Quiz: 11+12

14/10/2025

: Introduction to OOPS :

→ C-suppord

Modular Programming

Bank {example}

- Open Acc ( )
- deposit ( )
- withdraw ( )
- check bal ( )
- apply loan ( )

(Set of function)

object-oriented Programming

Govt :

objects:
- Electric service → new conn, close conn, Bill payment
- water
- education
- transport
- Banks

Departments

→ set of objects → contains → set of fn

* Principles of object-oriented :

- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

1. **Abstraction:** We don't wit... abstr...

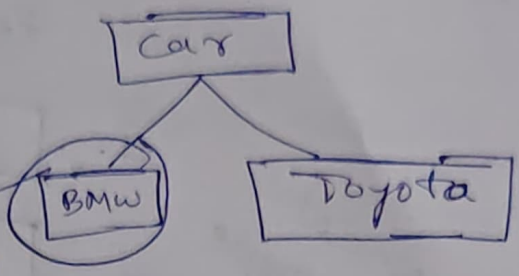
class my → blueprint of class
{
  1-Data
  2- function ( )
}

OOP's: bottom to down procedure.
- data is secure
- procedure and data structures are combined
- date use to perform code of piece specific task
- Debugging is

C++, Java, JS, Kotlin

→ we can holding data.

3. Inheritance →



Some ~~good~~ design ← are followed by all

4. polymorphism: ~~If~~ help of inheritance we use polymorphism

- one known car driving we also drive the all cars.

* class vs objects:

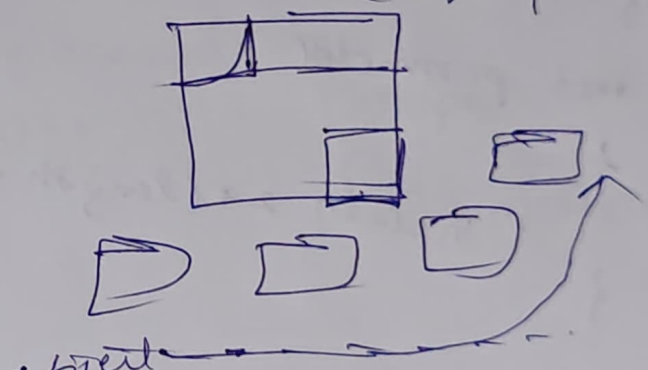
→ from classification → categorise everything

→ Criteria.

Class (Blue print)

class Human ← you, my

class car ← Bmw x1, Camry



class rectangle

{

data, ----

float length;

float breadth;

float area ();

float perimeters ();

float ---- ();

};

in u

~~cla~~

4 principle OOPs

1) ~~Abstraction~~

2) Encapsulation

3) Inheritance

4) Polymorphism.

{ Rectangle r1, r2, r3;
objects
}

* Writing a class:

Class rectangle { } own data type
{ public:
int length;
int breadth;

int area()
{
return length * breadth;
}
int perimeter()
{
return 2*(length + breadth)
}
};

Rectangle

1. b, l
2. b, l
3. b, l

area()
perimeter()

private

int main()
{ } { fn not occupying any memory }

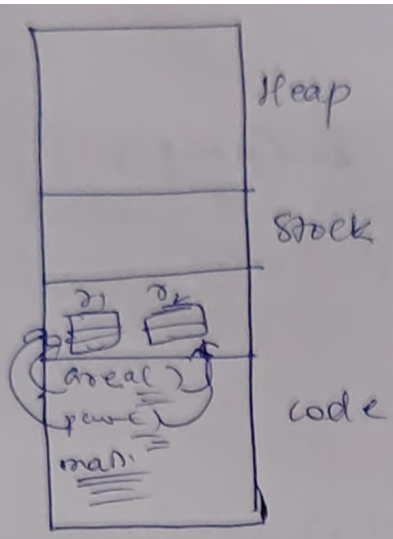
r1
4 | length | breadth

rectangle r1, r2;
r1.length = 10;
r2.breadth = 5;
cout << r1.area(); → 50
cout << r2.area(); →

data operator

We can't acces the data because by default it is public, when we write private; public; than we can't acess.

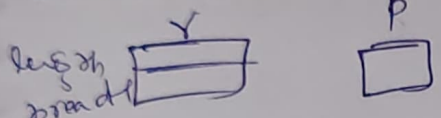
We also acess the from dot operator members & member data operator.



class def^{n} not there, but only have function.

if you don't write public then error "is private within this context",

→ Pointer to object:




```
int main()
{
  Rectangle r;
  Rectangle *p;        → for accessing pointer.
  p = &r;
  r.length = 10;
  p -> length = 10;
  p -> breadth = 5;
  cout << p -> area();
}
```

```
class Rectangle
{
  public:
    int length;
    int breadth;
    int area()
    { return length * breadth; }
    int perimeter()
    {
      return 2*(length + breadth);
    }
};
```

for heap (store):

```
void main()
{
  Rectangle *p;
  p = new Rectangle;
  Rectangle *q = new Rectangle(); // for creat object in heap
}
```

Rectangle r; ← Stack

Rectangle *p = new Rectangle(); ← (Heap)

Demo:

```
class Rectangle
{
  public:
    int breadth;
    int length;
    int area()
    {
      return length * breadth;
    }
    int perimeter()
    {
      return 2*(length + breadth);
    }
};
int main()
{
  Rectangle r1;
  Rectangle *ptr;
  ptr = &r1;
  ptr -> length = 10;
  ptr -> breadth = 5;
  ____________
  ____________
}
```

(ptr pointing on that address)

ptr → area()
ptr → perimeter()

both are same

```
{
    Rectangle *ptr = new Rectangle;
}
```

* Philosophy of Data hiding:

Now we all do public data, functions and in program.

→ If data public than we can access it.

→ If I write breadth (-5); if public is the code Area = (-50), Area can't be negative, not handling

* ↓ for that data members private, and made the public function, Now we can't read and write the data.

* It give garbage value,

(area with garbage)

setting value of length;

```
public:
    void setLength(int l)
    {
        length = l;
    }
    void setBreath(int b)
    {
        breadth = b;
    }
```

getting value;

```
int getLength()
{
    return length;
}
int getBreadth()
{
    return breadth;
}
```

Using for that

```
void main()
{
    Rectangle r;
    r.setLength(10);
    r.setBreadth(5);
    cout << r.area();  → 0
    cout << "length is" << r.getLength();
}
```

r

length	10
breath	5

→ write in fn

```
{ if(l>0)
    length=l;
  else
    length = 0;
```

↙ validate-function

```
if(b>=0)
{
    breadth = b;
else
    breadth = 0;
```

→ mutator
set → change the value
get → give output (as)
↳ Accessor
→ property-functions

```
class rectangle{
public:
  int area()
  {
    ——
  }
  int permeter()
  {
    ——
  }
};
```

int main()

→ write what we wrote (set and get functions)

add fraction of two number

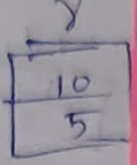
```
#include<iostream>
using namespace std;
class fraction
{
    public:
        int num;
        int dum;
        friend istream & operator>>(istream &in, fraction &f);
        friend fraction operator+(fraction f1, fraction f2);
        friend ostream & operator<<(ostream &out, fraction &f);
};
istream & operator>>(istream &in, fraction &f)
{
    in>>f.num>>f.dem;
    return in;
}
fraction operator+(fraction f1, fraction f2)
{
    fraction t;
    t.num = f1.num*f2.dem + f2.num*f1.dem;
    t.dem = f1.dem*f2.dem
    return t;
}
ostream & operator<<(ostream &out, fraction &f)
{
    out<<f.num<<"/"<<f.dem<<endl;
    return out;
}
int main()
{
    fraction f1,f2,f3;
    cin>>f1; cin>>f2;
    f3=f1+f2;
```

