Page No.
Date / 20

C++ LANGUAGE

```
# include <iostream>

int main () {

    std:: cout << "Hello World";
    return 0;

}
```

Basic C++ program

→ C++ was created by Bjarne stroustroff in 1974 as an extension of C.

→ C++ is used to make fast controlled performance, more control over system resources + memory managment.

→ High performance.

```
# include <iostream>
```

Header file

```
int main () {

    cout << "Hello World";
    return 0;

}
```

Function Body

Page No.
Date / 20

Header file: increase the functionality of our program.

iostream :- (input output) Help in doing input and in the program.

int main :- entry point of the program.

all other function will be call under the main function.

int means Value will be return in return type.

std::cout << "Hello World"; | print syntax.

→ insertion operator.
→ namespace file.
→ scope resolution operator.

or instead this

you can write

using namespace std; → before int main

return 0; means main ki Value 0 karo and shows succesfully running of the program

Page No.
Date / 20

Chapter 3 Variable & Comments

low level language :- Near to hardware

High level language :- Logically saved & away from hardware.

example :- [google] or form.

query → Google → pages

C and C++ are low level language.

Variable :- They are containers that store our data.

→ int :- -1, 0, 1
→ float :- -1.23, 1.24
→ char :- a, b, Mudit.
→ Double :- 1.234....
→ Boolean :- True / False

Comments :- To not use the text in the code.

// → single line comment.
/* */ → ~~Multi line comment.~~

syntax for printing a Variable.

cout << Sum;] this will print the value of sum provided by us.

cout << "Hello World" << Sum;

We need to leave space here if we want diff b/w world and sum.

Page No.
Date / 20

Chapter 4 Variable scope and Data types.

int sum = 34

sum ● is an int container which store ● the value 34.

1. local Variable
2. Global Variable. → based on scope.

Scope of a variable is the region in code where the instance of variable is valid.

Local Variable :- declared inside the brace of any function and can be accessed only from there.

Global Variable :- ● declared outside any function and can be accessed from anywhere.

Data Types

define the type of a variable can hold.

Built in
user defined
Derived → types.

Built in :- int, float, char, double are built in data types.

User defined :- structure.
Union.
Enum.

Derived :- Array.
function.
pointers.

In can only be declared in string type.

If boolean is true it will return 1 and 0 for false.

chapter 5 Basic input and output

input stream :- Direction of flow of bytes take place from input device to the main memory.

output stream :- Direction of flow of bytes from main memory to output device.

Input Stream

Cin >> num1;
↳ extraction operator.

Syntax to take value from the user.

Chapter 6 Header files and operator

Header files

→ System Header files
→ User defined Header files

System Header files

It comes with compiler.

User defined Header files

defined by the programmer in the programs

Cpp refrance to read the Header files

< abc.h > duplicate → mean the abc file is not going to be use in the future version

Operators

endl or \n both can be used to enter the new line.

→ **Airthmetic operator**

+ , - , \ , X , % , ++ , --

↓

increment & Decrement

→ [assignment operator]

use to assign value of operator

→ [comparsion operator]

>>, <<, =, <=, >=, !=

These all will return value in form of boolean

True/ false

and in C++ True is defind by 1 and false by 0.

→ [logical operator.]

&&, ||, !

Chapter 7 Refrance Variable & Type Casting

If you want to use your global variable in other function then

cout << "The Value of c" << c] → print local variable of name c

Cout << "The Value of c" << ::c] → print global variable of name c

By default Decimal number are taken to be as double.

[34.4] → default use as double

To use it as float

34.4(f) → float.

and 34.4(l) → long double.

Refrance Variable

```
float x = 455;
float &y = x;
```

↳ y is a refrence variable pointing toward x.

Type Casting

Changing variable type from 1 to another.
int → float → double etc.

```
float b = 456
cout << (int) b;
```

→ typecast from float to int.

output = 45

or

int(c) = int (b) → taking value of b as an int.

Chapter 8 :- Constants, Manipulator, operator precedence.

Constants

Variable which you can't change again in programs after they are declared.

[const int a = 30;] constant of name a is created.

Manipulator

operator which help you to in display formatting.

→ endl → to switch to next line

include <iomanip>

cout << (setw(4)) << a;

will print value of a in minimum 4 space.

without set w :- a = 4
with set w :- a = ___4

Page No.
Date / 20

operator Precedence

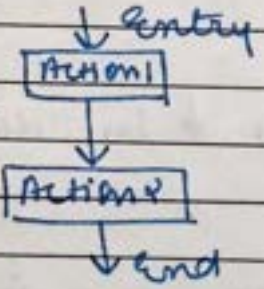
all are operator in C++ have their own order of precedence.

