Constructors - 1.) Default Construct — or have no arguments passed through them.

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

class Complex;
{
  int a, b;

public:
  // Creating a Constructor
  // Constructor is a special member func" with the same name as of the class.
  // It is used to initialize the objects of its class
  // It is automatically invoked whenever an object is created.
                    pass → parameter
  Complex(void); // Constructor declaration

  void printNumber()
  {
    cout << "Your number is " << a << " + " << b << "i" << endl;
  }
};

Complex :: Complex(void) // --> This is a default constructor as it takes no parameters
{
  a = 10;
  b = 20;
}
```

```
int main()
{
    Complex C1, C2, C3;
    C1.printNumber();
    C2.printNumber();
    C3.printNumber();

    return 0;
}

// Properties of Constructor

/* 1. It should be declared in the public section
      of the class.
   2. They are automatically invoked whenever the
      object is created.
   3. They cannot return values and do not have
      return types.
   4. It can have default arguments
   5. We cannot refer to their address
*/
```

Note: if we don't initialize then by default compiler will set garbage value

2. Parameterized Constructor

have arguments passed through them.

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

class Complex
{
    int a, b;

public:
    Complex(int, int); // Constructor declarati

    void printNumber()
    {
        cout << "Your no. is " << a << " " << b << "i." << endl;
    }
};

Complex::Complex(int x, int y) // --> This is a parameterized constructor as it takes two parameters
{
    a = x;
    b = y;
    // cout << "Hello World";
}

int main(){
    // Implicit call
    Complex a(4, 6);
```

→ Just mention argument data types. no need to write variable name.

a+bi

// Explicit Call

* Complex b = Complex (5, 7);

```
a.printNumber();
b.printNumber();

return 0;
}
```

O/P

4 + 6i
5 + 7i

Constructor Overloading (Multiple constructors)

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

Class Complex
{
  int a, b;

 public:
1.) Complex (int x, int y)
  {
    a = x;
    b = y;
  }
```

2.
```
Complex (int x) {
    a = x;
    b = 0;
}

void printNumber ()
{
    cout << "Your no. is" << a << " + " << b << "i"
        << endl;
}
};

int main ()
{
    Complex c1 (4, 6);
    c1.printNumber ();

    Complex c2 (5);
    c2.printNumber ();
    return 0;
}
```

Output

```
Your no is 4 + 6i
Your no is 5 + 0i
```

Constructors with default arguments

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

class Simple{
private:
    int data1;
    int data2;
public:
    Simple(int a, int b=9){
        data1 = a;
        data2 = b;
    }

    void printData();
};
```

By default private A.S. in C++.

note

RT CN :: fn(){

```
void Simple::printData(){
    cout<<" The value of data is "<<data1<<" and
" <<data2<<endl;
}

int main(){
    Simple S(1,4);
    S.printData();
    Simple S1(1);
    S1.printData();
    return 0;
}
```

output

The value of data is 1 and 4
The value of data is 1 and 9

→ runtime par

Dynamic Initialization of Objects Using Constructors :-

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

class BankDeposit {
    int principal;
    int years;
    float interestRate;
    float returnValue;

public:
    BankDeposit () { }          ← Note: It's must to make a default constru
    BankDeposit (int p, int y, float r); // r can be
                                    a value like 0.
    BankDeposit (int p, int y, int r); // r can be
    void show();                    a value like 14.
};
```

CN :: CON C)

```
BankDeposit :: BankDeposit (int p, int y, float r)
{
    principal = p;
    years = y;
    interestRate = r;
    returnValue = 0;
    for (int i = 0; i < y; i++)
    {
        returnValue = returnValue * (1+r);
    }
}
```

```
BankDeposit :: BankDeposit (int p, int y, int r)
{
  principal = p;
  years = y;
  interestRate = float(r)/100;
  returnValue = principal;
  for (int i = 0; i < y; i++)
  {
    returnValue = returnValue + (1 + interestRate);
  }
}

void BankDeposit :: show(){

  cout << endl << "Principal amount was " <<
  principal << endl << ". Return value after " << years
  << " is " << returnValue << endl;
}

int main(){
  BankDeposit bd1, bd2, bd3;
  int p, y;
  float r;
  int R;

Input line
  cout << "Enter the value of p y and r" << endl;
  cin >> p >> y >> r;
```

→ bd2 = BankDeposit(p, y, R);
bd2.show();

→ bd1 = BankDeposit(p, y, r);
bd1.show();
return 0;
}

Output

→ Enter the value of p, y and r
100
1
0.03

→ Principal amount was 100. Return value after 1 year is 103.
Enter the value of p y and R

→ bd3.show(); // possible because (BankDeposit(){}) if not there compiler will show error.

→ Principal amount was 1989829969. Return value after 8 years is 1.556e+033

Copy Constructor

Syntax: CN(CN &oldobj);

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

Class Number {
int a;
public:
   Number(){ // default constructor
      a=0;
   }

   Number(int num){ // parameterize constructor
      a = num;
   }
Note
// When no copy constructor is found, compiler supplies it own copy constructor
   Number(Number &obj){
      cout<<"Copy constructor called!!!"<<endl;
      a = obj.a;
   }
   void display(){
      cout<<"The no. for this object is "<<a<<endl;
   }
};

int main(){
   Number x, y, z(45); // z=a;
   x.display();
   y.display();
   z.display();
```

2 → Z2 = z; // copy constructor not called
z2.display(); Simp assignment

1 → Number z1(z); // Copy Constructor is invoked
z1.display();
// z1 should exactly resemble z or x or y
return 0;

Output

The no. for this object is 0
The no for this object is 0
The no. for this object is 45
Copy Constructor called!!!
The no. for this object is 45

Number → CN
Number Z3 = z;

Number
→ Z3 = z; // Copy constructor invoked.
z3.display();
Output (123)
The no. for this object is 0
" " " " " 0
" " " " 45
Copy Constructor called!!!
The no. for this object is 45
" " " " 45
Copy constructor called!!!
The no. for this object is 45

Note
Jo vi kux construct or के अंदर होगी वो ही print hojayega

Imp:-

Z2 = z; CC not invoked	ClassName Z2 = z; CC invoked;

Destructors

```
#include<iostream>
using namespace std;
// Destructor never takes an argument nor does it return any value
int count=0;

class num{
    public:
    num(){
    count++;
    cout<<"This is the time when constructor is called for object no."
    <<count<<endl;
    }
    ~num(){
    cout<<"This is the time when my destructor is called for object no:"
    <<count<<endl;
    count--;
    }
};

int main()
{
cout<<"We are inside our main function"<<endl;
```

tilde sign → ~num() { (constructor) class name

Block → ~~to~~ Andar jo chizen banegi block khatam hone pe destroy hojayegi.

```
cout << "Creating first object n1 " << endl;
num n1;
{
cout << "Entering this block" << endl;
cout << " Creating two more objects" << endl;
num n2, n3; ← Scope
cout << "Exiting this block" << endl;
}
cout << " & Back to main " << endl;
return 0;
```

	constructor called	destructor called (reverse)
Output:	Order: 1 > 2 > 3	Order: 3 > 2 > 1

We are inside our main function
Creating first object n1
This is the time when constructor is called for object number 1
Entering this block
Creating two more objects
This is the time when constructor is called for object number 2
This is the time when constructor is called for object number 3
Exiting this block
This is the time when my destructor is called for object number 3
This is the time when my destructor is called for object number 2
Back to main

This is the time when my destructor is called for object number 1

Inheritance in C++ - Overview

- Reusability is a very imp. feature of OOPs
- In C++ we can reuse a class and add additional features to it
- Reusing classes saves time and money
- Reusing already tested and debugged class will save a lot of effort of developing and debugging the same thing again

```
class Employee {
name
Age
DOB
}
```

→ Programmer
→ Manager
→ etc

DRY → Do not Repeat Yourself

What is Inheritance in C++

→ The concept of Reusability in C++ is

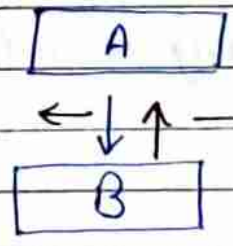
supported using Inheritance

- We can reuse the properties of an existing class by inheriting from it
- The existing class is called as the Base Class.
- The new class which is inherited is called as the Derived Class
- Reusing classes saves time and money
- There are different type of inheritance in C++

Forms of Inheritance in C++

Single Inheritance

- A derived class with only one Base Class

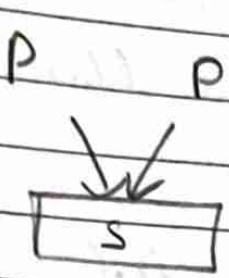


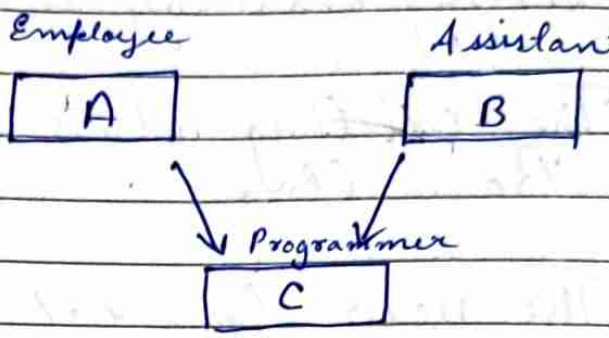
(B bana A se) derives ← ↓ ↑ — is derived from

gene flow ↓

Multiple Inheritance in C++

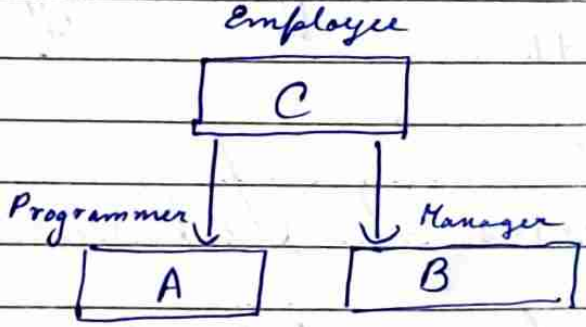
- A derived class with more than one Base class





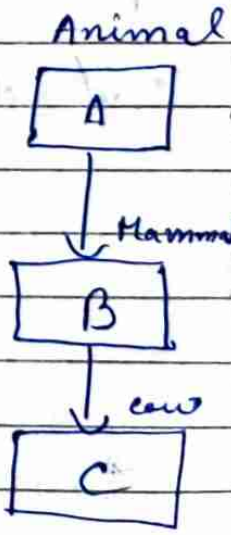
Hierarchial Inheritance in C++

- Several derived classes from a single base class



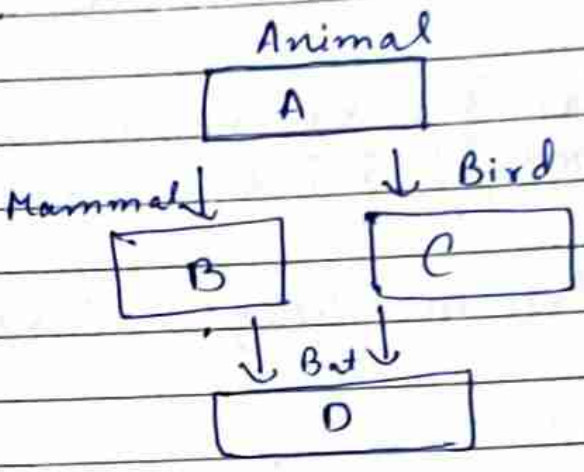
Multilevel Inheritance in C++

- Deriving a class from already derived class



Hybrid Inheritance in C++

- Hybrid Inheritance is a combination of multiple inheritance and multilevel inheritance.
- A class is derived from two classes as in multiple inheritance
- However, one of the parent classes is not a base class.



Inheritance Syntax & Visibility Mode

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

//Base class
class Employee{
```

```
public:
int id;
float Salary;
    Employee (int inpId){
        id = inpId;
        Salary = 34.0
    }
    Employee(){}
};
```

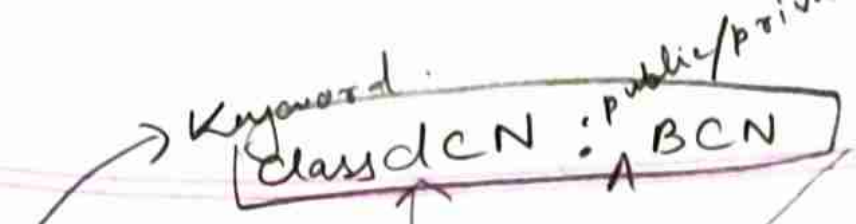
// Derived Class Syntax

/* class {{ derived-class-name }} : {{ visibility-mode }} {{ base-class-name }}
{
  class members/methods/etc...
}

Note: Visibility Modes :—

1. Default visibility mode is private
2. Public visibility Mode : Public members of the base class becomes Public members of the derived class
3. Private visibility Mode : Public members of the base class become Private members of the derived class.
4. Private members are never inherited

*/



// Creating a Programmer class derived from Employee Base class

```
class Programmer : Employee {
public:
Programmer(int inpId){
    id = inpId;
}
int languageCode = 9;
void getData(){
cout << id << endl;
};
```

Derived class

```
int main(){
Employee harry(1), rohan(2);
cout << harry.salary << endl;
cout << rohan.salary << endl;
Programmer skillF(10);
cout << skillF.id;
cout << skillF.languageCode << endl;
skillF.getData();
return 0;
}
```

Note:- → error because inherited privately can't be inherited publically.

Destructors :-

- Destructor is an instance member func^{n} of a class
- The name of the destructor is same as the name of a class but preceded by tilde (~) symbol
- Destructor can never be static
- Destructor has no return type
- Destructor takes no argument (No overloading is possible
- It is invoked implicitly when object is going to destroy.

```
#include<conio.h>
#include<iostream.h>
  class Complex
{
   private:
    int a, b;
  public:
    ~Complex()
     { cout<<"Destructor";}
  };
```

```
void fun()
{
  Complex obj;
}
void main()
{
  clrscr();
  fun();
  getch();
}
```

→ Why destructor?

- It should be defined to release resources allocated to an object.
- Only Destructors can be virtual.

Single Inheritance:-

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

Class Base {
   int data1; // private by default and is not
                 inheritable
 public:
   int data2;
   void setData();
   int getData1();
   int getData2();
};

void Base :: setData(void){
      data1 = 10;
      data2 = 20;
}

int Base :: getData1(){
      return data1;

}

int Base :: getData2(){
   return data2;
}
```

```
class Derived : public Base { // Class is being derived
                                 publically
  int data3;
  public:
     void process();
     void display();
};

void Derived :: process() {
     data3 = data2 * getData1();
}

void Derived :: display() {
  cout << "Value of data 1 is " << getData1() << endl;
  cout << "Value of data 2 is " << data2 << endl;
  cout << "Value of data 3 is " << data3 << endl;
}

int main() {
   Derived der;
  der.setData();
  der.process();
  der.display();

   return 0;
}
```

Output

```
Value of data 1 is 10
Value of data 2 is 20
Value of data 3 is 200
```

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

class Base {
  int data1;
  public:
    int data2;
    void setData();
    int getData1();
    int getData2();
};

void Base :: setData(void){
  data1 = 10;
  data2 = 20;
}

int Base :: getData1(){
    return data1;

int Base :: getData2(){
    return data2;

class Derived : private Base {
    int data3;
    public:
        void process();
        void display();
};
```

```
void Derived :: process(){
    setdata();
    data3 = data2 * getData1();
}

void Derived :: display(){
    cout << "Value of data1 is " << getData1() << endl;
    cout << "Value of data2 is " << data2 << endl;
    cout << "Value of data3 is " << data3 << endl;
}
```

getData2();

Same output as before!!!

Protected Access Modifier

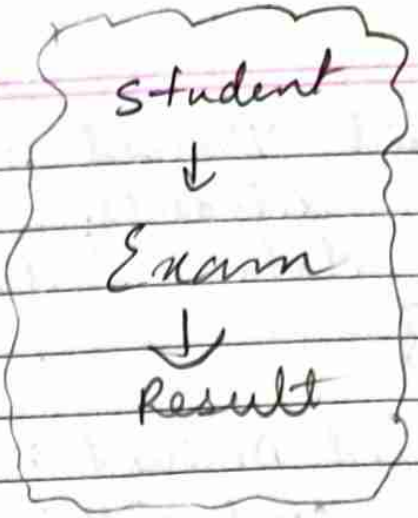
→ private wha lekin inherit hojaye → protected.

For a protected member :- A . M .

Derivation →

Base class / Members	Public	Private	Protected
1. Private	Not inherited	Not inherited	Not inherited
2. Protected	Protected	Private	Protected
3. Public	Public	Private	Protected

Multilevel Inheritance




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

class Student {
    protected:
        int roll_number;
    public:
        void set_roll_number(int); // declaration
        void get_roll_number(void);
};

void Student :: set_roll_number(int r){
  roll_number = r;
}

void Student :: get_roll_number() {
  cout << "The roll number is " << roll_number << endl;
}

class Exam : public Student {
    protected:
        float maths;
        float physics;

    public:
        void set_marks(float, float); // declaration
        void get_marks(void);
};
```

```
void Exam :: set_marks(float m1, float m2){
  maths = m1;
  physics = m2;

void Exam :: get_marks(){
  cout << "The marks obtained in maths are:"
  << m1 << endl;
  cout << "The marks obtained in physics are:"
<< m2 << endl;
}

class Result: public Exam{
  float percentage;
  public:
    void display(){
      get_roll_number();
      get_marks();
      cout << "Your percentage is " << (maths+physics)
[contd] → /2<<endl;
    }
};
  int main(){
  Result harry;
  harry.set_roll_number(495);
  harry.set_marks(94.0, 90.0);
  harry.display();
  return 0;
  }
```