$4/5 + \frac{2}{3}$

$\frac{12+10}{15} = \frac{22}{15}$

Algorithm	Best time	Average time	Worst time	Worst space
Linear S	$O(1)$	$O(n)$	$O(n)$	$O(1)$
BS	$O(1)$	$O(\log n)$	$O(\log n)$	$O(1)$
Bubl Sort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
SS	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$
IS	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$
MS	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
QS	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Bucket S	$O(n+k)$	$O(n+k)$	$O(n^2)$	$O(n)$
Heap S	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$
Radix S	$O(nk)$	$O(nk)$	$O(nk)$	$O(n+k)$
Shell sort	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$
Tim sort	$O(n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$

Constructors :



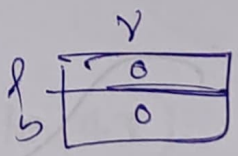
void main()
{
Rectangle r;
r.setLength(10); X
r.setBreadth(5);
Rectangle r(10,5);

* **Constructor :**
a special 'MEMBER FUNCTION' having the same name as that of its class which is used to initialized some valid values to the data member.

* Rectangle r;

1. Default constructor
2. Non-parametized "
3. Parametized constructor
4. copy constructor.

class Rectangle
{
private:
int length;
int breadth;
public:
Rectangle()
{
length=0;
breadth=0;
}
};



Non parameter

don't have return type

Rectangle r(10,5);

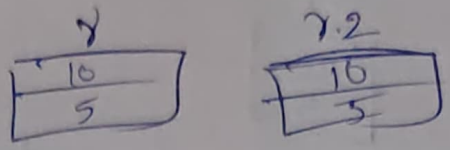
parameterised

do their work

class Rectangle
{
private:
int
void setBreadth(int b)
{ if(b>=0)
breadth=b;
else breadth=0;
}
int getLength()
{
return length;
}
int getBreadth()
{ return breadth;
}
};

Rectangle(int l, int b)
{ setLength(l)
set breadth(b);
}

from non-parameterised didn't get any design, Colour & Brand.



Copy constructor:

```
Rectangle (Rectangle &r)
{
    length = r.length;
    breadth = r.breadth;
}
```

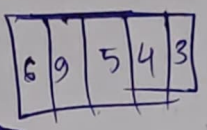
by reference

Rectangle r2(r);

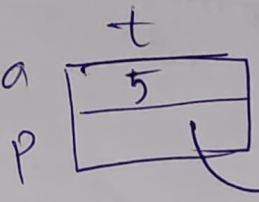
In main function

Overloading constructor repeated function 5

* Deep copy constructor:

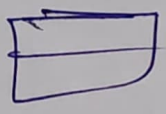


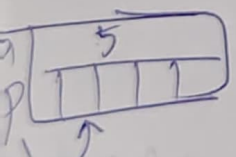
errors:




```
main()
{
    Test t(5);
    test t2(t);
}
```

```
class test
{
    int a;
    int *p;
    Test(int u)
    { a = u;
      p = new int[a];
    }
    Test(Test &t)
    {
      p = new int[a] a = t.a
      p = t.p;
    }
};
```





you we have copy the array but not defending one array

Demo:

```
#include <iostream>
using namespace std;
class rectangle
{
  privater:
    int length;
    int breadth;
  public:
rect ← rectangle ()
    {
      length = 1;
      breadth = 1;
    }
  rectangle (int l, int b)
  {
    setLength (l);
    setLength (b);
  }
  rectangle (rectangle &r)
  {
      length = r.length;
      Breadth = r.breadth;
  }
```

→ own constructor

```
int main()
{
  rectangle r1;
  cout << r1.area()
       << endl;
```

(1)

```
Rectangle r1(10,5);
```

(50)

```
Rectangle r2(r1);
cout << r2.area() <<
        endl;
```

```
int l=1, int b=1;
```

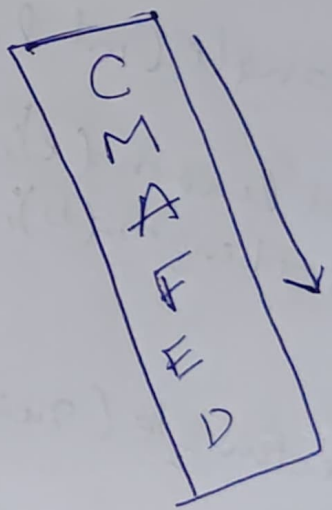
default agrm
evil.

if do this made

Types of functions in a class

20/10/2023

```
class Rectangle
{
    private:
        int length;
        int breadth;
    public:
        Rectangle();  → non-parameterised constructor
        Rectangle(int l, int b); → parameterised constructor
        Rectangle(Rectangle &r); → copy constructor
        mutator → setLength(), setBreadth();
        Accessor → getLength(), getBreadth();
facilator { int area();
            int perimeter();
enquire ← int isSquare();
        Destructor function;
};
```



Scope Resulation operator:

```
class Rectangle
{
    private:
        int length;
        int breadth;
    public:
        int area()
        { return length * breadth;
        }
```

}; Rectangle → scope resolution

int ::permiter() → same in class & outside class along scope resolution

{

return 2*(length + breadth);

}

main()

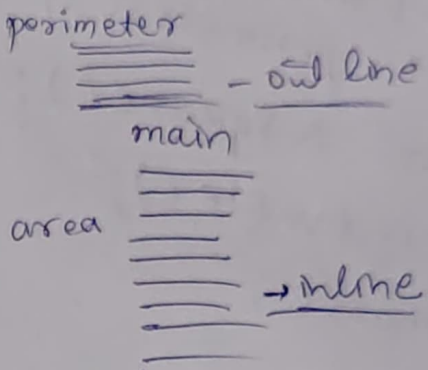
{

Rectangle r(10,5);

cout<< r.area();

cout<<r.perimeter();

}



(seperataly generated. in memory)

if write function in class that will be inline function.

- Never right loops in the class.