Chapter 9 control structure, if else and switch case statement.

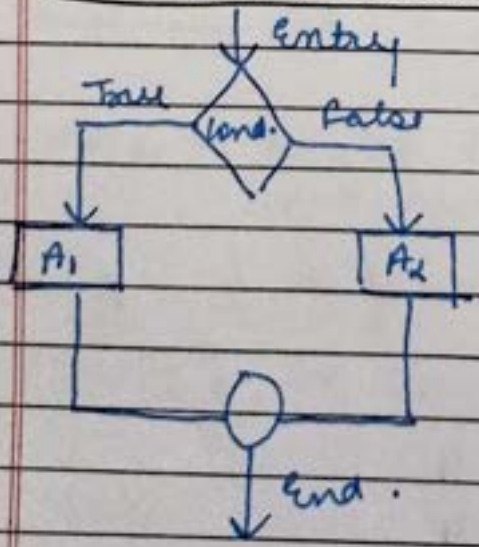
Sequence structure
Selection structure
loop structure

} Basic control structure

Sequence structure

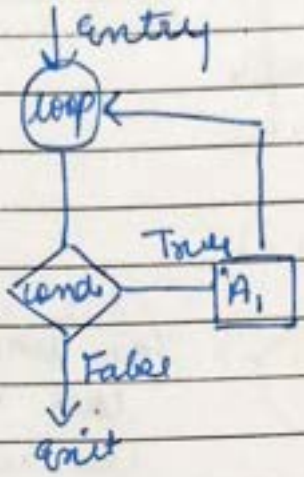


Selection structure



Page No.
Date / 20

loop structure



if else statement

```
if(condition)
  {
    Body
  }  → no comma
else (Body)
```

if else-if statement

```
if (condition) → no comma
  { Body
  }
else if ( condition)
  { Body
  }
else {
    Body
  }
```

if any one of if or else-if get true then loop will get break.

Switch Case statement.

~~switch (condition Expr~~
Switch (expression)
{
case (condition):
{ Body }
break; → It is imp else all the case ~~will~~ below this case will get print.

default:
{Body}
break; → if none of case get true then default will get print.
}

Chapter 10 for, while, do while loop

for (initialization; condition; iteration)
{
Body
}

This is syntax for for loop.

loop will work till the condition is true.

while loop

```
while( condition)
{
   Body
}
```

do while loop

```
do {
    Body
}
while( condition);
```

do while will process for atleast one even if the condition is false.

Chapter 11 Break and continue statement

Break statement are used to break the loop (for or while or do while)

```
for (insertion; condition; iteration)
{
    Body
    if ( condition )
    {
        Break;
    }
}
```

→ when this condition get true the loop will be exited even if loop condition was true.

Body can be written both above the break condition and below break condition.

Continue

It is use to skip a particular condition as when continue condition get true then every written below continue in that loop will be skipped.

Chapter 12 Pointers in C++

pointer:- data type which hold address of other data type.

int a = 3;

int *b = &a; → initializing pointer b which have the address of a.

&a = b = address of a
a = *b = Value of a is 3

(Revise Chapter 50)

int **c = &b

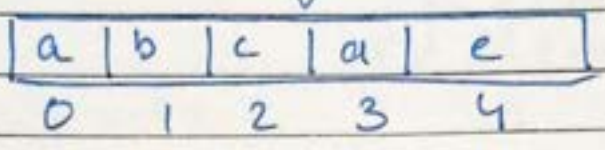
pointer to pointer Variable.

pointer which hold the adress of pointer Variable.

Page No.
Date / 20

Chapter 13 Arrays & pointers

an array is a collection of similar type of item in contiguous memory locations.



mark[0] = a , mark[1] = b

<data type> name [ ] = { };

or

<data type> name ;
name [0] = — ;

These two are the method of how can we add array in our list.

you can also change the value of an array just by same syntax you initialize an value

name[n] = <Value>;

Use of pointer and array together.

int a = 3
&a → used as address of a

marks → address of I block

&marks = X

pointer.

p = marks;

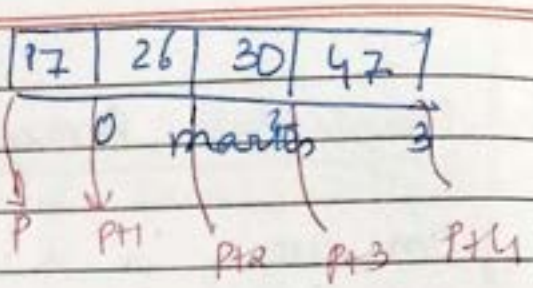
p++;

*(p) = 17

*(p+1) = 26

*(p+2) = 30

*(p+3) = 47



so in array

int *b = marks → initializing an array

pointer airthmetic

address = address + i * size of data type

new (p+i)

use (p)

chapter 14 structure, union & enums

Structure

In array we can only store the value of same type ie only ~~array~~ int, string etc.

to store all together we make structure.

creating a structure

typedef struct <name> → name of the structure

{

int <name1>

char <name2> → feature of structure

} sp → This can use in order to use sp in place of struct <name>.