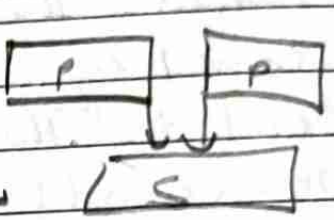
Output

The roll number is 495
The marks obtained in maths are: 94
The marks obtained in physics are 90.0
Your percentage is 92%

Notes If we are inheriting B from A and C from B : [A ---> B ---> C]

1. A is the base class for B and B is the base class for C.
2. A --> B --> C is called Inheritance Path

Multiple Inheritance




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

// Syntax for inheriting in multiple inheritance
// Class DerivedC : visibility-mode base1, visibility-mode
        base2 {
// Class body of class "DerivedC"
// };
class Base1 {
protected:
  int baseint;
public:
  void set_baseint(int a)
  {
    baseint = a;
  }
};
```

```
class Base2{
protected:
  int base2int;
public:
  void set_base2int(int a)
    {
      base2int = a;
    }
};

class Derived : public Base1, public Base2
  {
    public:
      void show(){
      cout << "The value of Base 1 is " << base1int << endl;
      cout << "The value of Base 2 is " << base2int << endl;
      cout << "The sum of these values is " <<
        base1int + base2int << endl;
    }
  };

/*
The inherited derived class will look something like
  this:
Data members:
    base1int --> protected
    base2int --> protected
```

Member functions:
Set_base1int() → public
Set_base2int() → public
Set_Show() → public
*/

output
The value of Base1 is 29
The value of Base2 is 1
The sum of these values is 30

```
int main()
{
    Derived harry;
    harry.set_base1int(29);
    harry.set_base2int(1);
    harry.show();

    return 0;
}
```

Note : You can add ~~another~~ more Base class(es) → Base 3 ++

Ambiguity Resolution :-

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

class Base1 {
    public:
        void greet(){
            cout << "How are you?" << endl;
        }
};
```

```
class Base2 {
public:
    void greet()
    {
      cout << " Namaste " << endl;
    }
};

class Derived : public Base1, public Base2 {
    int a;
};
```

→

```
public:
    void greet() {
        Base1 :: greet();
    }
    CN :: FN();
```

Yahan ambiguity resolve karani hoti hai

```
int main() {
    Base1 base1obj;
    Base2 base2obj;
    base1obj.greet();
    base2obj.greet();
    Derived d;
    d.greet();
    return 0;
}
```

→ ambigous Error

To resolve this error

Output

How are you?
Namaste
How are you?

Manually Ambiguity resolve krteh.

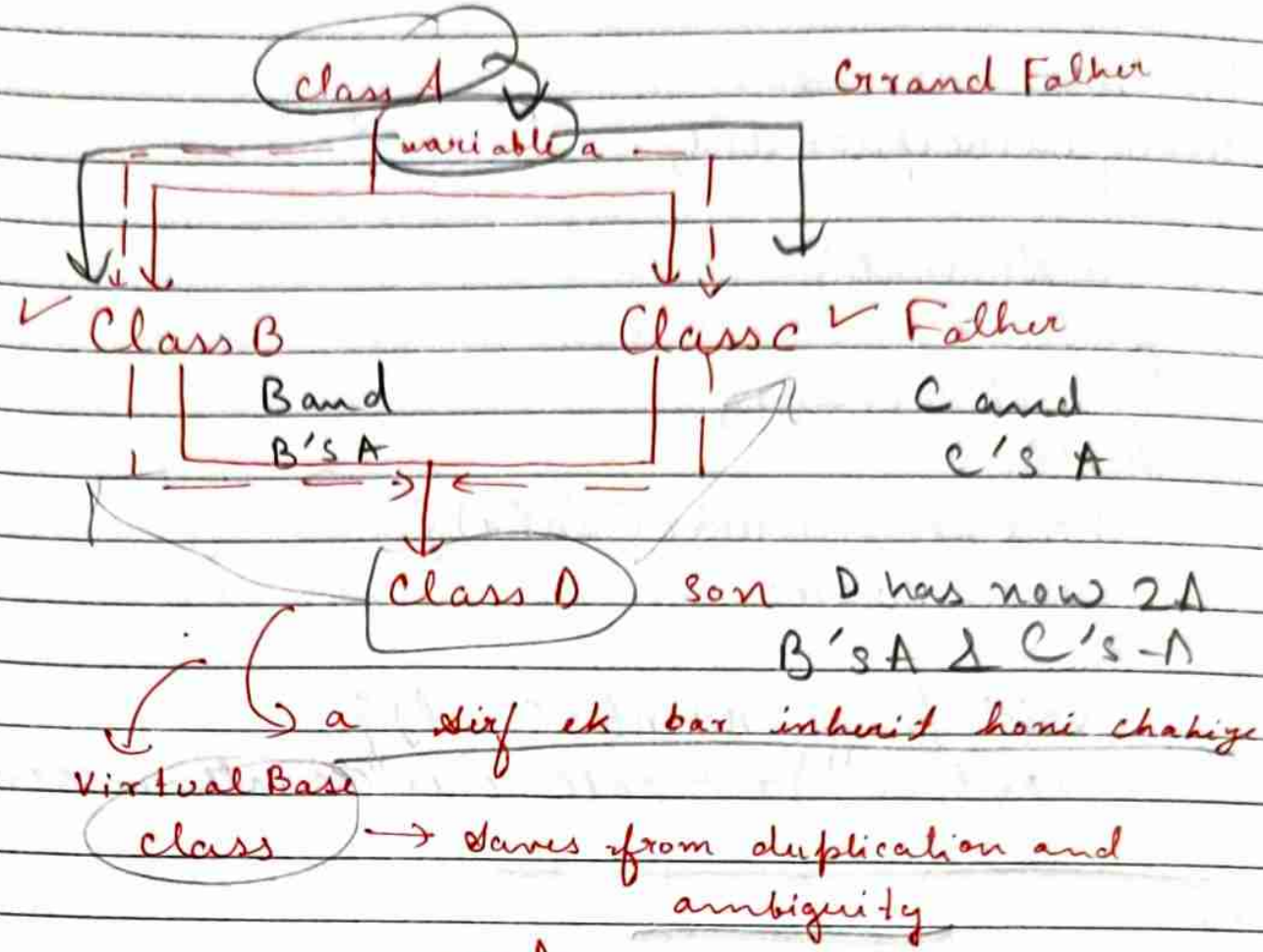
override

Ambiguity 2 → khud resolve hoti hai

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

class B {
  public:
    void say () {
      cout << "Hello World" << endl;
    }
};
// D's new say () method will override base's class
                                    say () method
class D : public B {
  int a;
  public:
    void say ();
    {
      cout << "Hello my beautiful people" << endl;
    }
};
int main () {

  B b;
  b.say ();

  D d;
  d.say ();

  return 0;
}
```

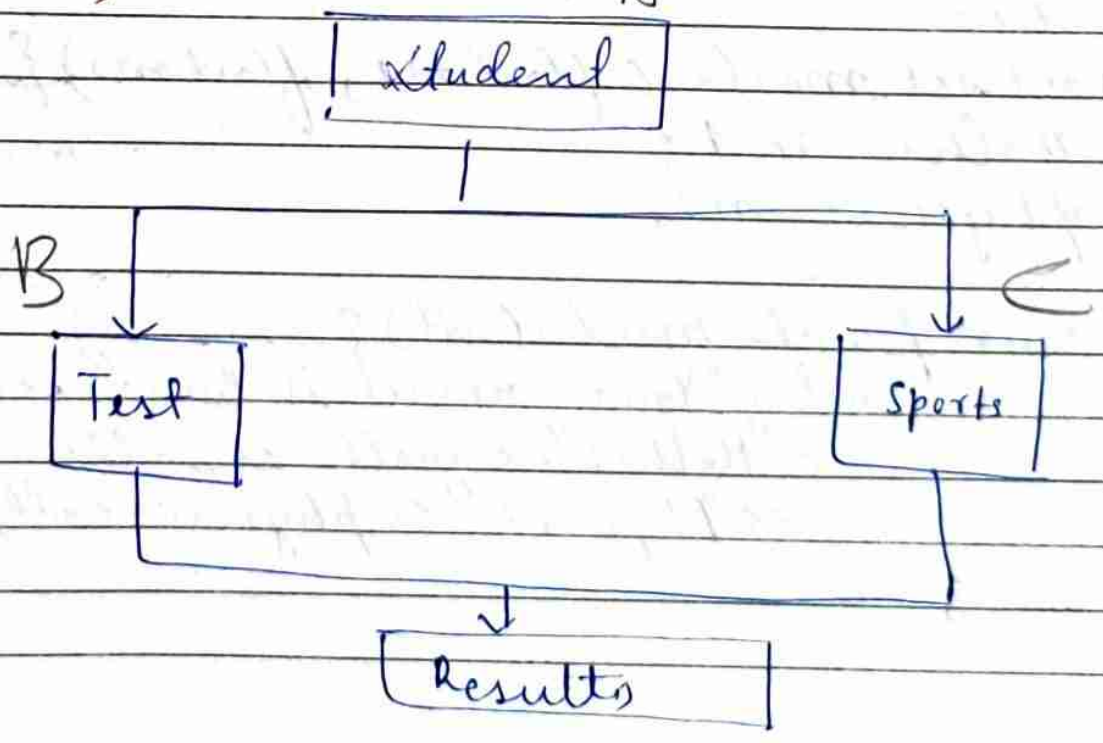
Output
Hello world
Hello my beautiful people

Virtual Base Class



class B : virtual public A
{
...
}

class C : virtual A {



Code:

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

Class Student {
protected:
    int roll_no;
public:
    void set_number(int a){
    roll_no=a;
    }
    void print_number(void){
    cout<<"Your roll no is "<<roll_no<<endl;
    }
};

Class Test: virtual public Student{
protected:
    float maths, physics;
public:
    void set_marks(float m1, float m2){
    maths = m1;
    physics = m2;
    }
    void print_marks(void){
        cout<<"Your result is here:"<<endl.
            <<"Maths:"<<maths<<endl.
            <<"Physics:"<<physics<<endl;
    }
};
```

```
class Sports : virtual public Student {
  protected:
    float score;
  public:
    void set_score (float sc) {
        score = sc;
    }
    void print_score (void) {
    cout << "Your PT score is " << score << endl;
    }
};

class Result : public Test, public Sports {
  private:
    float total;
  public:
    void display (void) {
        total = maths + physics + score;
        print_number();
        print_marks();
        print_score();
        cout << "Your total score is: " << total << endl;
    }
};
```

```
int main (){
    Result harry;
    harry . set_number(4200);
    harry . set_marks (78.9, 99.5);
    harry . set_score(9);
    harry . display ();
    return 0;
}
```

Output

```
Your roll no is 4200
Your result is here:
Maths : 78.9
Physics : 99.5
Your PT score is 9
Your Total score is: 187.4
```

Constructors in Derived Class

→ We can use constructors in derived classes in C++

→ If base class constructor does not have any arguments, there is no need of any constructor

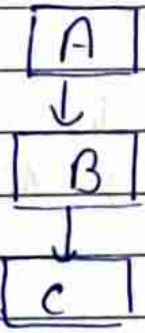
→ But if there are one or more arguments in the base class constructor, derived class need to pass arguments to the base class constructor

→ If both base and derived classes have constructors, base class constructor is executed first Exec: BC → DC

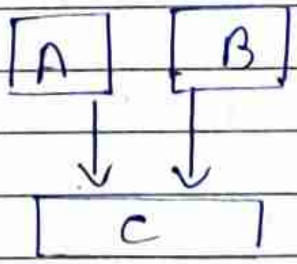
Constructors in Multiple Inheritance

→ In multiple inheritance, base classes are constructed in the order in which they appear in the class declaration

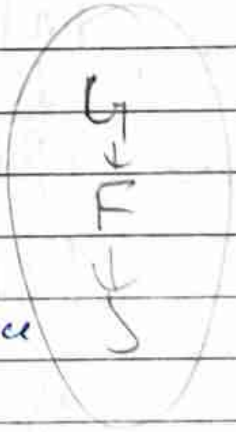
→ In multilevel inheritance, the constructors are executed in the order of inheritance



Multilevel



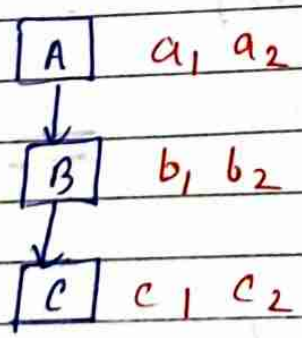
Multiple Inheritance



Special Syntax

- C++ supports an special syntax for passing arguments to multiple base classes.
- The constructors of the derived class receives all the arguments at once and then will pass the calls to the respective base classes.
- The body is called after all the constructors are finished executing.

Derived-Constructor(arg1, arg2, arg3, ...) : Base1-Constructor(arg1, arg2), Base2-Constructor(arg3, arg4) {
...
} Base1-Constructor(arg1, arg2)



C(a1' a2' b1' b2' c1' c2'): B(b1', b2'), A(a1', a2') { c1 = c1

C obj(3, 7, 9, 11, 12, 22)

V.I.P.

Special Case of Virtual Base Class

- The constructor for virtual base classes are invoked before any non virtual base class
- If there are multiple virtual base classes, they are invoked int the order declared.
- Any non-virtual base class are then constructed before the derived class constructor is executed

Code

```
#include<iostream>
using namespace std;
/*                                  Order :-
Case 1:
Class B: public A {
// Order of execution of constructor -> first A()
   then B()
};

Case 2:
Class A : public B, public C{
// Order of execution of constructor -> B() then C
C() and A()
};
```

constructor's order
virtual
Public → पहले वहां पहले
derived

Case 3:

```
Class A: public B, virtual public C {
// Order of execution of constructor -> C()
  then B() and A()
  };

*/
class Base1 {
  int data1;
  public:
    Base1(int i) {
      data1 = i;
      cout << "Base 1 class constructor called" << endl;
    }
    void printData(void) {
      cout << "The value of data1 is " << data1 << endl;
    }
};

class Base2 {
    int data2;
    public: Base2(int i) {
      data2 = i;
      cout << "Base2 class constructor called" << endl;
    }
    void printData(void) {
      cout << "The value of data2 is " << data2 << endl;
    }
};
```

Multiple inheritance

```
class Derived : public Base1, public Base2 {
    int derived1, derived2;
  public:
    Derived(int a, int b, int c, int d) : Base1(a)
      , Base2(b) {
        derived1 = c;
        derived2 = d;
        cout << "Derived class constructor called" << endl;
    }
    void printData(void)
    {
        cout << " The value of derived1 is " <<
        derived1 << endl;
        cout << " The value of derived2 is " <<
          derived2 << endl;
    }
};

int main() {
    Derived harry(1, 2, 3, 4);
    harry.printDataBase1();
    return 0;
}
```

Output

```
Base 1 class constructor called
Base 2 class constructor called
Derived class constructor called
The value of data1 is 1
```

```
#include <iostream>
using namespace std;
/*
Create 2 classes:

1. SimpleCalculator - Takes input of 2 numbers using a utility function and performs +, -, *, / and displays the results using another function.

2. ScientificCalculator - Takes input of 2 numbers using a utility function and performs any four Scientific operation of your choice and displays the result using another function.

Create another class HybridCalculator and inherit it using these 2 classes.
Q1. What type of Inheritance are you using?
Q2. Which mode of Inheritance are you using?
Q3. Create an object of HybridCalculator and display results of Simple and Scientific calculator.
Q4. How is code reusability Implemented?

*/
```

Initialization list in Constructors

```
#include<iostream>
using namespace std;
/*
Syntax for initialization list in constructor:
constructor(argument-list):initialization-
section
{
  assignment + other code;
}

class Test {
    int a;
    int b;
    public:
      Test(int i, int j): a(i), b(j) {constructor
        -body}
};
*/

                    // Test(int i, int j): a(i), b(j)
class Test          // Test(int i, int j): a(i), b(i+j)
{                   // Test(int i, int j): a(i), b(a+j)
    int a;          // Test(int i, int j): b(j), a(i+b)
    int b;          Note | → Red flag because a will be
                         | initialized first.
                         | ↳ Order dekha jata h.
public:
  Test(int i, int j): a(i), b(j)
  {
```

fair n lovely

```
    cout << " Constructor executed " << endl;
    cout << " Value of a is " << a << endl;
    cout << " Value of b is " << b << endl;
  }
};

int main()
{
  Test t(4, 6);
  return 0;
}
```

Output
Constructor executed
value of a is 4
value of b is 6

Revisiting Pointers new and delete

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

int main() {
  // Basic Example
  int a = 4;
  int * ptr = &a;
  cout << " The value of a is " << *(ptr) << endl;

  *ptr = 999;
  o/p: The value of a is 999.