We also use constructor use like scope resolution.

```
#include<iostream>
using namespace std;
class rectangle{
  private:
     int length;
     int breadth;
  public:
     Rectangle();
     Rectangle(int l, int b);
     Rectangle(Rectangle &r);
     int getLength();
```

```
int getBreadth();
void setLength(int l);
void setBreadth(int b);
int area();
int perimeter();
bool isSquare();
~Rectangle();
int main()
{
```

```
Retangle::Rectangle()
{
    _____
    _____
}
Retangle::Rectangle(int l,
                    int b)
{
    _____
    _____
}
```

Code:

```
#include<iostream>
using namespace std;
class recatangle
{
  private:
    int length;
    int width;
  public:
    recatangle(int l, int b);
    void setLength(int l);
    void setWidth(int b);
    int areae();
    int perimeter();
    bool isSquare(); ⟶ return true & false value
};
```

```
int main()
{
   rectangle r1(10,10);
   cout<<r1.area()<<endl;
   cout<<r1.perimater()<<endl;
    if(r1.issquare()) cout<<"YES"<<endl;
    else cout<<"NO"<<endl;
    return 0;
}
rectangle::rectangle(int l,int b)
{ length=l;
   width=b;
}
void rectangle::setLength(int l)
{ if(l>0) length=l;
   else length=1;
}
void rectangle::setWidth(int b)
{ if(b>0) width=b;
   else width=1;
int r1 ::area()
{ return
}
```

perimeter

Inline function:

91/10/2023

→ fn which expand which they are called.

```
int main()
{
    Test t;
    t.fun();
    t.fun2();
}
```

```
class Test
{
public:
    void fun()
    {
        cout<<"Inline";
    }
    → void fun2()
};
```

- If write function inside in class that inline

than it is inline ← if write here (inline) ←

```
void Test::fun2()
{
    cout<<"non-inline";
}
```

* this pointer: store name ambigunt (this)

```
public:
    Rectangle(int length, int breadth)
    {
        this → length = length;
        this → breadth = breadth;
    }
```

from this pointer we can access the member of class

```
Rational(Rational &r)
{
    this → p = r.p;
    this → q = r.q;
```

```
struct Demo
{
    int x;
    int y;
    void display()
    {
        cout << x << " " << y << endl;
    }
};
int main()
{
    Demo d;
    d.x = 10;
    d.y = 5;
    d.display();
```

- in C language only have data member, not functions
- But in C++ language have data member and also have functions
- C++ structure is like class
- In class all are private but in structure are all public.

* Write a class for student with 1. Roll, 2. Name 3. Marks in 3 subject, function for total marks, grade and required methods.

Code:

```
#include<iostream>
#include<string.h>
using namespace std;
class student
{
    private:
    int roll_no;
    string name;
    int math_marks;
```

Functions inside the class, that are become inline functions.

```
int phy_marks;
int chem_marks;
public:
student (int r, string s, int m, int p, int c)
{
  roll_no = r;
  name = s;
  math_marks = m;
  phy_marks = p;
  chem_marks = c;
}
int total()
{
  return math_marks + phy_marks + chem_marks;
}
float avg()
{
  return total()/3;
}
char grade()
{
  float average = avg();
  if(average > 60) return 'A';
  else if (average >= 40 && average < 60) return 'B';
  else return 'C';
}
};
```

```
int main()
{
    int roll;
    string name;
    int m,p,c;
    cout<<"Enter name of stud
    cin>>roll;
    cout<<"Enter marks of 3
    cin>>m>>p>>c;
    student s(roll,name,m,p,c
    cout<<"Total marks:"<<s.total()<<endl;
    cout<<"Average marks:"<<s.avg()<<endl;
    cout<<"Grade of students:"<<s.grade()<<endl;
    return 0;
}
```

* Object will consume memory for data members. functions will be common for all objects in memory.

31/10/2023

Quiz 7:

* Operator Overloading:

class matrix.

C = A + B

int, float + * - ( ) ++ new delete

Complex number

a + ib

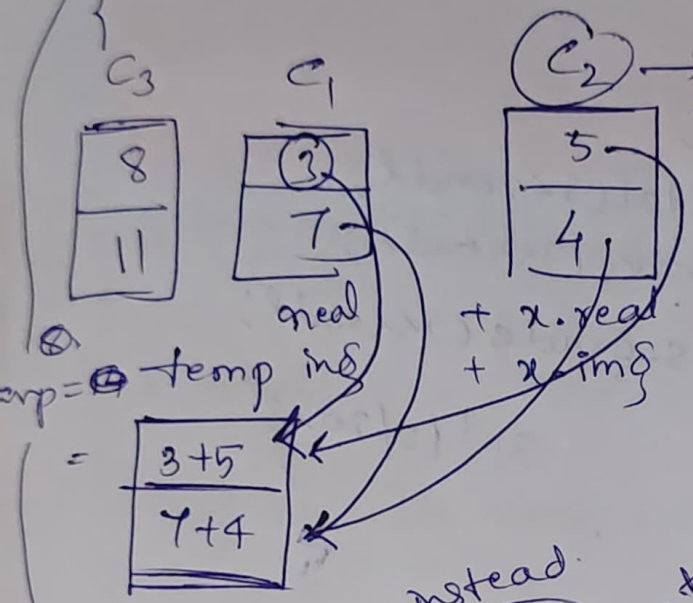
real img

$i = \sqrt{-1}$, $5+7i$, $2+9i$

$7+16i$ → real seperate & img seperate

$C_3 = C_1 + C_2$

main()
{ Complex C1(3,7)
Complex C2(5,4)
Complex C3(0,0);
C3 = C1 + C2;
C3 = C1.add(C2); } both are same
C3 = C2.add(C1);



real means C1 → real

Class Complex
{
private:
int real;
int img;
public:
Complex(int r=0, int i=0)
{
real = r;
imag = i;
}
operator+
Complex (Complex x)
{
Complex temp;
temp.real = real + x.real
temp.img = img + x.img
return temp;
}

C3 = C1.operator+(C2);
C3 = C1+C2;

instead → this not necessary

Simple

Program: Friend function:

main()
{
Complex

Example:

#include <iostream>
using namespace std;
class complex
{
public:
int real;
int imag;
pu

* Friend function:

main()
{
complex C1(3,7)
complex C2(5,4)
complex C3;

C3 = C1 + C2.

<<>>

* Insersation & extraction → used for output
>>

1/11/2023

fread ← C2 > give all money
fread ← C1 < to your fread

class complex
{
private
int real;
int img;
public;
friend complex operator+
(complex C1
complex C2

we don't use scope resultation operator.

complex operator + (complex C1, complex C2)
{;

complex t;
t.real = C1.real + C2.real
t.img = C1.img + C2.img
return t;
}

class
{
private
int real;
int img;
public:
void display()
{
cout << real << "+i" << img;
}
friend ostream & operator<<(ostream &O, complex &C1)
};

main()
{
complex C1(3,7)
int x = 10;
cout << x;
C1.display();
(3+i7) → like
cout << C1;

ostream = output stream

Syntax of friend:
inside class
friend data type operator
(——, ——);
outside class
complex operator + (
complex C1, complex C2)

- friend ostream & operator << (ostream &o, complex &c1);
};

```
ostream & operator << (ostream &o, complex &c1)
{
    o << c1.real << "+i" << c1.img;
    return o;
}
```

(o → object) — when cout passed in, cout became o

Demo:

```
int main()
{
    complex c(10,5);
    cout << c;
    operator << (cout, c);
}
```

cout << c; → (endl) → both are same

cannot resolve.

We have to write to made ostream.

