << E2.a << E3.a << endl;
}
```

O/p: 50 100 0
50 100 150

Friend function

present in C++ but not in Java

It leaks OOP principle in terms of privacy

∴ C++ is not pure OOP language.

(1)

```
class XYZ {
    public:
        friend void sum ();      //Not a member fn
};

void sum () {                    //No scope resolution

}
int main()
{ sum ();                        //No calling requirement of object
}
```

② Member function of a one class can be friend function of another class. In such case they defined scope resolution operator.

①
```
class A {
    public:
        void show();
};

Class B
{ public:
  friend void A::show();
};
```

Good to declare in such a way to avoid error
First telling that we will make class.

∴ Class B should be above Class A

②
```
class B; //
class A {
    public:
        void showB(B&);
};
class B {
    int b;
public:
    B() { b=0; }
    friend void A::showB(B& x);
};
void A::showB(B& x)
{
    cout << x.b;
}
```

→ Reference of B's object

```
int main()
{ A a;
  B x;
  a.showB(x);
};
```

// Not a member fn of B but member fn of A

① // Member function of A but not of B

O/P 0

Date: / / Page no.

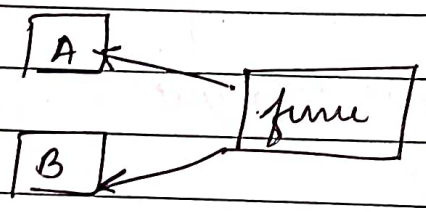
③ We can also declare all the member function of one class as the friend function of another class in such case class is called Friend Class.

Way1 Change Ⓘ to

friend class A; // other same

Way2 Individually change for each member function but time consuming

④ A Non member can access private members of multiple classes.




```
class B;
Class A {int x;
public:
void set (int a)
  { x = a; }
friend void max (A,B); };
class B {
  int b;
public:
void set (int x) { b = x; }
friend void max (A,B)
};

max (A obj1, B obj2)
{ If (obj1.x > obj2.b) {
      cout << obj1.x; }
}
```

```
int main()
{ A a;
  B x;
a.set (50);
x.set (100);
Max (a, x);
}
```

O/P: 100

```
else {
cout << obj2.b; }
```

26.09.2022 Monday.

Date: / / Page no:

Q) WAP to convert a decimal no. into binary no. using friend function.

Sol:
```
class dtoB {
public: int a;
        friend string dtoB deitoBin (int a);
};

string dtoB deitoBin ( int obj) { string s = " ";
int a=obj.a;  while ( a >0) {
                int rem = a % 2;
                s += toString (rem)
                a = a / 2;
            }
            return s; }

int main () {
        dtoB obj.a = 10; ;   obj.a = 10;
cout << deitoBin ( obj );
return 0;
}
```

Date: / / Page no.

Classes within Class (Nesting of Class)

(i) C++ permits declaration of a class within another class

(ii) A class declared as a member of another class is called as Nested class.

(iii) The nested class is in the scope of its enclosing class.

```
class stud {
  private:
      char name[20];
      int rollno;
  public:
    class date {
    private:
    int day, month, year;
    public:
    void setdata (int d, int m, int y)
    {   day = d;          // 2
        month = m;        // 4
        year = y; }       // 2022
    };

    void display ();
  };
int main() {
      stud obj;
      stud :: date obj1;      // scope of date within stud
      obj1.setdata (2, 4, 2022);
}
```

Stud
↓
date

we can make object date inside class stud without making scope

Date: / / Page no:

```
Class outer {
  { private:
        Class inner {
            private: int y;
        }
    }b
}a;

int main()
  { outer obj1;
  outer :: inner obj1; ✗  inner is in private
```

(iv) The members of an enclosing class have no special access to members of a nested class.
(v) The usual access rules shall be obeyed.
(vi) Inner Class can access private data of enclosing class.

Q) int x, y inner Class → public
private inner inner's fun. add then.
outer

Sol:
```
class outer {
    int x;
  public:
    class inner {
        int y;
  public:   [ get data (int a, int b) {inner.y=b; outer.x = a ;}
    int addInt (int a, int b) {
        return a+b;
    }
}b;
};a
```

Date / / Page no.