  // new keyword
  // int * p = new int(40);
  float * p = new float(40.78);
  cout << " The value at address p is " << *(p) << endl;
```

4

40.78

// dynamically allocated array

```
int * arr = new int[3];
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
delete [], arr;
cout << " The value of arr[0] is " << arr[0]
    << endl;
cout << " The value of arr[1] is " << arr[1];
<< endl;
cout << " The value of arr[2] is " << arr[2].
    << endl;
return 0;   // delete operator
}
```

frees space for dynamically allocated array

output

```
The value of a is 4
The value at address p is 10.98
The value of arr[0] is 15601336
The value of arr[1] is 15577760
The value of arr[2] is 30
```

```
int * arr = new int[3];
arr[0] = 10;
arr[1] = 20;
arr[2] = 30;
```

```
int *arr = new int[3];
arr[0] = 10;
*(arr+1) = 20;
arr[2] = 30;
```

Pointers to Objects and Arrow Operator

कभी भी Arrow Operator use करते हैं तो Pointer ही use करते हैं।

→ Pointers से object के members को कैसे access कर सकते

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

class Complex {
    int real, imaginary;
public:
    void getData() {
        cout << " The real part is " << real << endl;
        cout << " The imaginary part is " << imaginary << endl;
    }
    void setData(int a, int b) {
        real = a;
        imaginary = b;
    }
};
```

→ * लगा के dereference होजाता है

→ & address of होजाता है

→ new operator से object बनजाता

```
int main() {
    // Complex C1;
    // Complex * ptr = &C1;
    Complex * ptr = new Complex;
    (*ptr).setData(1, 54);
    (*ptr).getData();
    return 0;
}
```

output
The real part is 1
The imaginary part is 54

```
int main() {
  // Complex c1;
  // Complex * ptr = &c1;
Complex * ptr = new Complex;
//(*ptr).setData(1, 54); is exactly same
  as ptr -> setData(1,54);

//(*ptr).getData(); is as good as
  ptr->getData();
  return 0;
}
```

```
  // Array of objects
  Complex *ptr = new Complex[4];
  ptr1 -> setData(1, 4);
  ptr1 -> getData();
  return 0;
}
```

output
The real part is 1
The imaginary part is 4

index
↓
0 *ptr से point करेगा पहले वाले को
1 *ptr+1 point " दूसरे "
2 *ptr+2 point " 3rd को
3 *ptr+3 point 4th को

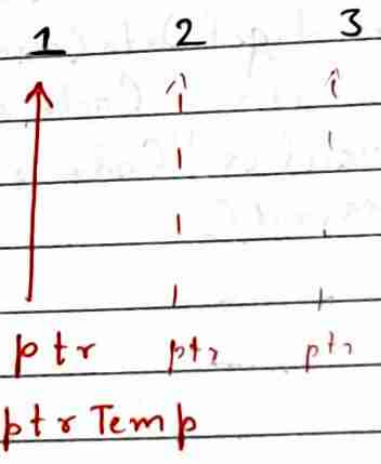
↳ Array banaye hai 4 objects के
जो के Point कर रहे हैं

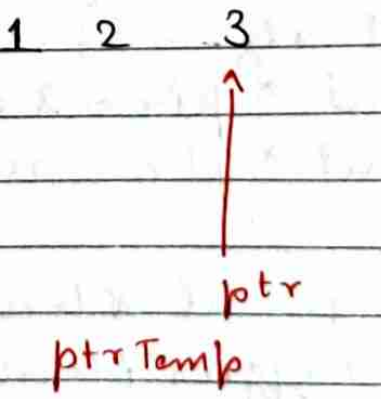
Array of Objects Using Pointers :-

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

class shopItem
 {
    int id;
    float price;
    public:
      void setData(int a, float b){
        id = a;
        price = b;
    }
      void getData(void){
      cout<<" Code of this item is "<<id<<endl;
      cout<<" Price of this item is "<<price
      <<endl;
    }
};

  int main(){
    int size = 3;
  // int * ptr = &size;
  // int * ptr = new int [3*3];

  // 1. general store item
  // 2. veggies item
  // 3. hardware item
```

```
ShopItem *ptr = new ShopItem[size];
→ ShopItem *ptrTemp = ptr;
int p;
float q;
int i;
for(int i=0; i<size; i++)
{
  cout<<"Enter Id and price of item"<<
    i+1<<endl;
  cin>>p>>q;
  //(*ptr).setData(p,q);
  ptr->setData(p,q);
  ptr++;
}
```

```
for(i=0;i<size;i++)
{
  cout<<"Item number:"
    <<i+1<<endl;
  ptrTemp->getData();
  ptrTemp++;
}

  return 0;
}
```





output

```
Enter Id and price of item1
1001
1.1
Enter Id and price of item2
1002
1.2
```

Enter Id and price of item 3
1003
3.3
Item number: 1
Code of this item is 1001
Price of this item is 1.1
Item number: 2
Code of this item is 1002
Price of this item is 1.2
Item number: 3
Code of this item is 1003
Price of this item is 3.3

This Pointer :-

→ The 'this' pointer is used to retrieve the object's x
→ hidden by the local variable x

when local variable is same as global variable.

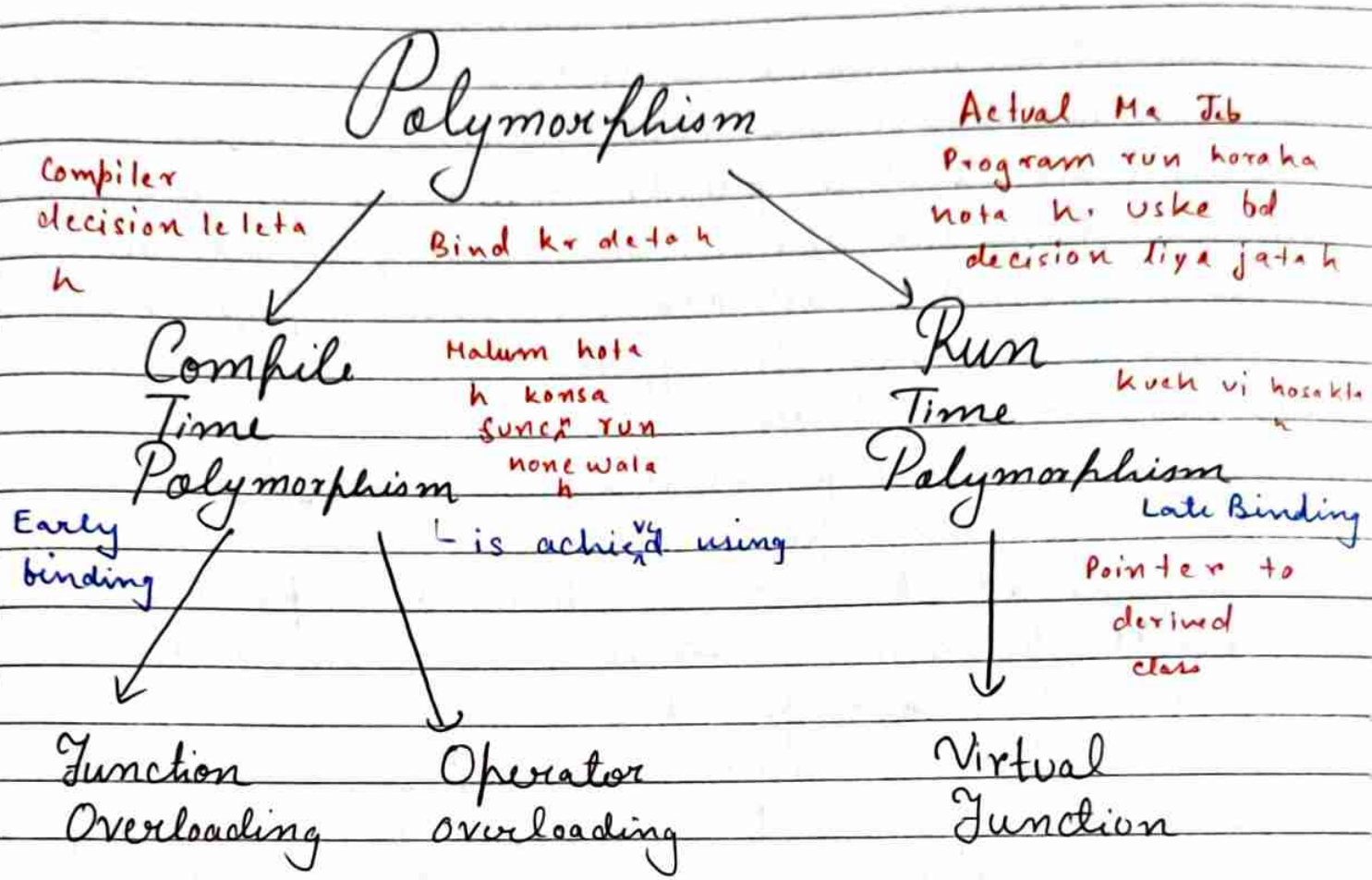
→ local variable is same as a member's name

```
#include<iostream>
using namespace std;
class A{
  int a;
public:
/*  void setData(int a){
      this -> a = a;
      return * this;
  }
*/
  void setData(int a){
    this -> a = a;
  }
  void getData(){
    cout << "The value of a is" << a << endl;
  }
};
```

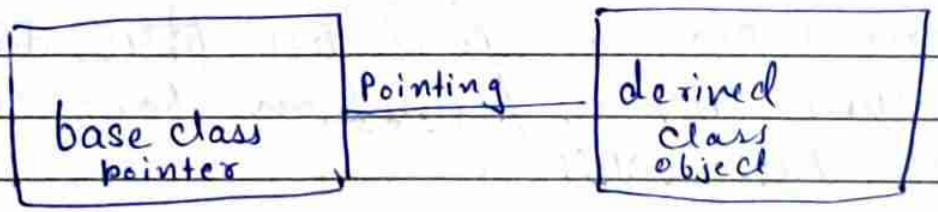
```
int main() {
// this is a keyword which is a pointer which
points to the object which invokes the
member function
A a;
a.setData(1){getData();}
a.getData();
return 0;
}
```

Polymorphism — many forms.

One name and multiple forms.



Pointer to Derived Class



Used to point the derived class object and pointer can still use aspects of base class.

Greeks For Greeks

Approach :-

- A derived class is a class which takes some properties from its base class.
- It is true that a pointer of one class can point to other class, but classes must be a base and derived class, then is possible.
- To access the variable of the base class, base class pointer will be used.
- As pointer is type of base class, and it can access all, public function and variable of base class, ~~base class pointer will be used and it can access all public~~ since pointer is of base, this is known as binding pointer.
- In this pointer base class is owned by base class but points to derived class object.
- Same works with derived class pointer, values is changed.

→ अगर आपका Pointer Base class का है और आपने उसे Derived class को Point करा दिया तब की आपका display function base class का ही Run होगा T

→ Base class का pointer banaya और derived class को Point करा दिया तो सिर्फ आप Base class के properties और Methods को Access कर पाओगे T

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

class Baseclass{
public:
  int var_base;
  void display(){
    cout<<"Displaying Base class variable var_base"
    <<var_base<<endl;
  }
};

class DerivedClass:public BaseClass{
public:
  int var_derived;
  void display(){
    cout<<"Displaying Base class variable var_derived"
    <<var_derived<<endl;
  }
};
  int main()
  Baseclass* base_class_pointer;
  BaseClass obj_base;
  DerivedClass obj_derived;
  base_class_pointer = &obj_derived; // Pointing base
                                     class pointer to
                                     derived class
```

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

class Baseclass{
public:
  int var_base;
  void display(){
    cout << "Displaying Base class variable var_base"
    << var_base<< endl;
  }
};

class DerivedClass: public Baseclass {
public:
  int var_derived;
  void display(){
    cout << "Displaying Base class variable var_derived"
    << var_derived << endl;
  }
};
  int main()
  Baseclass * base_class_pointer;
  BaseClass obj_base;
  DerivedClass obj_derived;
  base_class_pointer = &obj_derived; // Pointing base
                                     class pointer to
                                     derived class
```

base_class_pointer → var_base = 3400;
base_class_pointer → display();

base_class_pointer → var_base = 34;
// base_class_pointer → var_derived = 134; // will throw an error
base_class_pointer → display(); → base class ka function call hoga Run time pe

DerivedClass * derived_class_pointer;
derived_class_pointer = &obj_derived;
derived_class_pointer → var_base = 9448;
derived_class_pointer → display();

return 0;
}

Output

Displaying Base class variable var_base 34
Displaying Base class variable var_base 3400
Displaying Base class variable var_base 9448
Displaying Derived class variable var_derived 98

Naam toh bhol saare the but humne run-time pe decide kiya konsa function run hoga.

Virtual Functions in C

Base class को रहता है
अगर मेरे type का Pointer derived class के object को Point करे तो उसी का ही display function Run करना।
Ok मेरा display fn करना Run होने कि तुम पहले करते थे उसी का ही Run करना T

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

Class BaseClass {
public:
  int var_base = 1;
  virtual void display(){
    cout << " 1 Displaying Base class variable
    var_base " << var_base<<endl;
  }
};

Class DerivedClass : public BaseClass {
 public:
   int var_derived = 2;
   void display(){                    override
   cout << " 2 Displaying Base class variable var_base
    " << var_base <<endl;
   cout << " 2 Displaying Derived class variable
     var_derived " << var_derived <<endl;
   }
};
```

```
int main(){
BaseClass * base_class_pointer;
Baseclass obj_base;
DerivedClass obj_derived;
base_class_pointer = & obj_derived;
base_class_pointer → display();
return0;
}
```

Output

2 Displaying Base class variable var_base 1
2 Displaying Derived Class variable var_derived 2

→ Binding Runtime पे होती है ।

→ Program का output Program को दोबारा चला कर देखा जा सकता है ।

→ Actual में Compiler bind करता है । यानी Function के Address से object को Associate करता है तो करता है Runtime पर

Virtual Functions Example

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

class CWH{
  protected:
    string title;
    float rating;
  public:
    CWH(string s, float r){
      title = s;
      rating = r;
    }
    virtual void display(){}
};

class CWHVideo : public CWH
{
  float videoLength;
  public:
    CWHVideo(string s, float r, float vl)
    :CWH(s,r){
      videoLength = vl;
    }

    void display(){
    cout<<"This is an amazing video with title"
    <<title<<endl;
    cout<<"Rating: "<<rating<<"Out of 5stars"
    <<endl;
```

virtual likhte hum
If don pointer se
call hote hamay
ye apne apne
call honge
lekin agar virtual
nhi likhte hum
toh ye empty function
call hoke rah jayega
kuch ye nhi hoga!!