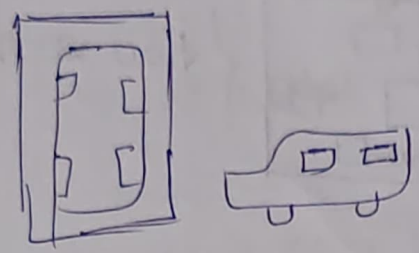
* Write a class for Rational number p/q with overloading + and << operator

$\frac{num}{denr}$. c1.add

$\frac{12+20}{24} = \frac{32}{24} = \frac{4}{3}$

* Inheritance:

Car → some more extra model.



design

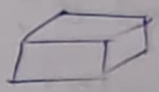
→ outdated class

→ borrow features of existing class to new class

→ Inheritance

- class rectangle
{ length;
  breadth;
  ===
};

Rectangle r1, r2;

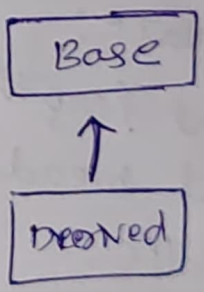


→ derived from Rectangle

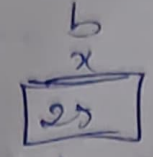
class cuboid Retaurange
{
  ———
  ===
};

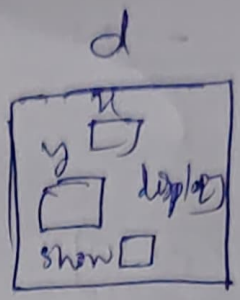
Example:

```
class base
{
  public:
    int x;
    void show()
    { cout<<x;
    }*
};
class Derived : public Base → deriving class syntax
{
  public:
    int y;
    void display() → borrowing from base class
    { cout<<"y"<<y;
    }
};
```



```
int main()
{
    base b;
    b.x = 25;
    b.show();
}
```





two variables.

```
Derived d;
d.x = 10;
d.y = 15;
d.show();
d.display();
```

Class show of b

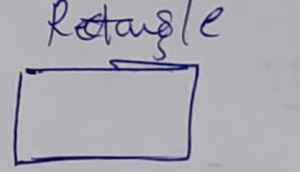
only give output x value

This give output x & y value

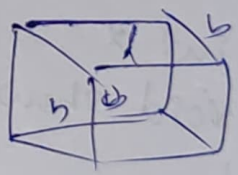
*

Class rectangle { In inheritance concept called · Base class.

```
{
    private:
        int length;
        int breadth;
    public:
        Rectangle(int r=0, int b=0);
        int getLength();
        int getBreadth();
        void setLength(int l);
        void setBreadth(int l);
        int area();
        int perimeter();
};
```



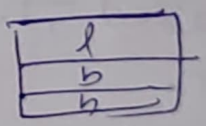
Cuboid



```
class cubiod : public Rectangle
{
    private;
     int height;
    public;
    cubiod (int l=0, b=0, int h=0);
    setLength(l);
    setBreadth(b);
}
int getHeight();
void setHeight();
int volume()
{
    return (getLength() * getBreadth * height);
}
};
int main()
{
    cubiod C(10,5,3);
    cout<<C.getLength();
    cout<<C.volume();
    cout<<C.Area();
}
```

This is not access directely because this are not member of cubiod class.

gives length of cubiod.



→ all functon include class cubiod.

* When you try to access the ~~sep~~ particular access member than that is only give that memory.

Constructor in Inheritance:

6/9/2023

```
int main()
{
    Derived d;
    Derived d(10);
    Derived d(20,10);
}
```

default

① parent class execute
→ next child class execute

```
class Base
{
public:
    Base()
    {
        cout<<"Default of Base"<<endl;
    }
    Base(int x)
    {
        cout<<"Param of Base"<<endl;
    }
};
```

non param

```
class derived: public Base
{
public:
    derived()
    {
        cout<<"Default of derived";
    }
    derived(int a)
    {
        cout<<"param of derived"<<a;
    }
    derived(int x, int a): Base(x)
    {
        cout<<"param of derived"<<a;
    }
};
```

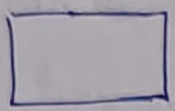
common calling

We can call parent class from child class extention any material added last.

→ In the first parent child/derived class will be called, than parent class.

isA vs hasA

16/11/2023



Class Rectangle
{
≡
}

↓ isA

Class cubiod : public Rectangle
{
—
}

cubiod is A Rectangle

Table has A rectangle

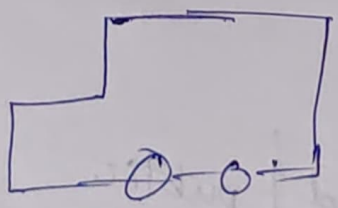
↓ hasA

table class

class Table
{
Rectangle top;
int Legs;
}

My car

Object



Access specifier:

public :
private :
protected :

int main()
{
Base x;
x.a = 15; } you can't access private and protected on object
x.b = 30;
x.c = 90;
}

x

15
30
90

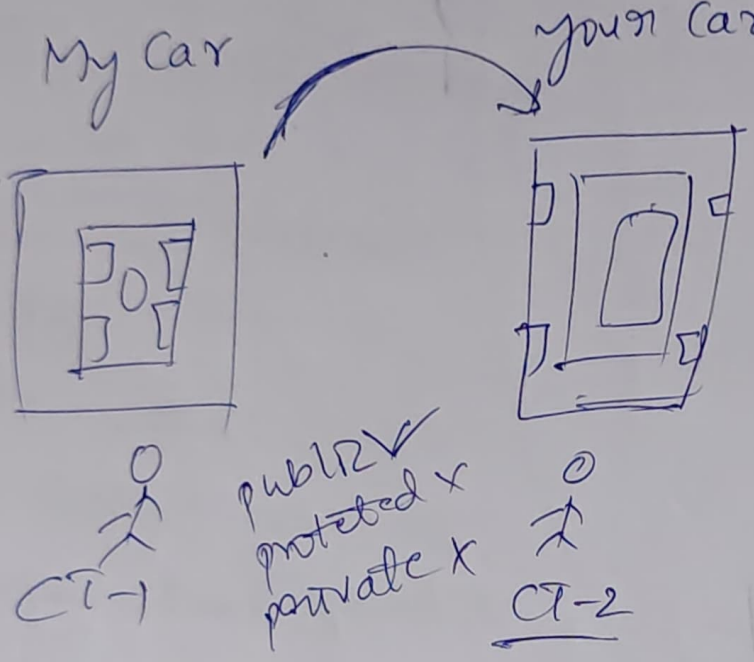
class Base ;
{
private :
int a;
protected :
int b;
public
int c;
void funBase()
{ a=10;
b=20;

```
class Derived : Base
{
public:
  funDerived()
    X a=1;
    X b=2;
      c=3;
};
```

Ans

all members are accessible only in that class not access that derived class

why private, protected, public [In that don't modified, that will be protected]




```
m1.setLength(10);
m1.setBreadth(5);
```

	private	protected	public
inside class	✓	✓	✓
inside derived class	✗	✓	✓
an object	✗	✗	✓

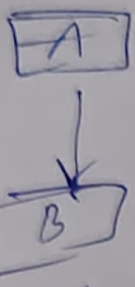
23/11/2021

```
#include<iostream>
using namespace std;
class triangle
{
    public s1 = "I am an isosceles triangle";
    void show()
    {
      cout<<s1;
    }
};
class isosceles:public triangle
{  public:
    string s2 = "In an isosceles triangle two sides are eq
    string s3 = "I am a triangle";
    void display()
    {
      cout<<s1<<"\n"<<s2<<"\n"<<s3;
    }
};
int main()
{
   isosceles iso;
   iso.display();
 return 0;
}
```

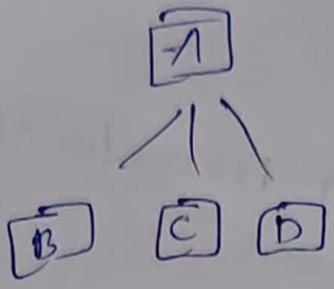
output:

I am an isosceles triangle
In an isosceles triangle two sides are eq
I am an triangle.