```
int main() {
    ~~int~~ Outer :: Inner obj;
        obj.getData(1,2)
        obj.~~int~~AddInt(
```

```
class Outer {
    int x;
    public:
        int a
        Outer() {                    // constructor of Outer
            x = a
        }

    class Inner {
        int y;
        public:
            Inner (int b) {          // Constructor of Inner
                y = b;
            }
            int addInt (Outer obj1, Inner obj2) {
                return obj1.x + obj2.y;
            }
    };
};

int main() {
    ~~Outer (2);~~ Outer obj1(2);
    Outer :: Inner obj2(3);
cout << obj2.addInt(obj1, obj2);
    return 0;
}
```

Date: / / Page no:

```
class outer {
    private:    int x;
    public:
      ~~class~~ class inner {
            private:
                int y;
            public:
                void fun (outer obj) {
                    cin >> y;
                    int z = obj.x + ~~obj~~ y;
                    cout << z;
                }
            };
    void outerfun ( ) {
            cin >> x;
        }
    };
void main ( ) {
        outer obj1;
    outer :: inner obj2;
        obj1.outerfun ( );
        obj2.fun (obj1);
    }
```

Nested class can access private members of enclosing class

But

Enclosing class can't access private data members of nested class

27.09.2022
Tuesday

Real life example of OOPs

Date. / / Page no.

INHERITANCE :

Reusability of Code

(i) Inheritance is the process of creating a new class from an existing class.

(ii) It is a form of software reuse in which you create a class that build on existing class capabilities. than customised or enhanced one.

(iii) The existing class is known as Base Class and newly created class is known as child class

Ex. ① 4 wheelers → Car, Truck, Bus

② Phone (Base Class) → MI, Samsung, Apple (Derived Class)

(iv) Inheritance represents a "IS A" Relationship

Car IS A Vehicle (4 wheelers).

Samsung IS A Phone.

Derived IS A Base

Derived Class (DC)

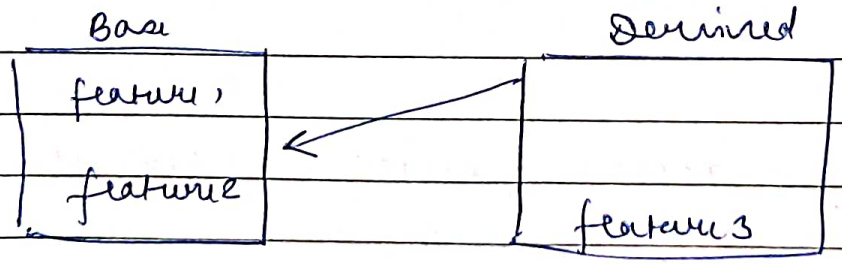
Base Class (BC)

(v) The main advantage of Inheritance are

(a) Reusability of Code

(b) Increase the Reliability of Code

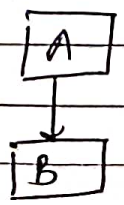
(c) To add some enhancement to the Base Class.



Types of Inheritance

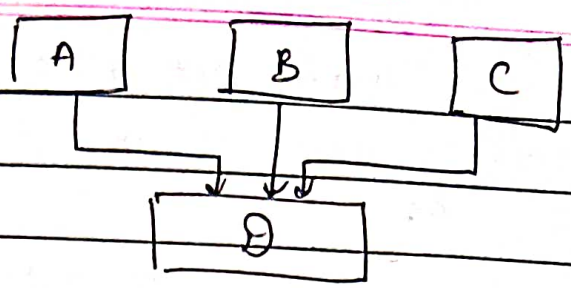
① Single Inheritance

(i) A DC with only one BC



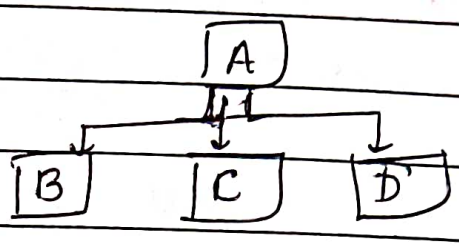
② Multiple Inheritance

DC with several BC.



③ Heirarichacal Inheritance

Property of 1 BC may be inherited by more than 1 DC.



④ Multilevel Inheritance

Deriving a class from another DC.

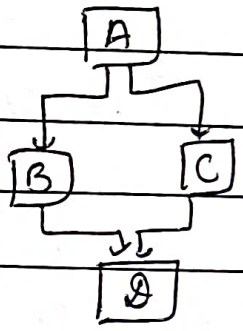
⑤ Hybrid Inheritance

It is a combi^n of 2 or more type of Inheritance.

Remove C, ~~Heirarichacal~~ Multilevel Inheritance

Remove A, Multiple Inheritance

Remove D, ~~Hu~~ Heirarichacal Inheritance



General Syntax of derived Class.

class derived_class_name : visibility_mode Base_Class_Name

{ Data member &
member functions
};

colon

Example of single Inheritance