```
cout << "Length of this video is " << videoLength
<< "minutes" << endl;
}
};

class CWHText : public CWH
{
int words;
public:
CWHText(string s, float r, int wc) : CWH(s, r){
words = wc;
}
void display(){
cout << "This is an amazing text tutorial
with title " << title << endl;
cout << "Ratings of this text tutorial: "<<
"out of 5 stars" << endl;
cout << " No of words in this text tutorial
is: " << words << " words" << endl;
}
};

int main(){
string title;
float rating, vlen;
int words;
```

```
vlen = 4.56;
rating = 4.89;
CWHVideo dj Video(title, rating, vlen);
// dj Video.display();

// for Code with Harry Text
title = "Django tutorial Text";
words = 499;
rating = 4.19;
CWHText dj Text(title, rating, words);
// dj Text.display();

CWH* tuts[2];
tuts[0] = &dj Video;
tuts[1] = &dj Text;

tuts[0] -> display();
tuts[1] -> display();
return 0;
}
```

Output

This is an amazing video with title Django tutorial
Rating: 4.89 out of 5 stars
Length of this video is: 4.56 minutes
This is an amazing text tutorial: 4.19 out of 5 stars
No. of words in this text tutorial is 499

// Rules for virtual functions

1. They cannot be static
2. They are accessed by object pointers
3. Virtual functions can be a friend of another class
4. A virtual function in base class might not be used.
5. If a virtual function is defined in a base class, there is no necessity of redefining it in the derived class

→ अगर उसको नही मिला derived class मे display का version तो वो Base class के display को Run करलेगा ✌ अपने आप !

Abstract Base Class & Pure Virtual Func.

Tumne pure Virtual banadiya function को और तुम उसको derive कर नही कर रहे हो

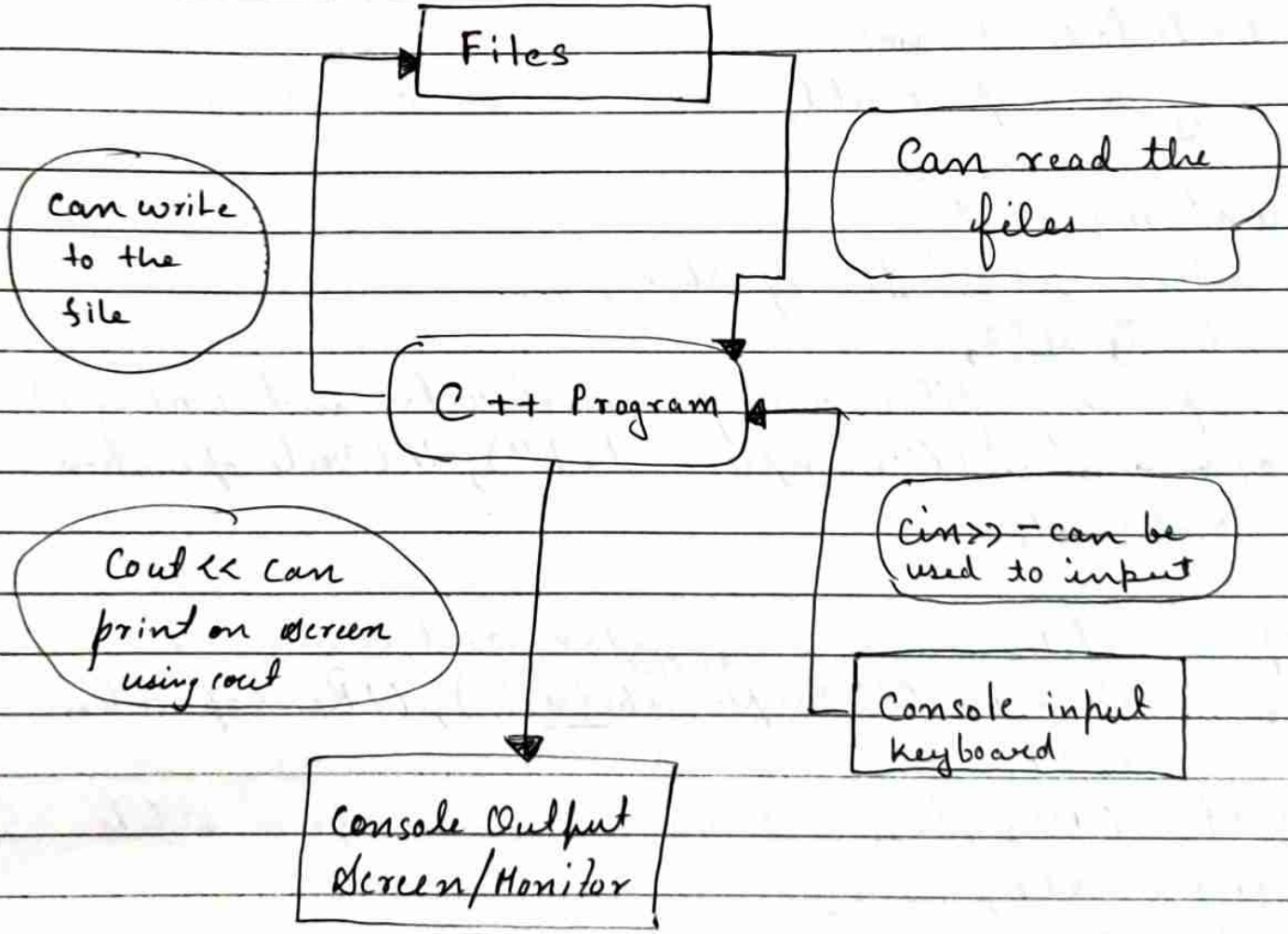
virtual void display() = 0; // do-nothing function → pure virtual function

pure virtual function का use Abstract Base class बनाने में होता है।

virtual void display() = 0;

⇒ khud run नहीं होगी। बोल रहा है आगे जाके Override कर दो मुझे impose कर देता है display का नया version write करना।

Abstract Base class - एक essi class जिसको आप Objects create ~~kis~~ बनाने के लिए use नहीं कर सकते इसका मतलब है कि design की गई है इसलिए की आप इससे और Class inherit करें



File I_O in C

The useful classes for working with files in C++ are:-

1. fstream
2. ifstream --> derived from fstreambase
3. ofstream --> derived from fstrambase

In order to work with files in C++, you will have to open it. Primarily, there are 2 ways to open a file:

1. Using the constructor
2. Using the member function open() of the class

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

int main(){
string st = "Harry bhai";
string st2;
// opening files using constructor and writing it
// ofstream out("sample60.txt"); // Write operation
// out<<st;

// Opening files using constructor and reading it
ifstream in("sample60.txt"); // Read operation
// in >>st2;
getline(in, st2); → This is coming from a file
cout<<st2;
return 0;
}
```

→ This

C++: File I/O [Reading & Writing to a File]

3 useful classes

1> fstreambase
2> ifstream → Derived from (1)
3> ofstream → Derived from (2)

} (fstream)

Read Operation

```
ifstream in ("this.txt");
string st;
in >> st; // Just like Cin
```

Write Operation

```
ofstream out ("this.txt");
string st = "Harry";
out << st; // Writes to a file this.txt!
```

```
#include <iostream>
#include <fstream>
using namespace std;
int main() {
    // connecting our file with out stream
    ofstream fout ("sample60.txt");

// Creating a name string and filling it with
the string entered by the user.
string name;
cout << "Enter your name";
cin >> name;
```

```
// Writing a string to the file
hout << name + " is my name ";
hout.close();

if stream hin ("Sample60.txt");
string content;
hin >> content;
cout << "The content of this file is: " << content;
hin.close();

return 0;
```

File I_O in C++ open() & eof()

endoffile

```
#include<iostream>
#include<fstream>
#include<string>
int main() {
    ofstream out;
    out.open("Sample60.txt");
    out" << "This is me \n";
    out << "This is me also";
    out << "This is also me";
    out << "This is also me";
    out.close();
```

method2

```
ifstream in;
string st, st2;
in.open("sample60.txt");
/*
in >> st >> st2;
cout << st << st2;
in.close();
return 0;
*/}
```

→ This is

method2

```
while(in.eof()==0) {
getline(in, st);
cout << st << endl;
}
in.close();
return 0;
}
```

→ saari lines print hojayegi

Templates

int → variable

Class → Object

Template → class

↓

Parameterized Classes.

Syntax For Templates

```
template <class T>
class Vector {
    T * arr;
    Public:
    Vector (T *ar)
     {
      // code
     }
// & many other methods
}
```

T can be int, float, char, etc...

```
int main(){
→ vector <int> myVec(ptr);
→ vector <float> myFVec(ptr);
```

represent
→ Many classes

CP
STL

Writing Our First C++ Template

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

template<class T>
class vector{
 public:
  T* arr;
  int size;
  vector(int m){
    size = m;
    arr = new T[size]
  }

  T dotProduct(vector &v){
  T d = 0;
  for(int i=0;i<size;i++)
  {
    d+= this->arr[i]*v.arr[i];
  }
  return d;
}
  int main(){
   // vector v1(3);
   // v1.arr[0]=4;
   // v1.arr[1]=3;
   // v1.arr[2]=1;
   // vector v2(3);
   // v2.arr[0]=1;
   // v2.arr[1]=0;
   // v2.arr[2]=1;
```

Custom data type T

```
// int a = v1.dotProduct(v2);
// cout<<a<<endl;
                    float
vector<int> v1(3);
v1.arr[0]=1.4;
v1.arr[1]=3.3;
v1.arr[2]=0.1;
vector<float> v2(3);
v2.arr[0]=0.1;
v2.arr[1]=1.90;
v2.arr[2]=4.1;
float ~~int~~ a = v1.dotProduct(v2);
cout<<a<<endl;
return 0;
```

Output

6.82

Templates With Multiple Parameters

```
#include<iostream>
using namespace std;
/*
template<class T1, class T2> // COMMA SEPERATED
Class nameofClass{
 // body
};*/
```

```
template<class T1, class T2>
class myClass{
  public:
    T1 data1;
    T2 data2;
    myClass(T1 a, T2 b){
      data1 = a;
      data2 = b;
    }
    void display(){
    cout<< this -> data1 <<endl<< this -> data2;
    }
    int main(){
    myClass<char, float> obj('c', 1.8);
    obj.display();
    return 0;
    }
```

Class Template with default Parameters.

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

template<class T1 = int, class T2 = float>
class Harry{
```

```
public:
  T1 a;
  T2 b;
  Harry (T1 x, T2 y) {
    a = x;
    b = y;
  }
  void display () {
    cout << " The value of a is " << a << endl;
    cout << " The value of b is " << b << endl;
  }
};
  int main () {
  Harry <> h (4, 6.4);
  h.display ()
   return 0;
```

Default values

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

template <class T1 = int, class T2 = float, class T3 =
          char>
Class Harry {
  public:
    T1 a;
    T2 b;
    T3 c;
```

```
Harry (T1 x, T2 y, T3 z) {
a=( x;
b = y;
c = z;
}
   void display() {
      cout << "The value of a is" << a << endl;
      cout << "The value of b is" << b << endl;
      cout << "The value of c is" << c << endl;
   }
};

int main() {
   Harry <> h(4, 6.4, 'c');
   h.display();
   cout << endl;
   Harry <float, char, char> g(1.6, 'o', 'c');
   g.display();
   return 0;
}
```

Output

```
The value of a is 4
The value of b is 6.4
The value of c is C

The value of a is 1.6
The value of b is o
The value of c is c
```

Function Templates & Function Templates with Parameters

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

// float funcAverage (int a, int b) {
// float avg = (a+b)/2.0;
// return avg;
// }

// float funAverage2 (int a, float b) {
//   float avg = (a+b)/2.0;
//  return avg;
// }

  template <class T>
  void swap (T& a, T& b) {
    T temp = a;
    a = b;
    b = temp;
  }
    template <class T1, class T2>
    float funcAvgeriage (T1 a, T2 b) {
      float avg = (a+b)/2.0;
      int x = 5, y = 7;
      swap (x, y)
      cout << x << endl << y;
      return avg;
    }
```

```
int main(){
float a;
a = funcAverage (5,2);
printf ("The average of these numbers is %.3f\n"
,a);
int x=5, y=7;
swap( x, y);
cout<<x<<endl<<y;
return 0;
}
```

Output
The average of these numbers is 3.500
7
5

Member Function Templates & Overloading Template Functions

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

template<class T>
class Harry{
  public:
    T data;
    Harry(T a){
      data = a;
  }
    void display();
};
```

```
template<class T>
Void Harry<T>::display(){
    cout<<data;
}
void func(int a){
    Cout<<" I am first func() "<<a<<endl;
}
  template<class T>
  void func(T a){
      cout<<" I am templatized func() "<<a<<endl;
}
int main(){
// Harry<float> h(5.7);
// Harry<char> h('c');
// Harry<int> h(87);
// Cout<<h.data<<endl;
// h.display();

func(4); // Exact match takes the highest priority
func<>(4); // Exact match takes the highest priority

return 0;
}
```

Geeks for Geeks

Overloading function templates in C++

Template:

- A template is a tool that reduces the efforts in writing the same code as templates can be used at those places.
- A template function can be overloaded either by a non-template function or using an ordinary function template.

Function Overloading: In function overloading, the function may have the same definition, but with different arguments. Below is the C++ program to illustrate ~~the~~ function overloading:

```
#include <bits/stdc++.h>
using namespace std;

// Function to calculate square
void square(int a)
{
    cout << "Square of " << a
         << " is " << a * a << endl;
}

// Function to calculate square
void square(double a)
{
    cout << "Square of " << a << " is " << a*a << endl;
}
```

output
Square of 9 is 81
Square of 2.25 is 5.0625

```
// Driver Code
int main()
{
    // Function Call for side as 9 i.e., integer
    Square(9);

    // Function Call for side as 2.25 i.e., double
    Square(2.25);
    return 0;
}
```

Explanation:

- In the above code, the Square is overloaded with different parameters.
- The function Square can be overloaded with other arguments too, which requires the same name and different arguments every time.
- To reduce these efforts, C++ has introduced a generic type called function template

✳ Function Template: The function template has the same syntax as a regular function, but it starts with a keyword template followed by template parameters enclosed inside angular brackets <>.

```
template <class T>
T function_Name (T arguments)
{
  // Function definition
  ..... ..... ..... .....
}
```

where, T is template argument accepting different arguments and class is a keyword.

Template Function Overloading :-

* The name of the function templates are the same but called with different arguments is known as function template overloading.

* If the function template is with the ordinary template, the name of the function remains the same but the number of parameters differs.

* When a function template is overloaded with a non-template function, the function name remains the same but the function's arguments are unlike.

```
#include<bits/stdc++.h>
using namespace std;

// Template declaration
template<class T>

// Templat overloading of function
void display(T t1)
{
  cout<<"Displaying Templates: "<<t1<<"\n";
}

// Template overloading of function
void display(int t1)
{
  cout<<"Explicitly display: "<<t1<<"\n";
}

// Driver Code
int main()
{
  // Function Calls with diff. arguments
  display(200);
  display(12.40);
  display('G');

  return 0;
}
```

Output

Explicitly display: 200
Displaying Template: 12.4
Displaying Template: G

Introduction to STL

STL → Standard Template Library

△ Library of What?
↓
→ Generic Classes and functions.

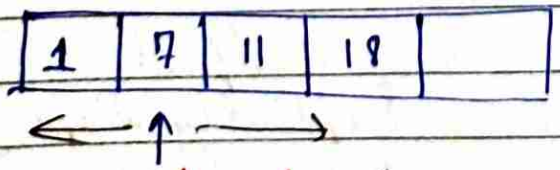
△ Why use STL → Reuse: Well tested Components
→ Time Savings!

Components of STL → (1) Containers → stores data
→ use template classes
(2) Algorithms → Sorting
→ Searching
→ use template functions.

STL is used because its a good idea not to Reinvent the wheel.

(3) Iterators
↳ object points to an element in a Container
↳ Handled just like pointer
↳ Connects Algo with Containers

Container:



iterator (moves as instructed by the Algo)

Containers in STL

STL = Containers + Algo + Iterators

- Containers: Object which stores data
- Algo: procedure to process data
- Iterators: Object which points to an element of a container

Containers

1. → Sequence Containers → Linear fashion → Vector → List → Deque
2. → Associative Containers → Direct access → Set/Multiset → Map/Multimap → Tree
3. → Derived Containers → Real World Modelling → Stack → Queue → Priority-queue

When to use which?

→ Sequence Containers

1. vectors → RA ⇒ Fast
   Insertion in Middle / Deletion ⇒ Slow
   Insertion at End ⇒ Fast

2. List → RA ⇒ Slow
   Insertion in Middle ⇒ Fast
   Del / Insertion at the end ⇒ Fast

→ Associative Containers ⇒ All operations are fast except RA.

→ Derived Containers ⇒ Depends → Data structure

Vectors in C

→ Arrays cannot be Resized.