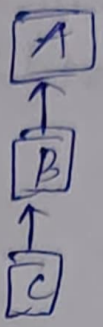
Types of inheritance:



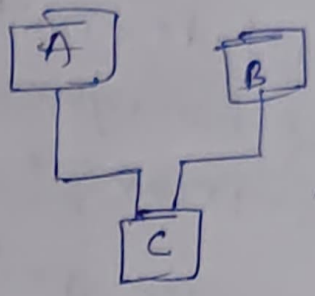
Simple / single



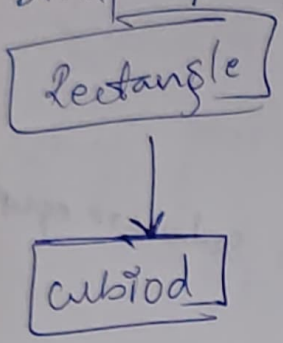
Hierarchical.

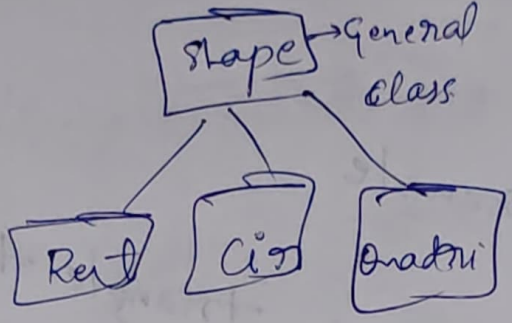


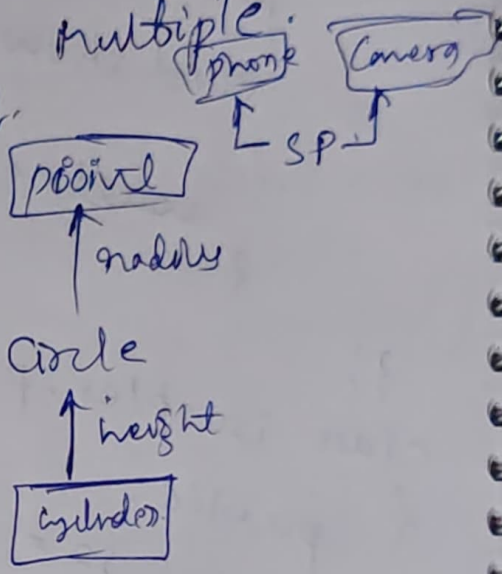
multi level



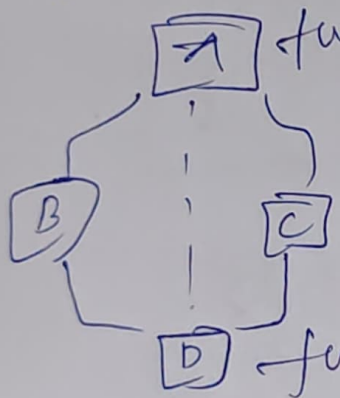
multiple







Hybrid → mix of two.



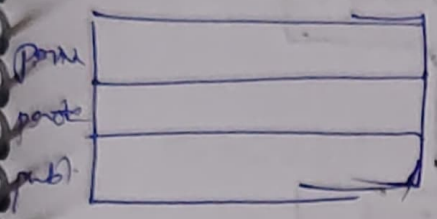
multipath.

→ from this we can remove ambigity

```
class A
{
  ==
};
class B : virtual public A
{
  ==
};
class C : virtual public A
{
  ==
};
class D : public B, public C
{
};
```

Ways of Inheritance

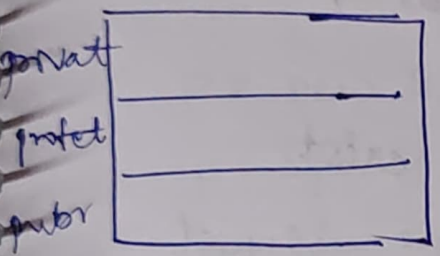
class parent



class Base
{ private : int a;
protected : int b;
public : int c;
};

class child : public parent { protected / private };

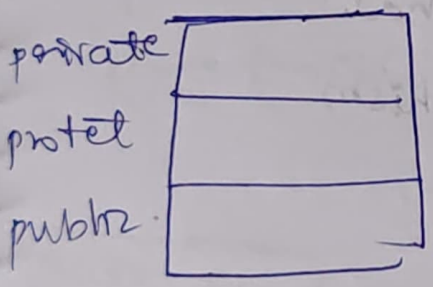
class Derived : privately Base



protet
will be protet
publr
will be publr

private :
void fun()
{
a = 10;
b = 5;
c = 3;
}

class grand child : publr child



};

→ when child class will be private grand child class not acces the grand parent class

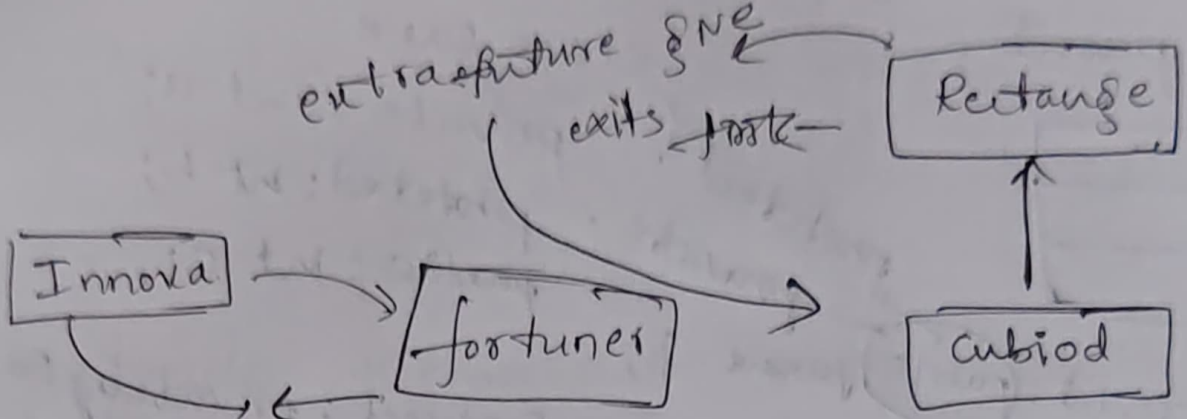
class parent

accesable:
protect
publr
become → protected.

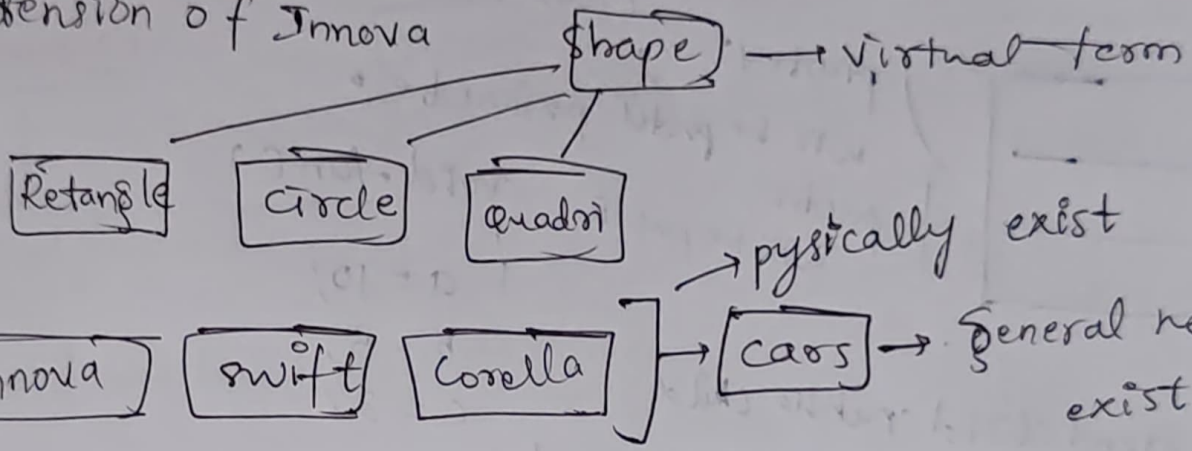
protete & pabli become → private.

class child.