Initiating Values

```
struct <name> <username1>;
    <username>.<name1> = <Value1>;
    <username>.<name2> = <Value2>;
```

To print

```
cout << <username>.<name1>  }→ print Value1
```

We can also create many username under the name with different Value.

Union

```
union <money>
{ int <rice>;      // 4 byte
  char <car>;      // 1 byte
  float <spend>;   // 4 byte
};
```

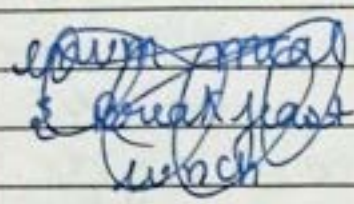
In Union I can use only 1 from these.

these are the byte that it taken.

so in (union) it will generate maximum from all these as a memory. In order to save space.

if you try to initialize a Value the compiler will throw a garbage value. if you try to find the Value of 1 you allocate first and will give correct Value of the one you allot after.

Enum



enum <meal>
{
<Breakfast>;
<lunch>;
<dinner>
};

meal m1 = < ? >
will store value in form of 0 1 2.

cout << Breakfast;
cout << lunch;
cout << dinner;

output = 0 1 2

Chapter 15 function & function prototype

functions:- Do once use forever.

```
int sum (int a + int b)   → This is the functions we create.
{
  int c = a+b;
  return c;
}
```

```
int main
{
cout << sum (9,10)
```

output = 19

Page No.
Date / 20

Function prototyping

It tells the compiler in advance that this is a function which is going to come.

```
int sum(int a, int b);   → This is function prototype.
int main()
{
}
```

If you not declare this and declare your function at last then compiler will give error.

```
int sum(int a, intb);
{
}
```

int sum(int a, b)
int sum(a, b)
] These are not acceptable.

int sum(int a, int b)
int sum(int, int)
] These are acceptable

parameter defined in a function are formal parameters and variable defined in main function are actual parameter.

void A(void) → it doesn't contain any value.

means it doesn't return any value

Page No.
Date / 20

Void is optional while declaring in C++

Chapter 16 Call by Value & Call by Reference

Call by reference using pointer

```
void swap(int *a, int *b)
{
int temp = *a
*a = *b
*b = temp
}
```

```
int main() {
    x=4, y=5
swap(&x, &y)
```

output
x=5, y=4.

↓
Here we give address of our variable

and address of variable in a memory remain same

Call by reference using C++ reference variable

```
void swap(int &a, int &b)
```

reference variable.

```
{
    a, b
}
```

```
{ int temp = a;
  a = b;
  b = temp;
}
```

output x=5, y=4

```
int main()
{
        x=4, y=5
swap(&x, y)
```

Here we make refrance variable in our function with name &a & &b so they start pointing toward x and y

So as whenever we change the value of them in that function x and y also change their value.

```
int & swap(int & a, int & b)
{  int temp = a
   a = b
   b = temp
   return a
}
```

→ This is now returning value as a

```
x = 4, y = 5
swap(x, y)
```

→ This is become value of x as it is returning a and is pointing to ward x

Page No.
Date / 20

Chapter 17 Inline function, Default argument & constant argument

Inline function

an enhancement we make while creating a function to improve execution and speed of the program.

Use only when function is not big else it will take lot of memory and space.

In Recursion never use inline function.

Static Variable

```
static int c=0;
c=c+1;
```

] use in function

at first c will be 0 and c+1 will happen and c become 1 and next time whenever we will call function

it will not take c=0 else it will have c=1 and will skip (c=0) and c+1 =2 will happen.

initialization line will run only one and the value of c will be saved and that

value will be executed at rent time.

Default argument

these are the value which you get by default if you have not given any value.

```
float money (int Currency, float factor = 1.05
{
return Currency * factor.
}
```

Default argument

```
int main()
{
int Rmoney = 100000;
cout << money (Rmoney);
```

→ Here we have not defined any factor so our default will be used.

output = 150000

```
cout << money (Rmoney, 1.1)
```

→ Here we have defined our factor so this will be used not default.