```
#include<iostream>
#include<vector>

using namespace std;
```

First)

```
int main(){
    vector<int> vec1; // Vector को बनाने का Syntax
    return 0;
}
```

Second)

```
void display(vector<int>& v){        → Template
    for(int i=0; i<v.size(); i++)
    {
        cout<<v[i]<<" ";
    }
    cout<<endl;
}

int main(){
    vector<int> vec1;
    int element, size;
    cout<<"Enter the size of your vector"<<endl;
    cin>>size;
    for(int i=0; i<size; i++)
    {
        cout<<"Enter an element to add
        to this vector: ";
```

```
cin>>element;
vec1.push_back(element); // Last wala element
}                           निकाल लो T
display(vec1);
```

Qulpul

```
Enter the size of your vector
5
Enter an element to add to this vector: 1
Enter an element to add to this vector: 2
Enter an element to add to this vector: 3
Enter an element to add to this vector: 4
Enter an element to add to this vector: 5
1 2 3 4
```

third)

```
//vec1.pop_back();
display(vec1);
vector<int>::iterator iter = vec1.begin();
vec1.insert(iter+1, 500, 566);
display(vec1);
```

iter+1 → at the position you wan
500 → no. of times you want the copy
566 → element to be inserted

1 (566) x 500 2 3 4

fourth) #include<iostream>
#include<vector>

using namespace std;

Template<class T>
void display(vector<T>& V){
cout<<"Displaying this vector";
for(int i=0; i<v.size(); i++)
{
cout<<v[i]<<" ";
// cout<<v.at(i)<<" ";
}
cout<<endl;
}

int main(){
// Ways to create a vector
vector<int> vec1; // zero length integer vector
vector<char> vec2(4); // 4-element character vector

// vec2.push_back('5');
// display(vec2);
// vector<char> vec3(vec2); // 4-element character vector from vec2
// display(vec3);
vector<int> V(6,3); // 6-element vector of 3s
display(vec4);
cout<<vec4.size(); // returns size

List in C++

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

void display (list <int> & list) {
    list<int> :: iterator it;
  for (it = lst.begin(); it != lst.end(); i++)
    {
      cout << *it << " ";
    }
    cout << endl;
}
int main(){
  // 6 8 9
  list <int> list1; // List of 0 length;
  list 1.push_back(5);
  list1.push_back (7);
  list1.push_back (1);
  list 1.push_back (9);
  list 1.push_back (12);
  display (list 1);

  list <int> list2 (3); // Empty list of size 3
  list <int> :: iterator iter;
  iter = list 2.begin();
  *iter = 45;
  iter++;
  *iter = 6;
  *iter++;
```

list1.pop_front(); delete

Middle delete

list1.remove(9); deletes all occurrences

output: 5 7 1 9 12 → 5 7 1 9 → 4 5 6 9

list1.pop_back(); last element popped

display(list1);

```
*iter = 9;
iter++;
display (list2);
```

Output
5 7 1 9 12
45 6 9

```
// Sorting the list
// list1.sort();
// display (list1);
```

Before
5 7 1 9 12
1 5 7 9 12

```
/* list1.merge (list2);
cout << "List 1 after merging: ";
display (list1);
*/
```

output
5 7 1 9 12 4 5 6 9

```
/* list1.sort();
list2.sort();
list1.merge (list2);
cout << "List 1 after merging: ";
display (list1);
*/
```

output
List 1 after merging:
↳ 1 5 6 7 9 9 12 15

```
// Reversing the list
list1.reverse();
```

Output
5 7 1 9 12
12 9 1 7 5

Void Pointer in C/C++

A void pointer is a pointer that has no associated data type with it. A void pointer can hold addresses of any type and can be typecasted to any type.

// C++ Program to demonstrate that a void pointer can hold the address of any type-castable type

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

int main()
{
  int a = 10;
  char b = 'x';
  void *p = &a; // void pointer holds address of int a
  p = &b; // void pointer holds address of char b
}
```

Advantages of void pointers 1) malloc() and calloc() return void* type and this allows these functions to be used to allocate memory of any data type (just because of void*)

Note: that the above program compiles in C, but doesn't compile in C++. In C++, we must explicitly typecast return value of malloc() to (int*).

2) void pointers in C are used to implement generic functions in C.

3) a void pointer cannot be dereferenced

4) The C standard doesn't allow pointer arithmetic with void pointers. However in GNU c it is allowed by considering the size of void 1.

Deep Dive

C++

In C++ we cannot assign the address of a variable do the variable of a different datatype

```
int *ptr; // integer pointer declaration
float a = 10.2 // floating variable initialization
ptr = &a // This statement throws an error in C++ but not in C
```

```
void *ptr; // void pointer declaration
int a = 9; // integer variable initialization
ptr = &a; // storing the address of 'a' variable in a void pointer variable.
std::cout<<&a<<std::endl;
std::cout<<ptr<<std::endl;
return 0;
}
```

output

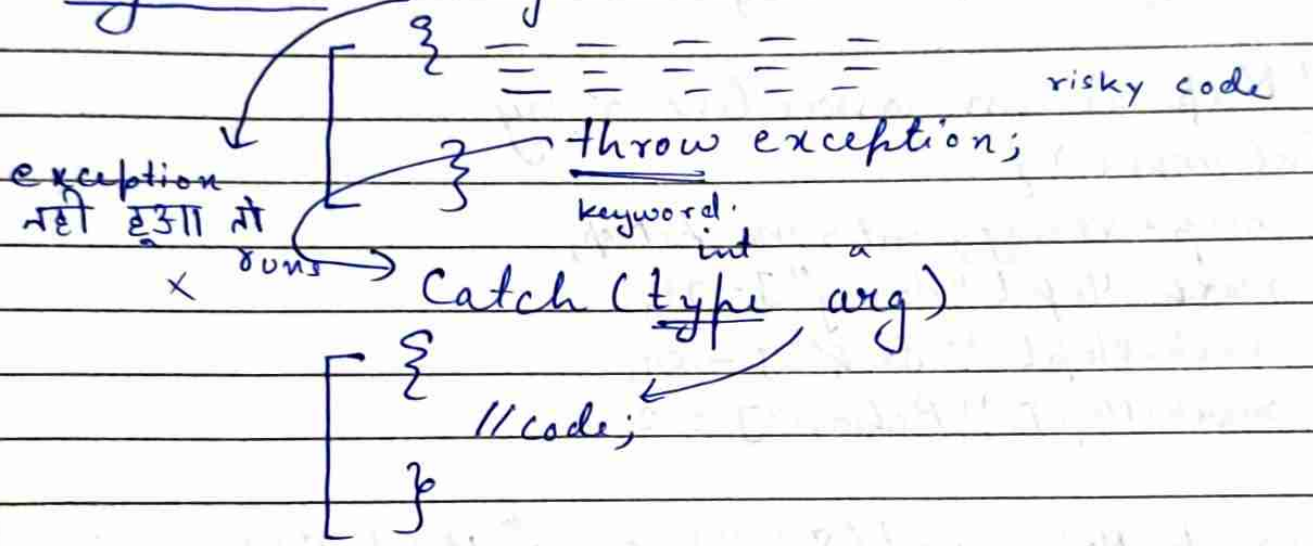
```
0x7ffcf1da5e04
0x7ffcf1da5e04
```

Exception Handling

What is Exception handling? And its syntax...

Exception Handling is a process to handle runtime errors. We perform exception handling so, the normal flow of program can be maintained even after runtime errors.

Syntax : try



at a time either try block will execute or catch block will execute

(a) try : It represents a block of code that can throw an exception.

(b) catch : It represents a block of code that is executed when a particular exception is thrown.

(c) throw : It is used to throw an exception

Maps in C++ STL

key → value
key → value
key → value

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

// Map is an associative array
int main(){
  map<string, int> marksMap;
  marksMap["Harry"] = 98;
  marksMap["Jack"] = 59;
  marksMap["Rohan"] = 2;

marksMap.insert({{"kozume", 169}, {"kuroo", 187}});
map<string, int>:: iterator iter;
for(iter=marksMap.begin(); iter != marksMap.
    end(); iter++){
    cout<<(*iter).first<<" "<<(*iter).second<<
    "\n";
}
return 0;
}
```

Output

```
Harry 98
Jack 59
kozume 169
kuroo 187
Rohan 2
```

Function Objects (Functors) in C

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

int main(){
// Function Objects: Function wrapped in a Class
so that it is available like an object
int arr[] = { 1 , 73, 4, 2, 54, 16};
// sort(arr, arr+5);
sort(arr, arr+5,
    greater<int>());
for(int i=0; i<6; i++)
{
  cout << arr[i] << endl;
}
return 0;
}
```

output:
1
2
4
54
73
16

sort

because arr+5

output
73
54
16
4
2
1

descending order

Templates in C++

Templates are powerful features of C++ which allows you to write generic programs. In simple terms, you can create a single function or a class to work with different data types using templates.

The concept of template can be used in two different ways:

- Function Templates
- Class Templates

// Function Template

```
#include<iostream>
using namespace std;
void sum(int a, int b){
  cout<<"Total is : "<<a+b;
}
int main(){
  sum(2.3, 3.5);
}
```

Output :
Total is : 5.8 What we want

→ Here we get
Total is 5 which is not exact.

You must mention this

```
#include<iostream>
using namespace std;
template<class T>
// Syntax of function Template
T sum(T a, T b){
  cout<<"Total is: "<<a+b;
}
int main(){
  sum(2.3, 3.5);
}
```

// Class Template

```
#include <iostream>
using namespace std;
template <class X>
class Demo
{
 private:
  X num1, num2;

 public:
  Demo(X n1, X n2)
  {
    num1 = n1;
    num2 = n2;
  }

  void check()
  {
   if (num1 > num2) {
    cout << num1 << " is the largest number" << endl;
   }
   else {
    cout << num2 << " is the largest number" << endl;
   }
  }
};
int main()
{
  Demo obj1(5.2, 5.6);
  obj1.check();
  return 0;
}
```

keyword.

keyword.

we can take anything not compulsory to take X or T.

Demo <float> obj1(5.2, 5.6);

* obj बनाने से पहले किस type का values pass करा रहे हैं T <Mention> करो

error.

output

5.6 is the largest number

Recursion : a func^n calls itself

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

long fact(int n)
{
  if(n==0)   // base class
  return 1;
  else
  return (n*fact(n-1));  // recursive func$^n$ call
}

int main()
{
  int num;
  cout << "Enter a positive no.:";
  cin >> num;

  cout << "Factorial of " << num << " is " << fact(num);

  return 0;
}
```

2

(2 * 1) => (2 * 1 * 1) = 2

Enter a positive no.: 2
Factorial of 2 is 2

Solving Tower of Hanoi. Problem

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

void TOH (int d, char tower1, char tower2,
  char tower3)

{
  if (d==1)   // base class
  {
  cout << "\n Shift top disk from" << tower1
  << " to tower :" << tower2;
  return;
  }
  else
  {
   TOH (d-1, tower1, tower3, tower2);
  cout << "\n Shift top disk from tower "<<
   tower1 << " to tower" << tower2;

  TOH (d-1, tower3, tower2, tower1);
}
int main()
{ int disk;
  cout << "Enter the no. of disk: ";
  cin >> disk;
  if (disk < 1)
  cout << "\n There are no disks to shift";
  else
  cout << "\n There are " << disk << "disk in
                                tower 1\n";
```

```
TOH (disk, '1', '2', '3');
cout << "\n\n" << disk << "disks in tower 1
are shifted to tower 2";

return 0;
}
```

Static Data Members

A data member of a class can be qualified as Static. The properties of a static member variable are similar to that of a C static variable.

Sps. features:-

* It is initialized to zero when the first object of its class is created. No other initialization is permitted.

* Only one copy of that member is created for the entire class and is shared by all the objects of that class, no matter how many objects are created.

* It is visible only within the class, but its lifetime is the entire program.

Note Static variables are normally used to maintain values common to the entire class

Note that the type and scope of each static member variable must be defined outside the class definition. This is imp. because the static data members are stored seperately rather than as a part of an object. ∴ they are associated with the class itself rather than with any class object they are also known as class variables.
Some initial value can also be assigned to var.

int item : : count = 10;

```
#include <iostream>
using namespace std;
class item
{
  static int count;
  int number;
public:
 void getdata(int a)
  { number = a;
    count ++;
  }
  void getcount(void)
  {
    count<<"count:";
    count << count << "\n";
  }
};
int item::count;
int main()
 {
   item a, b, c;          // count is initialized
   a.getcount();          to zero
   b.getCount();          // display count
   c.getcount();

   a.getdata(100);  // getting data into object a
   b.getdata(200);// getting data into object b
   c.getdata(300);// getting data into objects

count << "After reading data" << "\n";
```

```
a.getcount();      //displaycount
b.getcount();
c.getcount();
return 0;
}
```

Output

```
count:0
count:0
count:0
After reading data
count:3
count:3
count:3
```

Static Member Functions :-

Properties :-

- A static function can have access to only other static members (functions or variables) declared in the same class.
- A static member function can be called using the class name (instead of its obj) :

```
class-name :: function name;
```

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

Class test.
{
    int code;
    Static int count; // static mem. var.
public:
    void setcode (void)
    { code = ++count;
    }
    void showcode ( void)
    {
     cout << "Object number: " << code << "\n";
    }
    static void showcount (void) // static mem fun
    {
    cout << " count : " << count << " \n";
    }
};
int test :: count;
int main()
{
  test t1 , t2;
   t1 . setcode ();
   2 2 . Set code ();
 test :: Showcount (); // accessing static function

  test t3;
  t3 . setcode ();

  test :: showcount;
```

```
    t1.showcode();
    t2.showcode();
    t3.showcode();
    return 0;
}
```

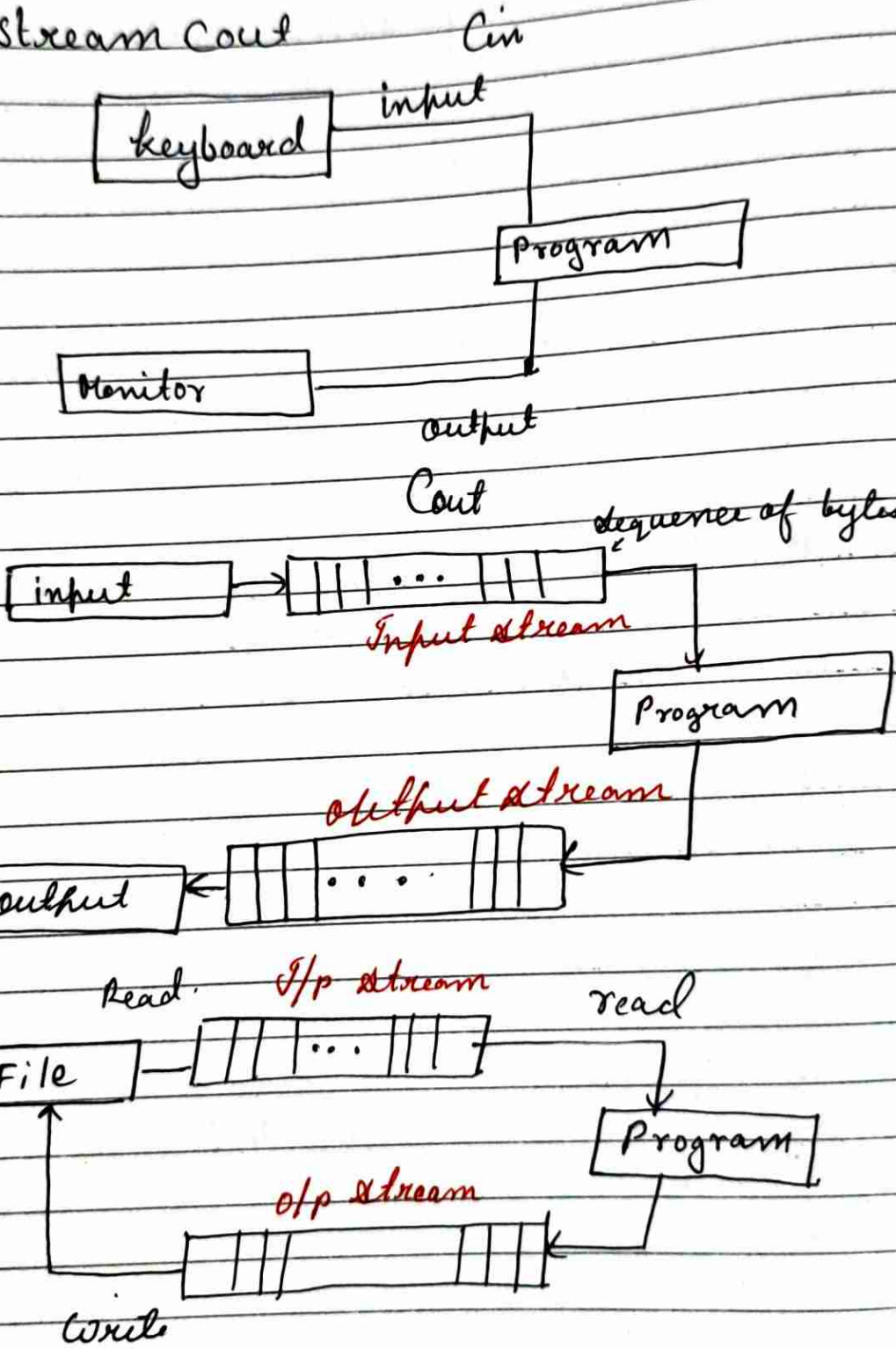
Output

```
count: 2
count: 3
Object number: 1
Object number: 2
Object number: 3
```