```
class B {
    int a;
    public:
        int b;
    int void get_ab ();
```

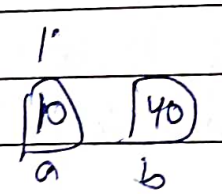
```
    void get_a();
    void show_a();
};

class D : public B
{   int c;
    public:
        void mul();
        void display();
}

void B:: get_ab() {
        a=10;   b=20; }

int void B :: get_a() {
        ~~a=~~ return a;
    }

void B:: show_a()
    { cout << a;
    }
```

```
void D:: mul() {
    get c = b * get_a()
    }

void D:: display()
{ cout << get_a()
    cout << b
    cout << c;
}

int main() {
    D d;
    d.get_ab();
    d.mul();
    d.show_a();
    d.display();
    ~~return 0;~~
}
```

```
d.b=40;
d.mul();
d.~~show_a()~~;
d.display
```

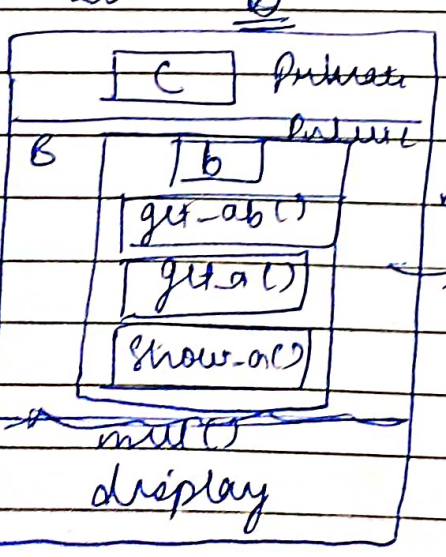
```
d.b=40
d.a=60;
d.mul();
d.~~show_a~~;
d.display()
```

o/p
10
10
20
200



10 × 40
c = 400

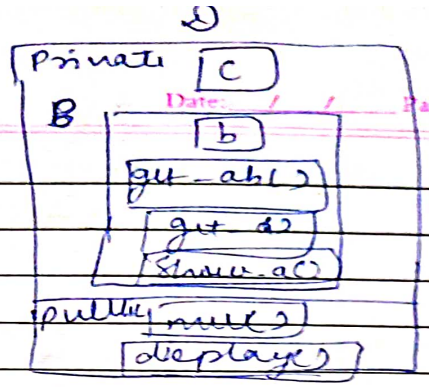
10
10
40
400



private data member of Class B a will generate error.

public

If I do visibility mode = Private



visibility mode divides →
Public के साथ क्या होगा।