[Generalization vs specialization]



extension of Innova



parent exist and child barrow from that.

- same name different object ⇒ polyporphism

→ active polyporphism,

↓

generalization.

* Write a class for employee

Derived classes

1. Fulltime employee with salary
2. Part time employee with daily wages

Error is occur;

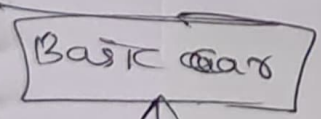
because, member fuction

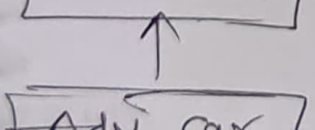
not, no parameter.

Base class pointer Derived class object

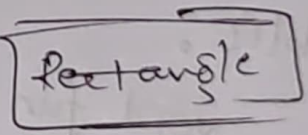
```
int main()
{
    Derived d;
    d.fun1();   (Normal)

    Base *p;
    p = new Derived();
    p->fun1();
    p->fun2();
    p->fun3();
    p->fun4(); X
    p->fun5(); X.
}
```

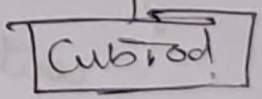
```
class Base
{
public:
    void fun1();
    void fun2();
    void fun3();
};
class Derived : public Ba
{
public:
    void fun4();
    void fun5();
};
```





→ more than basic car





Rectangle *p;
p = new cuboid();

Base *ptr = &d;

if we write function in int main that not present in that is not accessable in.

Rectangle r;

cuboid *q = &r; → cannot initialize a variable of type 'cuboid*'

16/12/2023

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

class Base
{
   public:
      void fun1()
      {
         cout<<"fun1 of Base"<<endl;
      }
};
class Derived: public Base
{
   public:
      void fun2()
      {
         cout<<"fun2 of Derived"<<endl;
      }
};
int main()
{
   Derived d;
   d.fun1();
   d.fun2();
}
```

Base ptr

base b;
derived ~~*~~ptr = &b;

invalid conversion from base to Derived

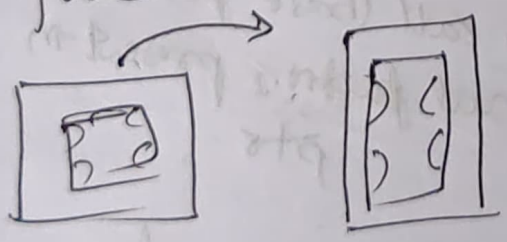
function Overriding:

```
class parent
{ public:
    void display()
    {
      cout << "Display of parent";
    }
};
class child :publi parent
{
  public :
    void display()
    {
      cout << "Display of child";
    }
};
```

```
int main()
{
  parent p;
  p.display();
  → display 4 parent
  child c;
  c.display();
  → display 4 parent
```

redefining

Redifing function of parent class and child class is is function overriding.



→ writting them

* prototype of function must be same

```
class base
{
    public:
    void fun()  →  virtual void fun()
    {
        cout<< "fun of Base"<<endl;
    }
};
class derived : public base
{
    public:
        void fun()
        {
            cout<< "fun of Der
        }
};
int main()
{
    derived d;
    d.fun();
}
```

or

```
derived d;
base *ptr = &d;
ptr->fun() → call (base ptr), means
             call function present in
                  ptr
```

When base class function is defined as virtual and derived d;
~~point~~ base *ptr = &d;
ptr → fun() → call derived ptr object

* Using virtual function, function overriding, base class pointer we can achieve polymorphism.

Polymorphism

```
int main()
{
    car *c = new Innova();
    c->start(); // car base class will be called.
}
```

---

if do base class functions are virtual need to be existance of body

---

```
class swift: public car
{
  public:
    void start()
    {
      cout<<"swift started";
    }
    void stop()
    {
      cout<<"swift stopped";
    }
};
```

```
class car
{
  public:
    void start()
    {
      cout<<"Car started";
    }
    void stop()
    {
      cout<<"Car stopped";
    }
};
class innova: public car
{
  public:
    void start()
    {
      cout<<"Innova started";
    }
    void stop()
    {
      cout<<"Innova stopped";
    }
```

if made base class. function will be virtual that class become virtual not real → than that class will not be exist.

virtual void start() = 0 { pure virtual functions }

→ when assign a class function after made virtual functions than they called pure virtual functions.

{ inheritance allows reusebility }

* Abstract class:

If class have pure virtual functions that it is called.

Abstract class

Base b; X { we can't access as object

Base *c; { we can access as pointer

```
class base
{
    public:
virtual void fun1()
    {
        cout<<"Base fun1";
    }
    void fun2()
    {
        cout<<"Base fun2";
    }
};
class derived : public Base
{
    public:
    void fun2()
    {
        cout<<"Derived fun2";
    }
};
```

* Reusebility
* polyporhism

Base All concrete

↓

Reusebility

Base { Abstract
All pure virtual fn

↓

polymorphism

Base class { Abstract
some concrete
some pure virtual

↓

Reusability
polymorphism

friend function

- Upon object only public member will be access

```
class my
{
  private:
    int a=10
  friend class your;
};
class your
{ public:
    my m;
    void fun()
    {
      cout<<m.a;
    }
};
```

class your →

```
class Test
{
  private:
    int a;
  protected:
    int b;
  public:
    int c;
  friend void fun();
};
void fun()
{
  Test t;
  t.a = 15;
  t.b = 10;
  t.c = 5;
}
```

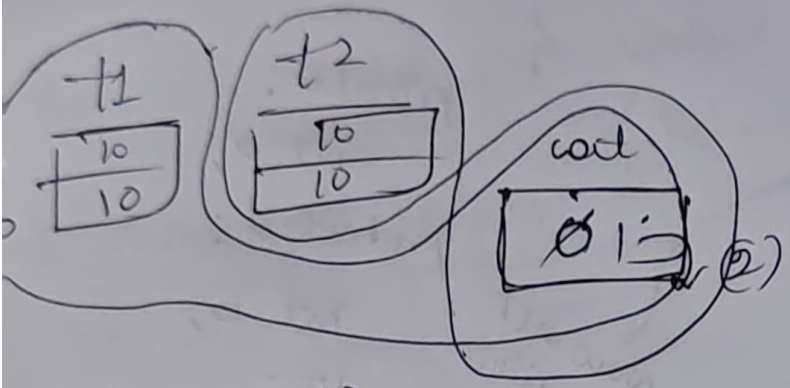
Now all allow

→ to access the private of one class from other class by we object we have to write (friend class)

if access from function, write friend function_name)

if access from class, write friend (class-name)

Static member




```
int main()
{
    Test t1;
    Test t2;
}
```

```
class Test
{
  private:
    int a;
    int b;
  public:
    static int cout;
    Test()
    { a=10
      b=10
    } cout++;
};
```

if made as static it will occupy memory only once

if declare outside and only accessable in the class for that class

```
int test::cout=0;   → global variable
```

test::cout;

```
static int getcount()
{
    a++;
    return cout;
}
```

static function

cannot access object member

only access static member function member.

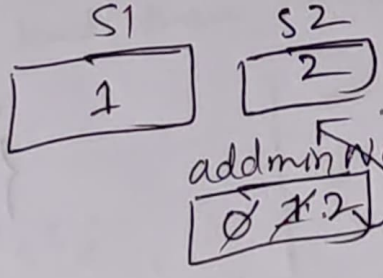
→ if set initial value if don't create objects of that class

* Static member or function belong to class not a object.