File Handling and Streams in C++

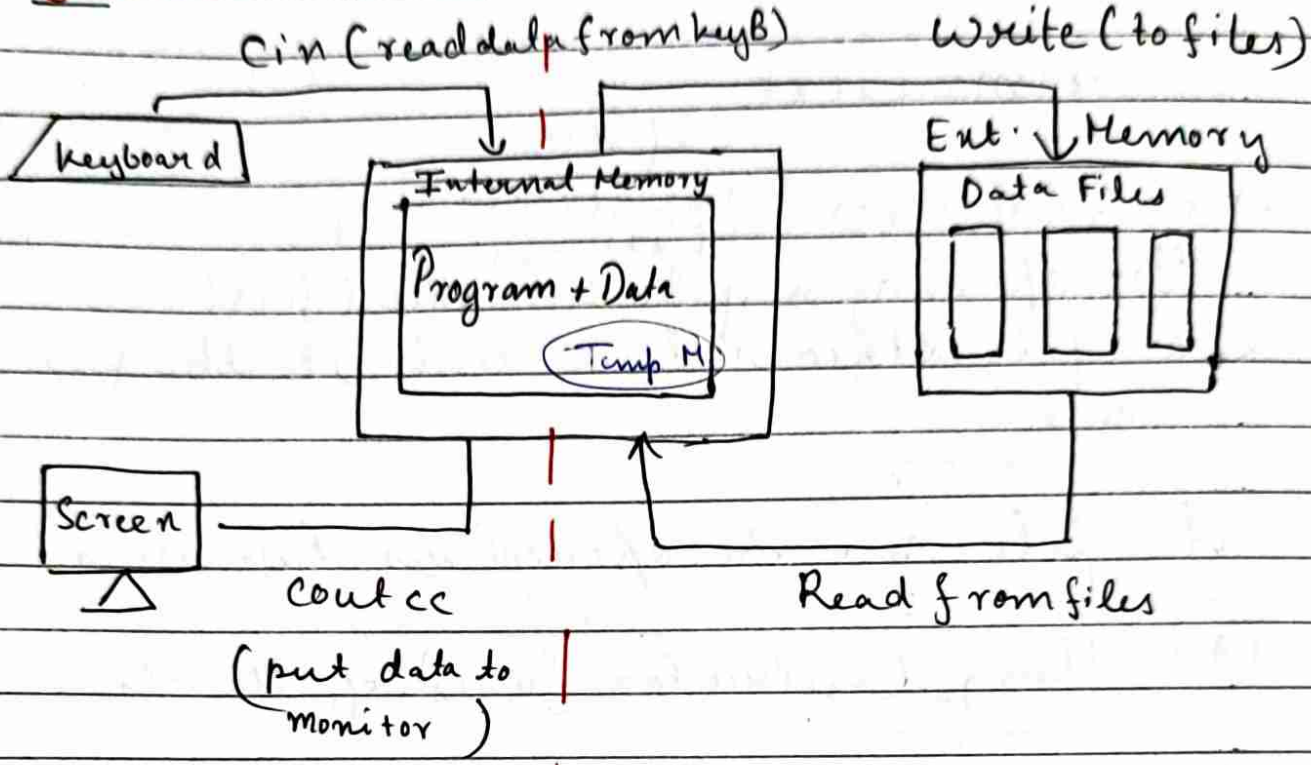
Stream

<iostream.h>
istream cin
Ostream cout



<fstream.h>

ifstream → Read
ofstream → write
fstream → Bath



← Console program interaction → | ← File – Program Interaction →

Opening & Closing Files in C++

(1) Suitable name for the file
(2) Data type and structure
(3) purpose
(4) Opening Method

(1) Suitable name for the file

raman.doc
↑ ↑
first second

raman.txt

(2) <fstream> → ifstream
→ ofstream
→ fstream

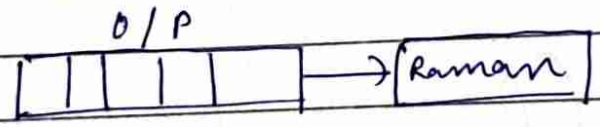
For opening a file we must first create a file stream then link it the file name.

A file can be opened in two ways

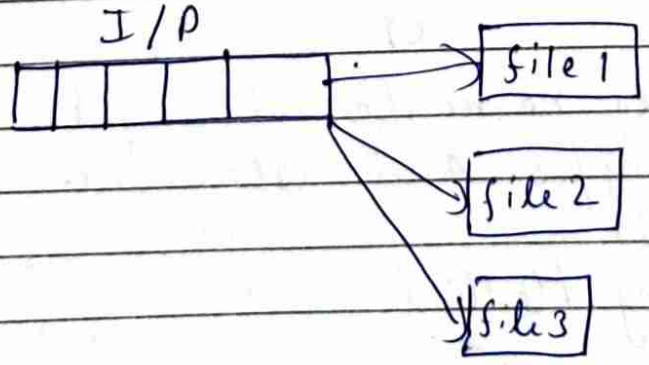
(A) Using Constructor Func^n of the class

(B) Using the Member Func^n opens of the class

(A)



(B)

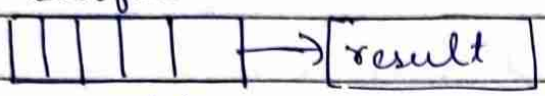


(A)

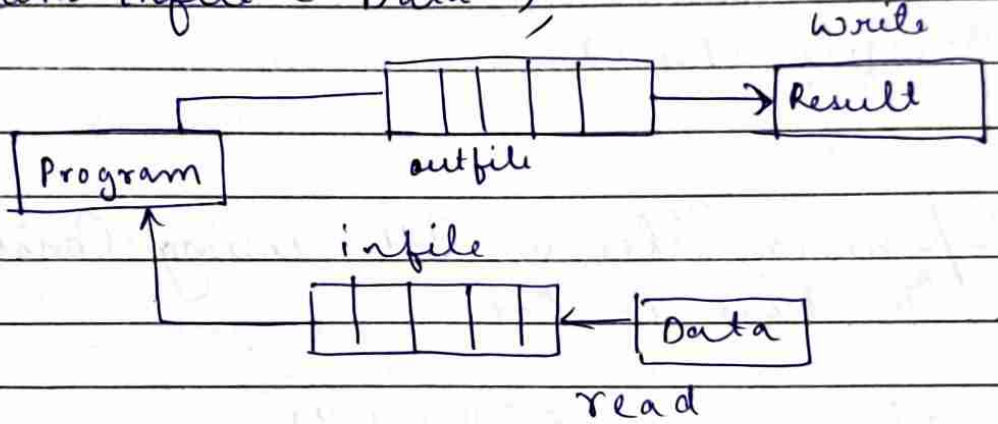
① ifstream , ofstream

② filename

ofstream outfile ("result");



ifstream infile ("Data");



ofstream outfile ("result");

outfile.close();

(B) यहाँ जो हमने stream create किया है वो आगे किस file se attached रहेगी है।

File_Stream_Class Stream_Object

Stream_Object.Open ("raman")

जिस stream filename को use करते हुए किस file को Manage कर सकते हैं।

ofstream

outfile.open("Data1");

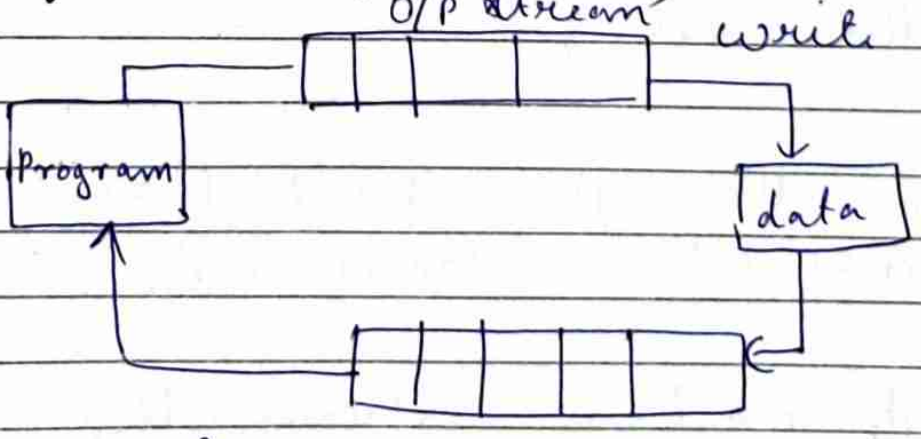
outfile.close();
outfile.open("Data2");

outfile.close();

Opening Files in C++ using Constructor Functions in C++

ofstream outf("data");

ifstream inf("data");



```
#include<iostream.h>
#include<fstream.h>
int main()
{
    char name[20];
    float cost;
    ofstream outf("data");
    Cout << "Enter item name";
    Cin>>name
    Cout << "Enter item cost";
    Cin >>cost;
    outf << name << "\n";
    outf << cost << "\n";
    outf.close();
    ifstream inf("data");
    inf >> name;
    inf >> cost;
    Cout << "\n Item Name" << name;
    Cout << "\n Item Cost" << cost
    inf.close();
    return 0;
}
```

obj single file work

obj उसी समय file name pass होता है

disconnect

obj

file से data read

class

dusre variable name vi use kr sakte h.

Open Files using open function in C++

```
ofstream outf;
outf.open("data")
outf << "Shiwi";
outf.close();
```

```
#include <iostream>
#include <fstream.h>
#include <conio.h>
int main()
{
    char name[20];
    int cost;
    ofstream fout;
    fout.open("data");
    fout << "keyboard \n";
    fout << 350 << "\n";
    fout.close();
    ifstream fin;
    fin.open("data");
    fin >> name;
    fin >> cost;
    cout << "\n Item name" << name;
    cout << "\n Item cost " << cost;
    fin.close();
    getch();
    return 0;
}
```

file modes.

open ("file name", file Mode);

fstream

default modes.

ifstream ios::in

ofstream ios::out

Parameters

ios::in ios::out
ios::app ios::ate

ios::nocreate ios::noreplace
ios::Binary "raman.txt" ios::trunc

ios::ate | ios::noreplace);

(A) get(char*)

(B) get(void)

```
char ch;
cin>>ch;
```

(A)

```
char ch;
cin.get(ch);
ch = cin.get();
```

Put

```
cout.Put('E');
char ch='x';
cout.put(ch);
```

```
#include<iostream.h>
#include<conio.h>
int main()
{
  int count=0;
  char ch;
  cout<<"Input Text";
  while(ch!='\n')
  { cin.get(ch);
    cout.put(ch);
  count++;
```

```
cout << "Number of characters" << (count-1);
getch();
return 0;
}
```

I/P : easytuts4you.com

O/P : easytuts4you-com

getline

```
cin.getline(line, size);
```

line – variable

size –

```
char name[10];
cin.getline(name, 5);
```

vikas pandey

name → vika

name → vikas pand

0 – (n-1)

last → null

Write

```
cout.write(line, size);
```

line – variable

size – no. of characters

```
cout.write(line, size);
cout.write(name, 5);
```

vikas pandey

size > length (12)

```
#include<iostream>
#include<conio.h>
int main()
{
    char name[20];
    cout<<"Enter your name";
    cin.getline(name,15);
    cout.write(name,10);
    cout<<"\n Name:"<<name;
    getch();
    return 0;
}
```

Complete Searching & Sorting Algorithm

Linear Search / Sequential Search

→ Searching technique is done in sequential manner

Algorithm:

Array name ↑ → element to be found

Seq_Search (A , i , X , n)

pointer ↑ → max^m size

5	9	12	20	25
0	1	2	3	4

X = 12

1. Set i = 0
2. if i > n
   Print : Element Not Found
3. if A[i] = x
   Print Element found
4. i = i + 1

← loop

i=0 n=5
0>5
5≠12
i=1

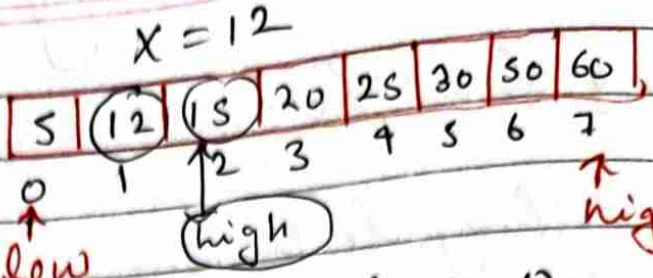
5. Exit

O(n)

Binary Search :

→ It is fast searching algo. with time complexity O(log n)
→ Based on divide & Conquer
→ Data items should be in sorted manner

Algorithm



① while (low ≤ high)

② mid = (low + high) / 2

low = 0

high = 60

mid = 3

③ if X = A[mid] return mid

④ if X > A[mid]
low = mid + 1

else high = mid - 1

12 ≠ A[3] ≠ 20

12 > 20 होगा तो low Badhao

नहीं तो high = 2
(high = mid - 1)

⑤ Exit

Sorting

Bubble Sort

In Bubble Sorting technique, each pair of adjacent compared and elements are swapped if they are not.

→ Worst Case time Complexity = $O(n^2)$

Algorithm

(1) for (jb+k sord, na ho jay)

(2) if (list[i] > list[i+1])
swap (list[i], list[i+1])
end if
end for

(4) Exit

Selection Sort:

→ In this sorting technique the list is divided into 2 parts

Sorted | Unsorted

→ Worst Case time complexity $O(n^2)$

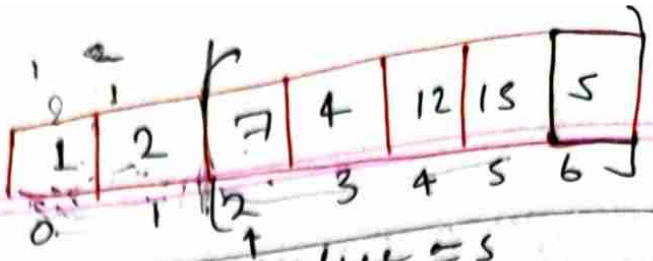
Algorithm

(1) Set min = 0

(2) Search minimum element in the list

(3) Swap with value at locan min

(4) Increase min to point to next element.



(5) Repeat until list is sorted

Insertion Sort:

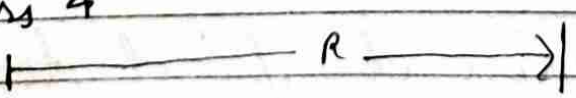
→ A sublist (or Sorted Array) is maintained which is always sorted.
→ Not suitable for large set of list
→ Avg and worst case time complexity is $O(n^2)$
→ $(n-1)$ pass are required to sort n elements
→ In each pass we insert current element at appropriate place so that elements in current range are in order

eg

Pass 1
↳ 5 6

Pass 2

Pass 3
4 5 6
Pass 4



2 4 5 6

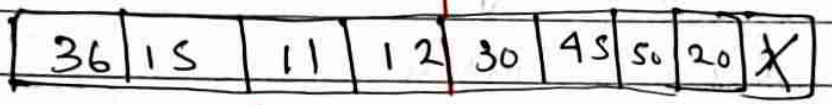
Merge Sort

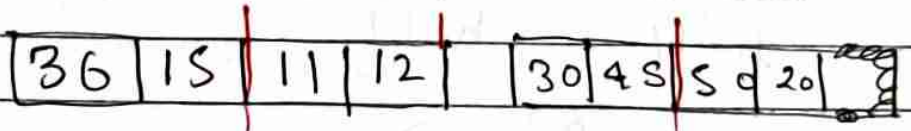
Based on divide and conquer technique
Worst case time complexity $O(n \log n)$

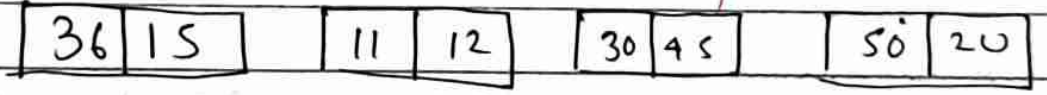
It divides the array into equal halves
And then merge them in sorted manner

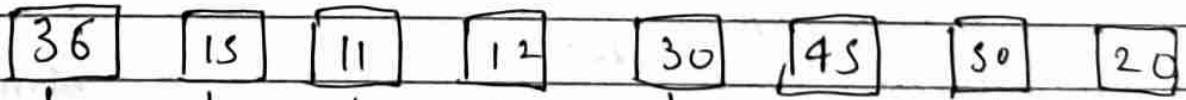
Algo

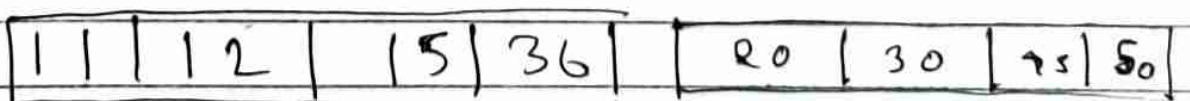
Beg : Start of array
end : End of array
if (beg < end)
then mid = (beg + end)/2