① The derived class 'D' inherits the properties of Base Class 'B' including its data members & member function. With all form of Inheritance private member of BC is not accessible directly.

② ~~If visible~~ By default visibility mode is treated as private (If visibility mode is not defined)

Making a Private member Inheritable:-

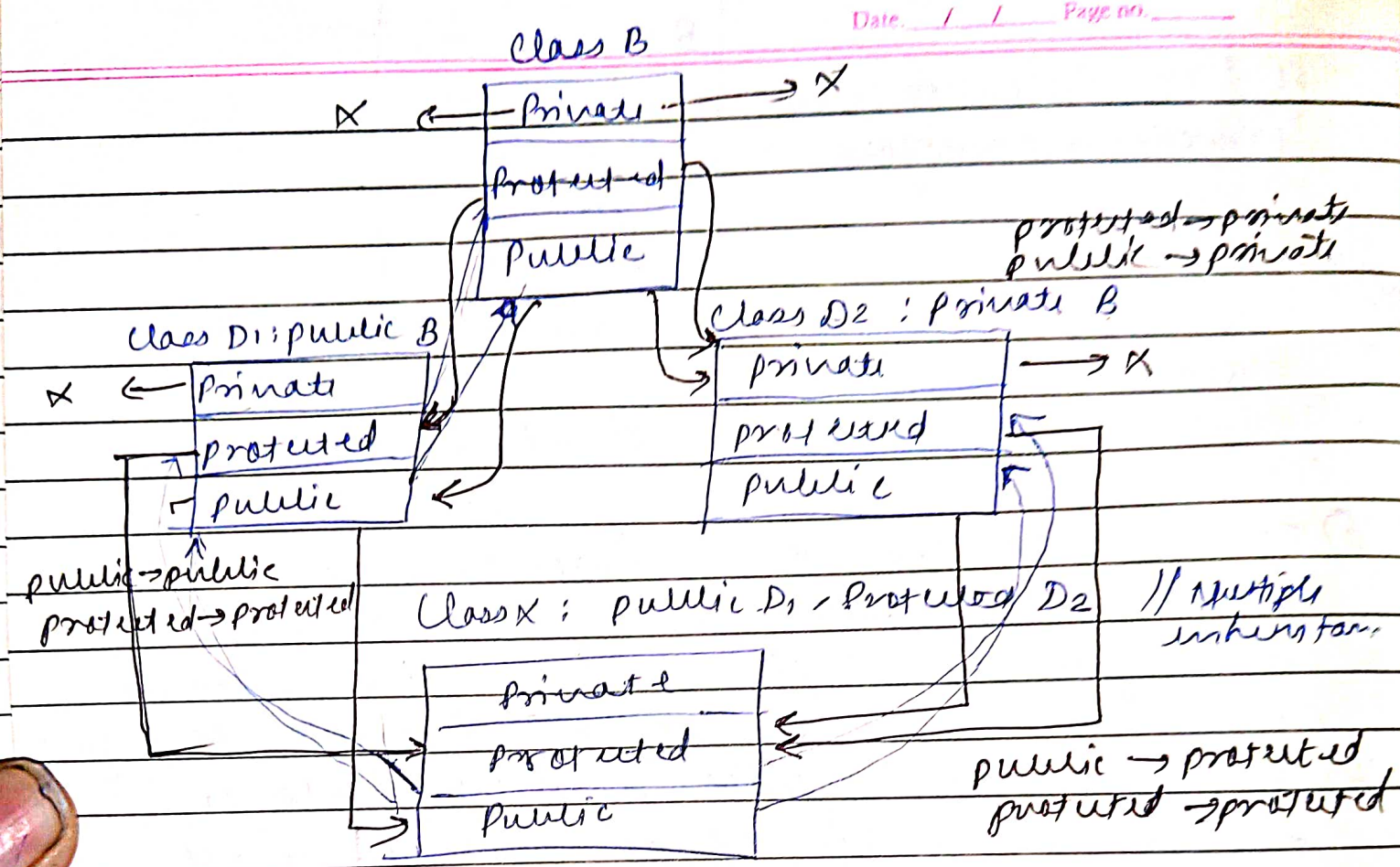
(i) Private members of base class can't be inherited & therefore ~~can~~ it is not available directly

(ii) If the private data needs to be inherited to DC, it can be possible by making it public but it become open for all.

(iii) C++ provides third visibility modifier "Protected" which serve a limited purpose in inheritance.

(iv) A member declared as protected is accessible by the member f^n within its class and any class immediately derived from it.

(v) It can't be accessed by functions outside these 2 classes.



B → D1 → X
protected is inherited

B → D2 → X
~~protected to~~ D2 private
Not accessible

public → जो जैसा, वैसा ही।

others → 3+4