```
int main()
{
    cout<< Innova :: getPrice()
    Innova my;
→   cout<< my.getPrice();
```

→ It is belong to class not object.

```
Class Car
{ public:
    static int price;
    car()
    {
    ==
    ==
    ==
    }
    static int getPrice()
    { return price;
    }
};
int car::price=20;
```




```
int main()
{
    (student S1;)
    student S2;
```

roll no

assign serial number.

```
class student
{
    public:
        int rollno;
        static int addminNO;
        student()
        {
            adminNO++;
            rollno = adminNO;
        }
};
int student::addminNO=0;
```

- Static member as used for counter like this
- static members are used as shared memory, like on object wants something and another object need something.

↳ Live Information.

Inner class don't access the outer member of class.

```
class linkedlist
{
    class Node
    {
        int data;
        Node *next;
    };
    Node *Header;
    ==
    ==
    ==
    ==
};
```

```
class outer
{ public:
  ① int a=10;
  ② static int b;
    void fun()
    {
      i.show()
      cout<<i.x;
    }
class inner
{
  public:
    int x=25;
    void show()
    {
      cout<<"show";
    }
};
③ inner i;
int outer::b=20
```

→ we can't create inner obj class object in outer class.

→ outer class can we the objects of inner class

- Inner classes can be static or non-static
- Inner classes can be delared as member of a function, or as members of the outer class.
- Inner classes can be used to create resuable components.

Exception Handling

24/12/2022

Error:

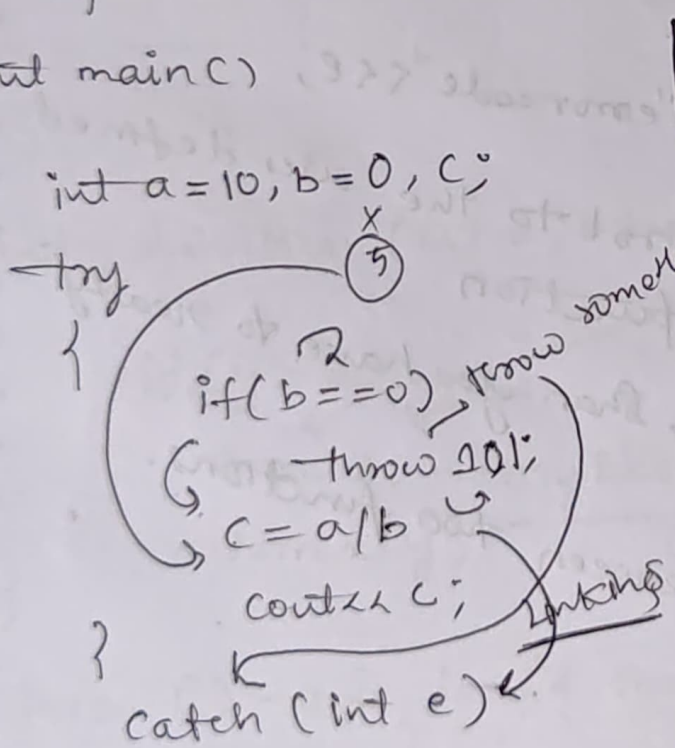
1. syntax error → complie/ compiles
2. Logical error → debugger (to slove logical error)
3. Runtime error →
   - by programmer
   - user → 1. bad input
     2. problem with Resources

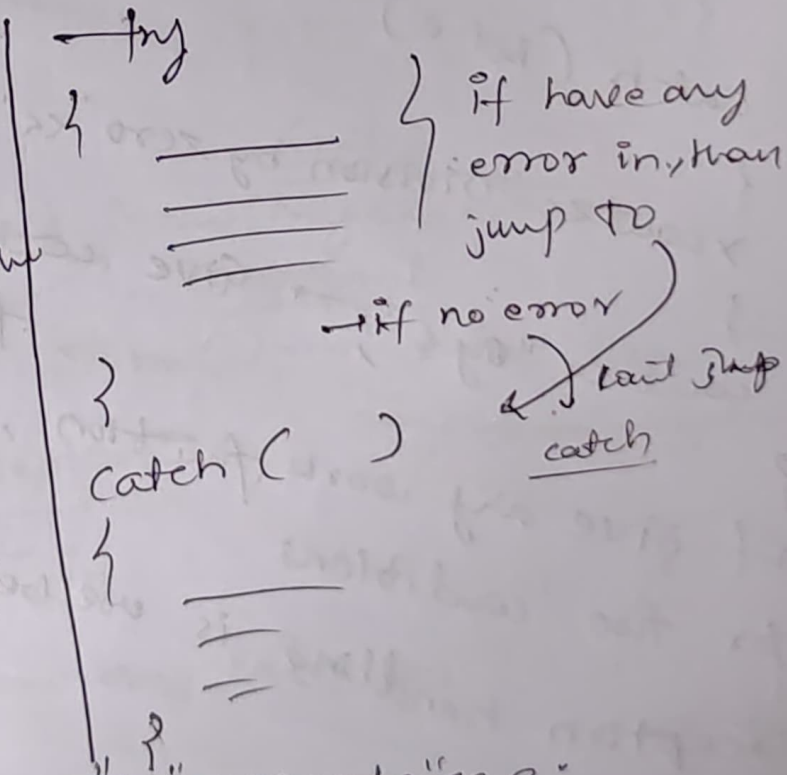
In the type of Runtime error program will be clashes.

→ Runtime errors are called exception.

if input will be bad than suggestion. to give good input

```
int main()
{
    int a=10, b=0, c;
    try
    {
        if(b==0) throw 101;
        c=a/b
        cout<<c;
    }
    catch (int e)
    {
        cout<<"Division by zero"<<"error code"<<e;
    }
    cout<<"Bye";
}
```





→ Bye display always if exception or not.

* The try and catch functions

```
int main()
{
  int a=10, b=0, c;
  try
  {
    1. c= division(a,b);
    2. cout<<c;
  }
  catch (int e)
  {
  cout<<"Division by zero"<<"error code"<<e;
  }
  cout<<"Bye";
}
```

```
int division(int x, int y)
{
  xy
  return xy;
}
if (y==0)
  throw 1;
return xy;
}
```

→ Give control to the user defined function.

If give any work function, than you have to ready for two conditions.

Exception handling is use between two functions.

* All about throw:

```
int division (int x, int y)
{
  if(y==0)
    throw
    return xy;
```

→ int, float, double
→ message also send.

You can throw your class.