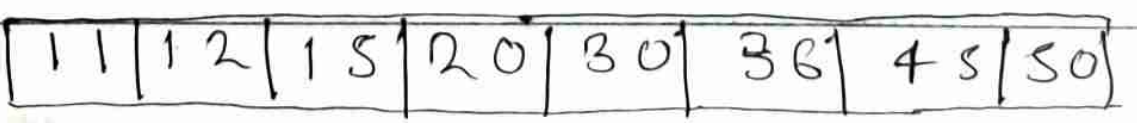












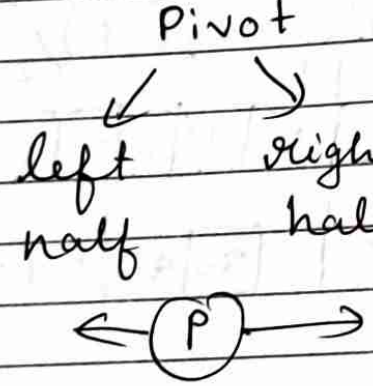
Quick Sort

→ Based on partitioning of array of data in smaller array

→ Uses the concept of divide & conquer

→ Efficient for large sized data list

→ Average & worst case time complexity is O(nlogn)

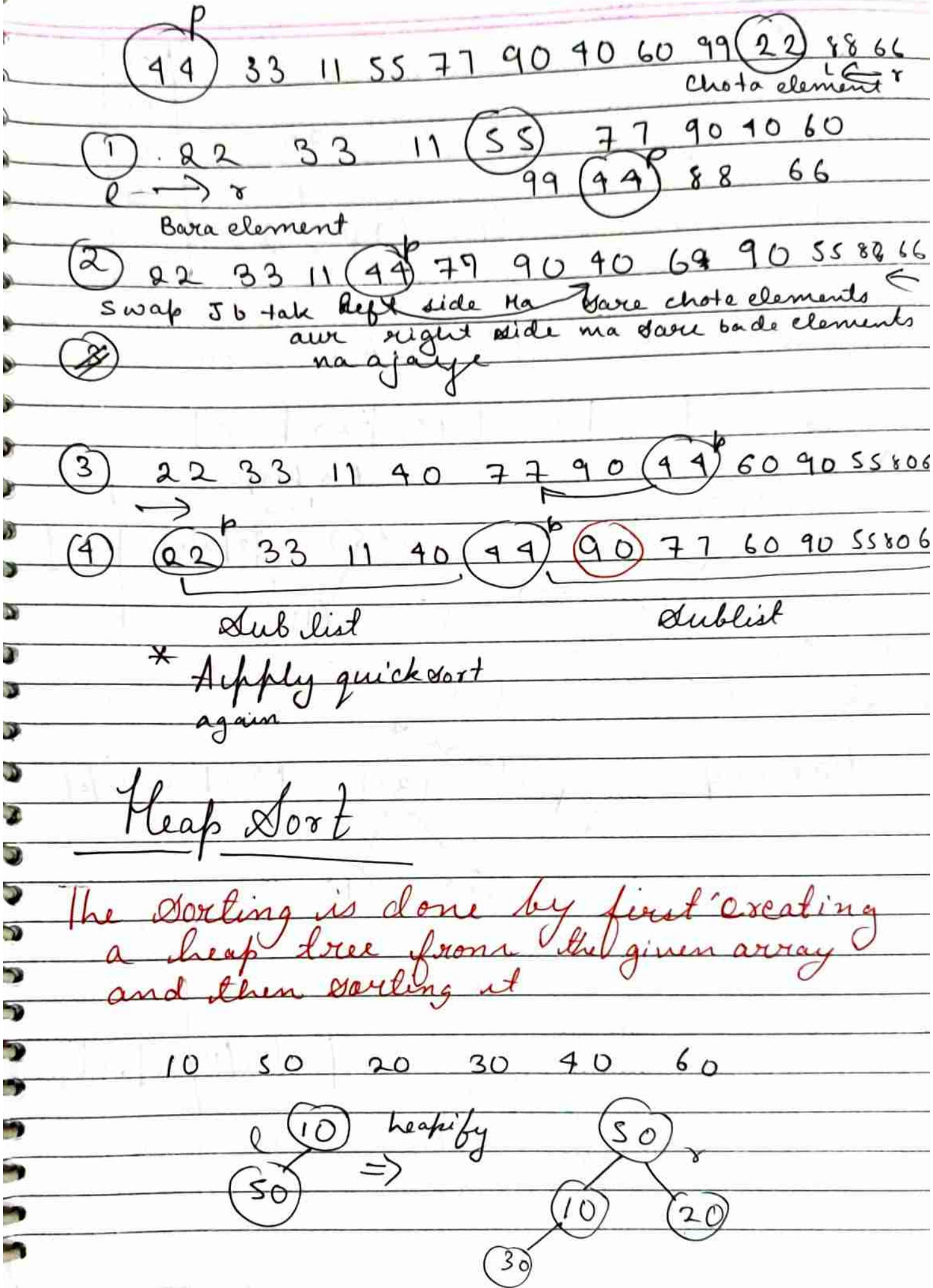


Eg: 44 33 11 55 77 90 40 60 99 22 88 66

divide

→ larger

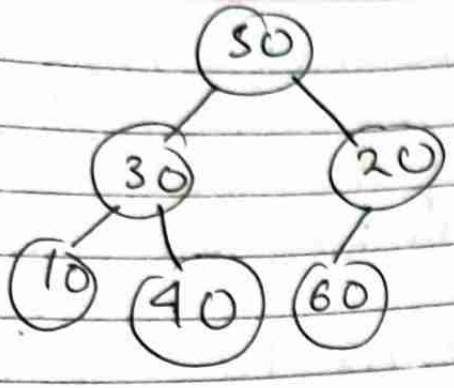
smaller ←

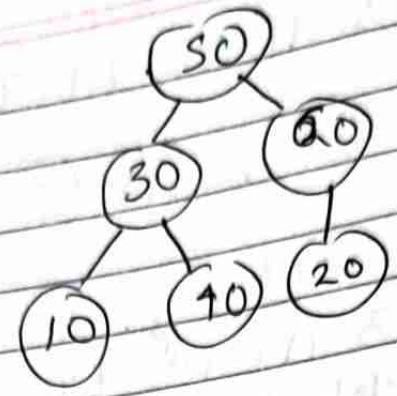


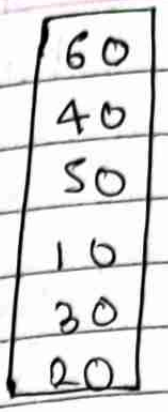
Heap Sort

The sorting is done by first creating a heap tree from the given array and then sorting it

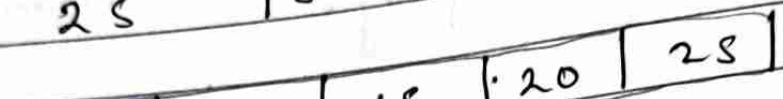
10 50 20 30 40 60



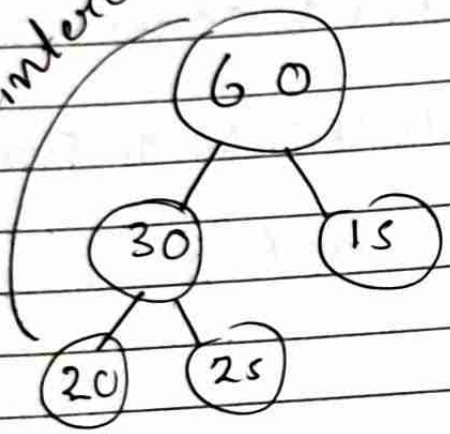




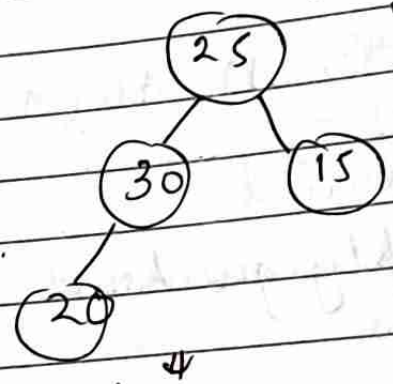
Q. 30 25 15 60 30



interchange

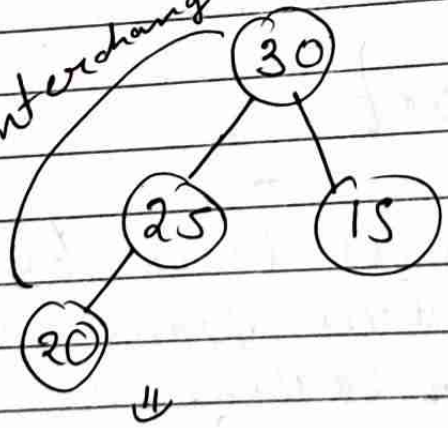


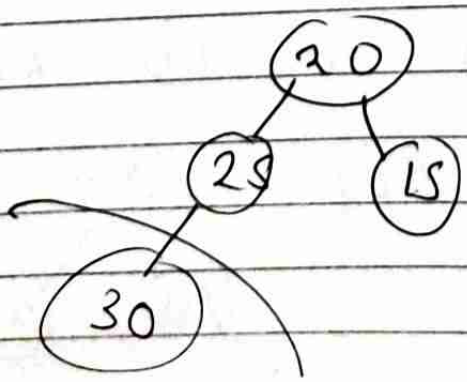
not a heap tree

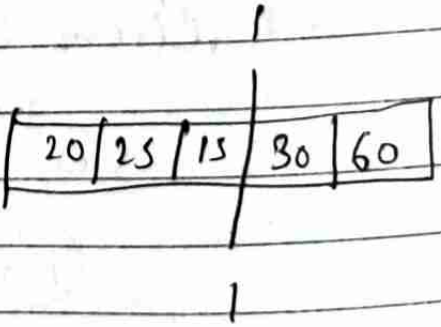


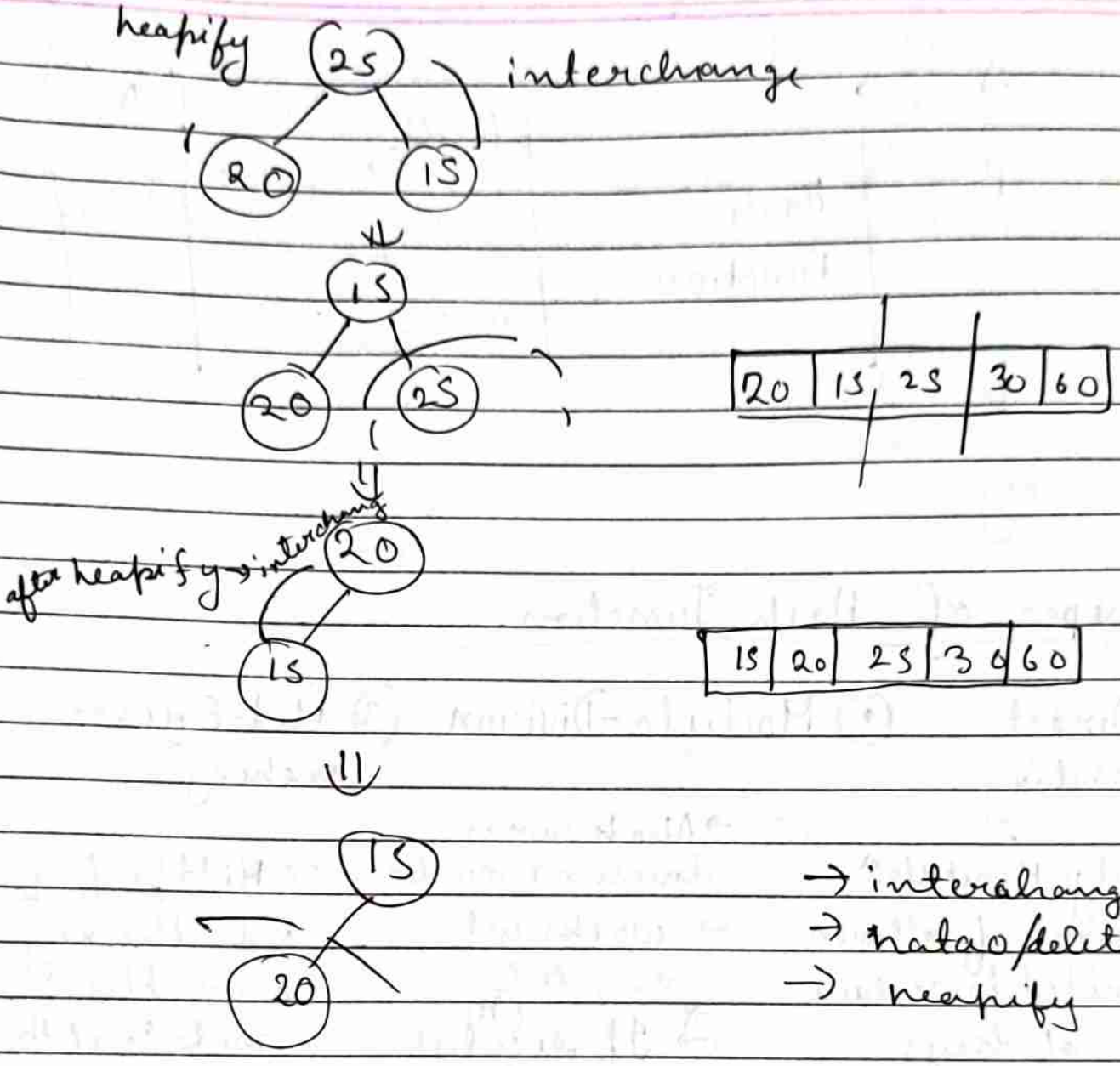
Max heap

interchange







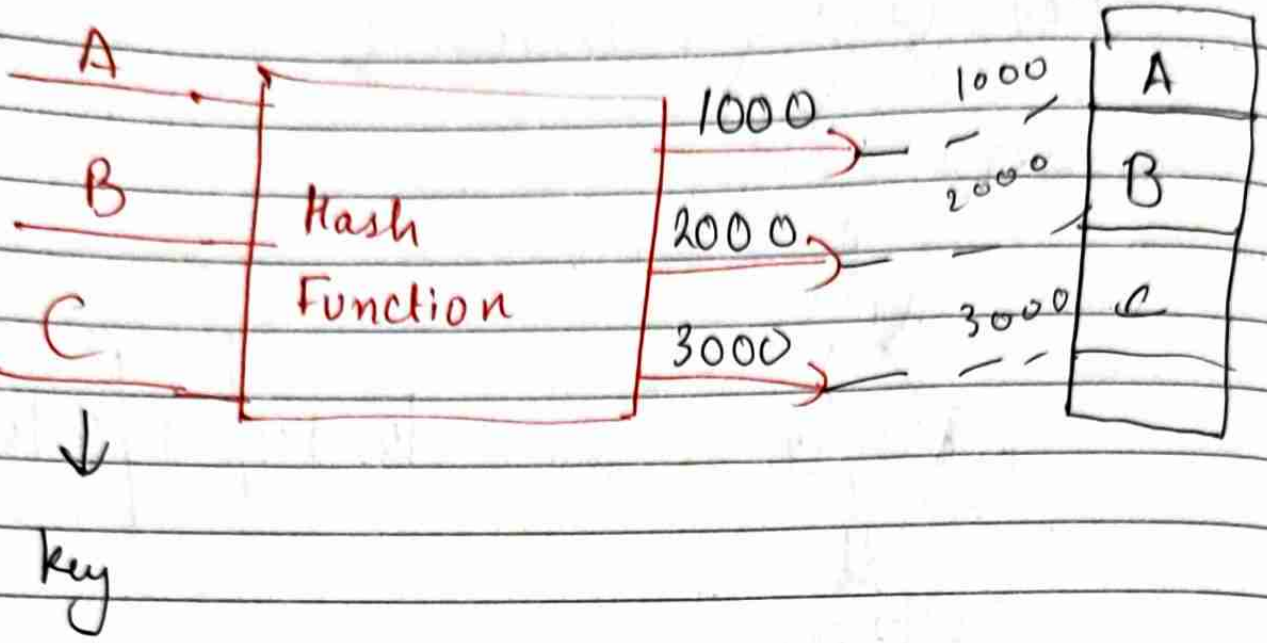


→ interchange
→ data o/delete
→ heapify

Hashing

→ Hashing involves less key comparision and searching can be performed in const. time

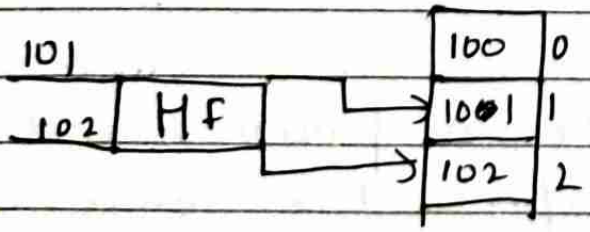
→ The goal of hashed function is to find target data in only one test ie O(1)



Types of Hash Function

(1) Direct hashing

→ No algo Manipulaⁿ
→ Min no. of collision
→ limited to certain no. of keys
→ Not suitable for large key values



(2) Modulo-Division

→ Also known as division remainder
→ works wd any list
* → If size list (n) is a prime no. then few collisions encountered

$h(k) = k \% n$, n=10

101 % 10 = 1
100 % 10 = 0
111 % 10 = 1

(3) Mid-Square hashing

→ Middle of Sq
→ Sq. the key and the middle of the req. no. of address will be the addr

eg k=35 n=2
$k^2 = 1225$
h(k) = 22

(5) Substraction hashing
→ substract a fix no from key $h(k) = k - c$
→ No collision
→ suitable for small size list
$h(k) = k - c$

(4) Folding Hashing

Fold Shift Hashing
→ key value divided into parts and then added to get the req. address
eg 123 456 789

123
456
789
discard 1368

$h(k) = 368$

Fold Boundary hashing
→ left and right nos are folded on a fixed boundary b/w them
eg: 123 456 789
leave as it is

321
987
discard 456

1764

$h(k) = 1764$

Collision Resolution:

→ A collision resolution occurs when a hashing algorithm produces an address for a key and that address for a key and address is already occupied.