```
class myexception : public exception
{
    public :
        class *what() → method for
        {
            return "my exception";
        }
};
```

} Java

```
class myexception : public exception
{
    ========
    ========
    ========
    ========
};
```

exception → But in class in c++

```
int division (int x, int y) throw(myexception)
{
    if(y==0)
        throw 1;
    return x/y;
}
```

(int) → optional

same → myException. ↳ if

throw() → doesn't throw any exception

throw() → It doesn't throw any exception.

```
class Myexception
{


};
int main()
{
    int x=10, y=0, z;
    try
    {
        if(y==0)
            throw Myexception();
        z= x/y;
        cout<<z<<endl;
    catch (MyException e)
    {
        cout<<"Division by zero"<<endl;
    }
}
```

```
int division(int a, int b)
                        throw(MyException)
                              ↓
                           Mention
{                          data type
    if(b==0)
        throw 10;
    return a/b;
}
```

* More about catch & try:

i)

```
try                         catch(int e)
{                           {
  ________  int  ~~~~~~~>       ____
  ________
  ________ float                ____
  ________      \           }
}                \          catch(float e)
                  \-------> {   ____
                                ____
                            }
```

We can have multiple catch block:

```
catch (....) → catch all
{
  ——
  —— → handling all type of exception.

}
```

catch all block is kept in

```
try
{
  ——
  ——
  ——
  try
  {
    ——
    ——
    ——
  }
  catch ()
  {
  }
}
catch ()
{ —
  —
  —
}
```

nested of catch & try

```
class MyException1
{
};
class MyException2:
        public MyException1
{
};
try { ——
      —— }
catch (MyException1 e)
{
  —
}
catch (MyException2 e)
{
  ——
  ——
}
```

you m

You must write catch block for child class and after that write to the parent class

child class → (1)
parent class → (2)

function for double that error will be obtain. That is not catch-function for double.

22/12/2023

Quiz:

IMP

Templates

```
template<class T>
T maximum(T x, T y)
{
    return x>y ? x : y
}
maximum(10,15);
"       (12.5, 9.5
```

```
public:
    void push(int x);
    int pop();
```

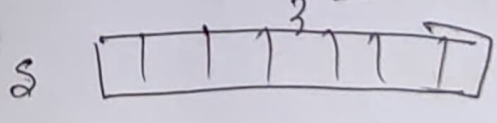
```
template<class T>
    T s[10];
    int top;
public:
    void push(T x)
    T pop();
};
```

```
template<class T, class R>
void add(T x, R y)
{
    cout<<x+y;
}
add(10, 12.4)
```

```
template<class T>
void stack<T>::push(T x)
{
    ——
    ——
    ——
}
template<class T>
T stack<T>::pop()
{
    =
    =
}
```

Template class:

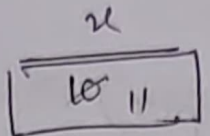
```
class stack
{
private:
    int s[10];
    int top;
```




```
stack<int> S;
stack<float>S2;
```

#define u 10 → not consume memory

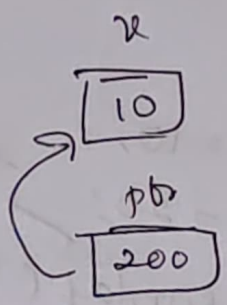
```
int main()
{
const int u=10;  → it will consume memory
   x u++;
   cout<<u;
}
```



if write constant, then that will called constant identifier

→ We also can't modify this data.

```
int main()
{
   int u=10;
constant int *ptr = &u;
   x ++*ptr;
   cout<<u;  → 11
   cout<<*ptr;  → 10
}
```



(or) int const * ptr = &u

pointer can't modify data.

```
int main()
{
   int u=10;
int const *ptr = &u;
   int y=20;
   ptr=&y;
   ++(*ptr);
}
```

const int u =10;

← Reading from Right left, identifire as constant

pointer is locked.
→ can't modify

const int * const ptr = &x → it locks data and pointer also.

× (++(*ptr))
× (ptr = &y)

```
class demo
{
  public:
    int x = 10;
    int y = 20;
    void display() const
    {
      x++;
      cout<<x<<" "<<y<<endl;
    }
};
void fun(int &x, int &y)
{
  x++;
  cout<<x<<" "<<y;
}
int main()
{
  int a=10, b=20;
  fun(a,b)
}
```

→ then y will restricted modifying data members

```
int main()
{
  demo d;
  d.display()
}
```

→ "Read only object" >> 11 20 → ERROR

const → now can't modify

directly accessed.

call by reference

a/x b/y

10	20
11	20

Preprocessor Directives / Macros 24/12/2023

```
# define PI 3.145  → symbolic constants
# define c cout
int main()
{
    cout<<PI;
}
```

3.145 → PI

c << 10

if write this, cout replace by before compile

It is not variable

```
# define SQR(x) (x*x)
int main()
{
    cout << SQR(5);
}
```

(5*5) → before compiling

```
# define MSG(x) #x
int main()
{
    cout << MSG(Hello)
}
```

#x → Like string

"x"

→ "Hello

* we can't redefined macro

```
# ifndef
    # define PI 3.1425
# endif
```

ifndef → If not define

if use condition so enclose with brackets

Namespaces → Removing name constant

```
namespace First
{ void fun()
  {
    cout << "First";
  }
};
```

```
namespace second
{ void fun()
  {
    cout << "second";
  }
};
```

using namespace first → instead that

```
int main()
{
  first::fun();  → first
  second::fun(); → second.
}
```

scope resolution.

→ if write only fun() → error will be occured

accessing the namespace → using.

* **Destructor:**

```
main()
{
  Test t;
  Test *p = new Test();
  delete P;
}
```

```
class Test
{
  public:
    Test()
    { cout << "Test creat";
    }
    ~Test()
    { cout << "Test destroyed";
    }
};
```

constructor is called when object is constructing, destructor is called when object is destroyed.

Use of destructor: Use for deallocated [illegible]

```
class Test
{
    int *p;
    ifstream fis;

    Test()
    {
        p=new int[10];
        fis.open( )
    }
    ~Test()
    {
        delete[]p;
        fis.close();
    }
};
```

can't have overload the destructors.

When create object on heap dynamically that that time constructor will be called.

delete p;
↳ destructor is called

can't delete memory atomatically, so for that we use delete []p.
↳ realise the memory.

→ if we don't use destructor than Memory leak problem will be occured.

```
class Base
{ public:
    Base()
    { cout<<"Base constructor";
    }
virtual ~Base()
    { cout<<"Base destructor";
    }
};
```

* Destructor:
↳ member function of class

```
int main()
{   derived d;
    ;
    ;
  - Derived destructor
  - Base des~~
}
```

Base cont
derived

```
class Derived : public Base
{ public:
      Derived()
      { cout<<"Derived constructor";
      }
     ~Derived()
      { cout<<"Derived Destructor";
      }
};
```

Virtual destructor:

calling of destructor by base pointer.

```
int main()
{
    Base *p = new Derived()
    ;
    delete p;
}
```

first call Base class destructor

Base class pointer

Base destructor

↳ if write virtual keyword, first call derived class to destruct.

Message will be showed → upon call of functions.

* Destructor is called just before the end of object life.

27/12/2023

Streams:

flow data from one place to another place.