CR
- Open Addressing
  - Linear Probe
  - Quadratic Probe
  - Pseudo Random
- L.L

$h(k) = k \% n$

$n = 10$

$h(k) = k \% 10$

$h(101) = 101 \% 10 = 1$

$h(111) = 111 \% 10 = 1$

$y = ax^{Key} + c \rightarrow$ const

$h(k) = Y \% n$

① Open Addressing:

Home area Address are searched for an open or unoccupied element where new data can be placed.

★ Linear probe

$h(k) = k \% n$

$h(101) = 101 \% 10 = 1$

$h(111) = 111 \% 10 = 1$

$i = (i+k) \% n$

$k = 1, 2, 3, 4$

$2 \% 10 = 2$

★ Quadraticprobe

$h(k) = k \% n$

$h(101) = 101 \% 10 = 1$

$h(111) = 111 \% 10 = 1$

$i = (i + k^2) / n$

$k = 1, 2, 3$

$2 \% 10 = 2$

$10 \overline{)2}\ 0$

2

$5 \% 10 = 5$

② Linear linked list.

* Reverse a ll iterative method

```
void reverse()
{
struct node * prevnode, *currentnode
        * nextnode;
 Prevnode = 0;
Current node = nextnode - head;
while (nextnode != 0)
 {
 nextnode = nextnode → next;
 currentnode → next = prevnode
  prevnode = currentnode;
   current node = next node;
 }
 } head = Prev node;
 }
```

* Del from Pos

```
struct node
 {
  int data;
 struct node * next;
 };
struct node * head, * temp;
Del_from_Pos()
{ struct node * nextnode;
 int Pos, i = 1;
 temp = head;
 Printf = ("Enter the position");
 Scanf ("%d", &Pos);
 while (i < Pos - 1)
 temp = temp → next;
  i++;
 }
 nextnode = temp → next
 temp → next =
    nextnode → next;
  free (nextnode);
 }
```

Deque

```
#define N 5
int deque[N];
int f = -1, R = -1;
void enqueue(int x)
if (f == 0 && R = N-1) || (front = rear+1))
{ printf("queue is full"); }
else if (f == -1 && R == -1)
  {
    F = R = 0;
    deque[F] = x;
  else if (front == 0)
  {
    f = N-1;
    dequeue[F] = x;
  }else {
      f --;
    deque[F] = x; }}
```

Diff b/w Struct & Union.

```
#include<iostream>
using namespace std;
struct stu
{
  int marks; // int 4 bytes
  float avg ; // float 4 bytes
  double salary ; // double 8 bytes   (4+4+8=16)
};
union stu2
{ int roll; string name;
  int marks;
   float avg;
   double salary; // double 8 bytes  (8)
};                          For largest data-type
int main()                    allocate memory.
{
  union stu s;
  cout<<"Enter student roll number :";
  cin >> s.roll;
  cout<<" roll number"<<s.roll;
  Cout <<"Enter the student's name.";
  cin >> s.name;
  Cout <<" Student name "<<s.name;
  cout<<" Enter the student's marks:";
   cin >> s.marks;
    return 0;
```

Enumeration: is a user defined name that consists of integral constant

Syntax:- week → 7 days.

enum enum-name { value1, value2, ... valueN };

Start होता है ।

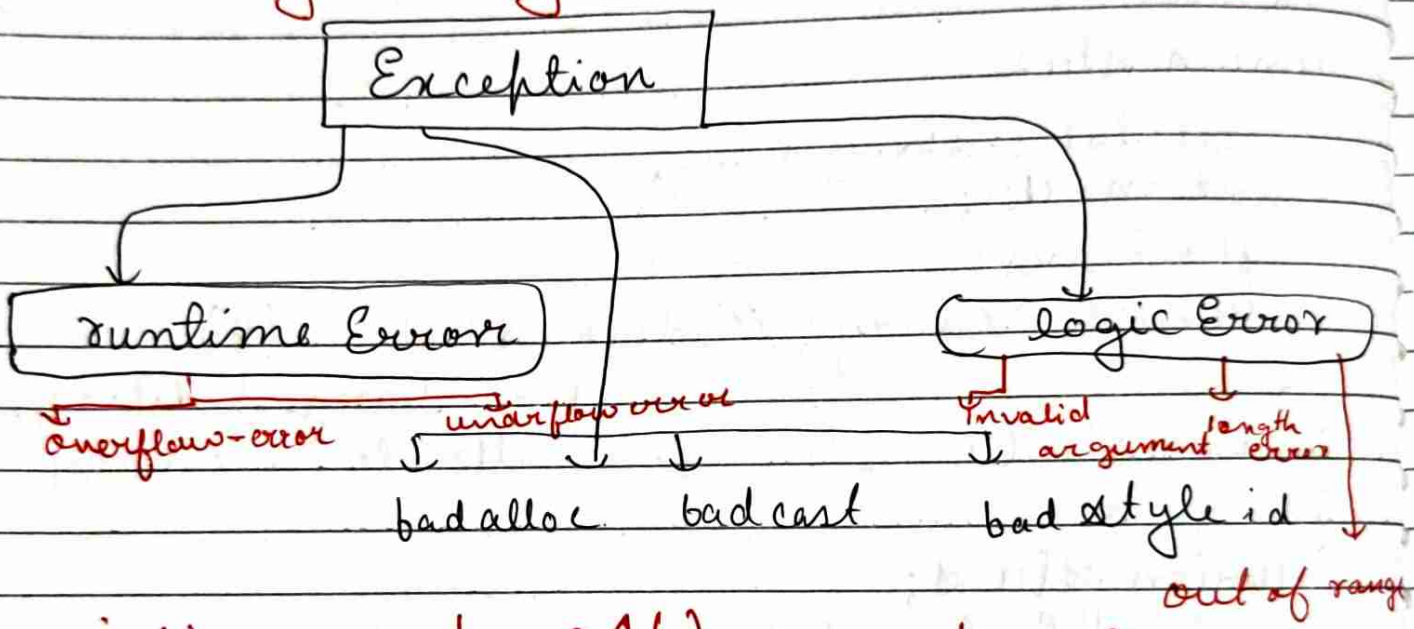
cout << value1 ; ⇒ 0

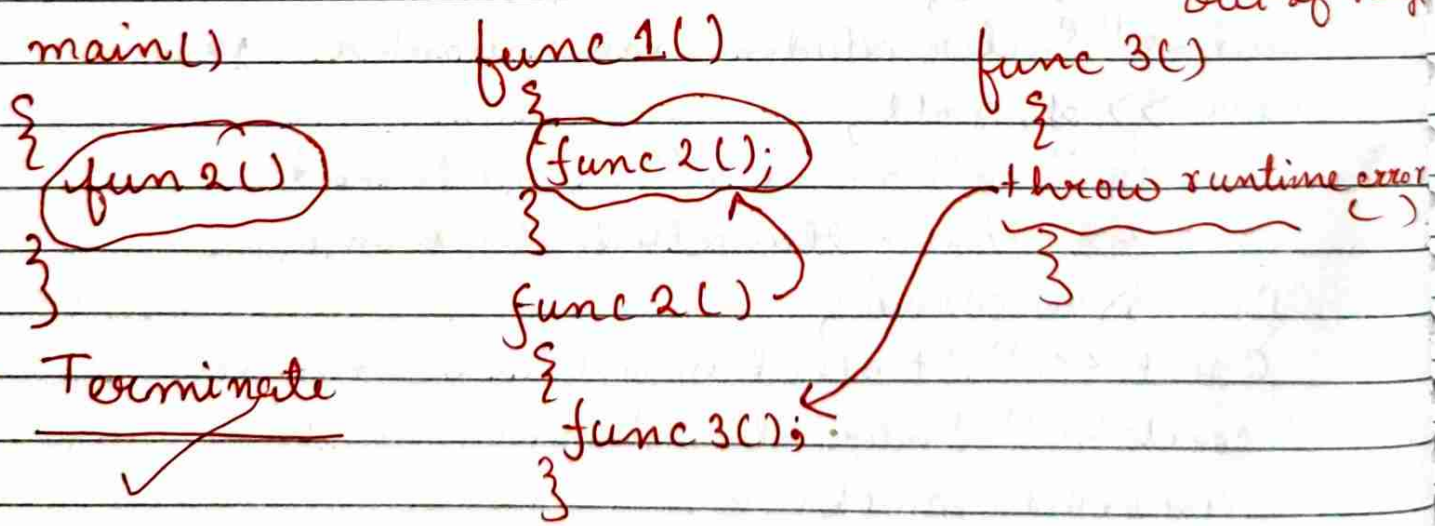
enum . name var = value2;

cout << var; ⇒ 1

Stack Unwinding

Standard library Exception Hierarchy





Stack Unwinding occurs when an exception is thrown but not caught in a particular scope the function called stack is unwound and an attempt is made to catch exception in the next outer try catch Block. Unwind the function called stack means the funcⁿ in which when exception was caught terminates and all local variables in that function are also destroyed And the control returns to the statement that originally was invoked in that funcⁿ so if a try catch block encloses this statement an attempt is made to catch this exception if a try catch block doesn't enclose the statement stack unwinding occurs again.

If no catch handler ever catches this exception function terminal is called to terminate the program.

Note: whatever types of exceptions u have occured you have to handle them carefully.

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

void function3() {
    cout << "In function 3" << endl;
    throw runtime_error("runtime error in
            function3");
}
void function2() {
    cout << "function 3 is called inside function 2" <<
    function3();                                        endl;
}

void function1() {
    cout << "function 2 is called inside function 1"
         << endl;
    function2();
}

int main() {
    try {
        cout << "funtional1 called " << endl;
        function1();
    }
    catch (runtime_error & error) {
    cout << "Exception occured : " << error.what()
                                   << endl;
    cout << "Exception handled in main function"
         << endl;
    }
}
```

logic error X — Ok terminate

O/P

function 1 called
function 2 is called inside function 1
function 3 is called inside function 2
In function 3
Exception occured : runtime error in function3
Exception handled in main function

(manual) Typecasting

Explicit ← → Implicit (automatic)

(type) expression
or
(type) expression

Ex:- int m = 5;
float y;
{After} y = (float) m / 2; → int int → o/p float
{Before} o/p → int

Char → int → long int → float → doub
Low High

int m = 15
float x = 3.1
y = m * x;
float
high rank
y → float o/p.

Q. Replace cat with dog

```
#include <iostream>
using namespace std;
int main()
{
  string str1 = "This is C language";
  string str 3 = "java language";
```

Copy Constructor

not compulsory

syntax:-

```
ClassName ((const) ClassName &old_obj);
```

```
#include<iostream>
using namespace std;
class Point
{
private:
  int x, y;
public:
Point(int x1, int y1)
  {
     x= x1;
     y = y1
// Copy constructor
Point(const Point & p2)
  {
     x = p2.x;
     y = p2.y;
int getX() {
   return x; }
int getY() {
   return y; }
int main()
{
  Point p1(10, 15); // Normal parameterize constructor is called here
```

Point p2 = p1; // Copy Constructor is called here.

// let us access values assigned by constructors.

cout << " p1.x = " << p1.getX() << ", p1.y = " << p1.getY();

cout << " \n p2.x = " << p2.getX() << ", p2.y = " << p2.getY();

return 0;

}

O/P

p1.x = 10, p1.y = 15

p2.x = 10, p2.y = 15

CC

Used to :-

- Initialize one object from another of the same type.
- Copy an object to pass it as an argument to a function.
- Copy an object to return it from a function.

Dynamic initialization of object in C++ using Dynamic Constructors.

- Dynamic intialization of object refers to initializing the objects at a runtime i.e., the initial value of an object is provided during the runtime.
- It can be achieved by using constructors and by passing parameters to constors.
- This comes in really handy when there are multiple constructors of the same class with different inputs.

Dynamic Constructor

- The constructor used for allocating the memory at runtime is known as the dynamic constructor.
- The memory is allocated at runtime using a new operator and similarly, memory is deallocated at runtime using the delete operator.

Prog, new is used to dynamically initialize the variable in default constructo and memory is allocated on the heap.
The obj of the class geek calls the functi and displays the value of dynamically allocat variable i.e ptr.

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

class geeks{
    int* ptr;

public:
    // Default constructor
    geeks()
    {
        // Dynamically intializing ptr
        using new

        ptr = new int;
        *ptr = 10;
    }
    // Function to display the value of ptr
    void display()
    {
        cout << *ptr << endl;
    }

};

// Driver Code
int main()
{
    geeks obj1;
```

```
// Function call                    O/p
obj1.display();                     10
return 0;
}
```

Dynamic deallocation:

In eg, delete is used to dynamically free the memory.
The contents of obj1 are overwritten in the object obj2 using assignment operator, then obj1 is deallocated by using delete operator.

```
#include <iostream>
using namespace std;
int * ptr;
public:
// Default constructor
geeks()
{
    ptr = new int;
    * ptr = 10;
}.

// Function to display the value.
void display()
{
    cout << " Value: " << *ptr << endl; n
}
};
```

```
// Driver Code
int main()
{
    // Dynamically allocating memory
    using new operator
    geeks* obj1 = new geeks();
    geeks* obj2 = new geeks();

// Assigning obj1 to obj2.
    obj2 = obj1;

// Function Call
    obj1 -> display();
    obj2 -> display();

// Dynamically deleting the memory
    allocated to obj1.

    delete obj1;

    return 0;
```

O/P

Value : 10
value : 10

Overloading Unary Minus

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

class space
{
  int x;
  int y;
  int z;
public:
  void get data(int a, int b, int c)
  void display(void);
  void operator-(); //overload unary minus
};

void space::get data(int a, int b, int c)
{
    x=a;
    y=b;
    z=c;

void space::display(void)
  {
    cout<<"x = "<<x<<" ";
    cout<<" y = "<<y<<" ";
    cout<<" z = "<<z<<"\n";
```

RTCN :: operator operator-symbol()

```
void space :: operator~()
  {
    x=-x;
    y=-y;
    z=-z;
```

→ op x or x op.

```
};
int main()
{
  space S;
  S.getdata(10,-20,30);
  cout << "S : ";
  S.display();
  -S;                    //activates operator-() function
  cout << "-S:";
  S.display();
  return 0;
}
```

func^ invoked

O/P

S : x = 10 y = -20 z = 30
-S : x = -10 y = 20 z = -30

Overloading + Operator

```
#include <iostream>
using namespace std;
class complex
{
  float x;
  float y;
```

```
public:
  complex() { }
  complex(float real, float imag)
  { x = real; y = imag; }
  complex operator+(complex);
  void display(void);
};

complex complex::operator+(complex c)
{
  complex temp;
  temp.x = x + c.x;
  temp.y = y + c.y;
  return(temp);
}
void complex::display(void)
{
cout << x << "+j" << y << "\n";
}
int main()
{
  complex C1, C2, C3;
  C1 = complex(2.5, 3.5);
  C2 = complex(1.6, 2.7);
  C3 = C1 + C2;

  cout << "C1 = "; C1.display();
  cout << "C2 = "; C2.display();
  cout << "C3 = "; C3.display();
  return 0;
}
```

O/P

C1 = 2.5 + j 3.5
C2 = 1.6 + j 2.7
C3 = 84.1 + j 6.2

Benefits of Exception Handling

(1) Exception handling can control run time errors that occur in the program.

(2) It can avoid abnormal termination of the program. also shows the behaviour of program to users.

(3) It can provide a facility to handle exceptions, throws message regarding exception and completes the execution of program by catching the exception.

(4) It can separate the error handling code and normal code by using ~~no~~ try - catch block.

(5) It can produce the normal execution flow for a program.

(6) It can implement a clean way to propagate error, that is when an invoking method cannot manage a particular situation, then it throws an exception and asks the invoking method to deal with such situation.

(iii) It develops a powerful coding which ensures that the exceptions can be prevented.

(iii) It also allows to handle related exceptions by single exception handler. All the related errors are grouped together by using exceptions. And then they are handled by using single exception handler

(iv) Separation of normal code from error handling code eliminates the need for checking the errors in normal execution path there by decreasing the cycles.

(x) A failed constructor can also be handled by throwing an exception. The code for try, catch and throw blocks must be written in the constructor itself.