Output = 110000

Page No.
Date / 20

Whenever we make a function inline it is a request from compiler

so it depend on compiler whether it will make a inline or not.

In default variable, one should define main variable whose value is to be compulsary befor and then to define default at the last.

Constant argument

one whose value cant be modified by the function.

```
int sum ( const int a)
```

const argument

chapter 18 Recursion & Recursive function.

Recursion :- process in which function call itself again and again until it satisfy the base condition.

```
int factorial (int n)
{
if
return n * factorial (n-1);
}
```

```
int factorial (int n)
{
    if (n<=1) → base condition.
    {
        return 1;
    }
    return n * factorial (n-1);
}
```

this loop will work until the base condition is true / fulfilled.

Chapter-19 function overloading with example.

function Overloading :- specification of more than one function of the same name in the same scope.

```
int sum (int a, int b)
{
return a+b
}
int sum (int a, int b, int c)
{
return a+b+c
}
```

This is function overloading. we can make 2 function with same name. this situation is called function overloading.

Another example of function prototype

Page No.
Date / 20

Important

Chapter :- 20 object oriented in C++

OOPs :- object oriented programming

why oops?

→ earlier C++ was known as C with classes.

→ as size of program ↑ readibility, maintainibility and bug free nature of program decreases.
using function create lot of mess in big programming.

→ we use classes to solve the problems by modeling program as real world scenario

example Animal
→ type
→ name
run()
sound()

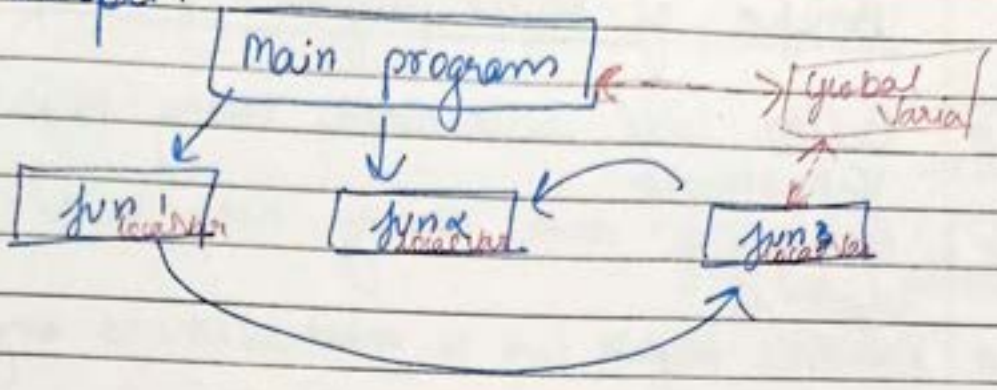
dog ← object we made.
Tommy : Animal obj

tommy run()

procedure oriented programing (POP)

- consist of writing of set of instructions for the computer to follow
- main focus is on function not on flow
- function can either use local or global data.
- Data moves openly from function to function.

example.



Object Oriented Programming (OOP)

- Work on concept of classes & object
- Class is a template to create object
- Treats data as a critical element.
- Decompose the problem in object and builds data and function around the object.

Basic concept in object oriented programing

Classes:- Basic template for creating objects.
Objects:- Basic runtime entities.
Data abstraction & Encapsulation:- Wrapping data and function into single unit. (four basic of oops)
Inheritance:- property of one class that can be inherited to other.
Polymorphism:- ability to take more than one forms.
Dynamic Binding (late Binding):- code which will execute is not known until program runs.
Message passing:- Object.message call format.

Page No.
Date / 20

Benefits of object programing language

- Better code reusability using objects and Inheritance.
- Principle of data hiding help build secure systems.
- Multiple objects can co-exist without any interference.
- Software complexity can be easily managed.

Chapter 01 Classes, Public and Private

(Revise structure in chapter 14)

→ we need to make classes even when structure is very good because struct are not protected and anyone can access any value in it.

→ we can't add function inside structure.

Syntax to make a class

```
class <Class name>
{
  private:
    int a, b, c;

  public:
    int d, e;
```

this is we created a class

Function prototype. → new function declare

Void setdata(int a1, int b1, int c1);
Void get data() {
cout → a
b
c
d
e
};

→ while creating function we don't write ; but while creating class we write ;.

(class is created by programmer.)

Void setdata(int a1, int b1, int c1).

~~Void Program~~

Void <class name> :: setdata(int a1, int b1, int c1)

we have to write this → scope resolution operator

∵ we have defined our function inside a class with name <class name>

{
a = a1 ;
b = b1 ;
c = c1 ;
} → no ; ∵ it is a function.

int main () {
<class name> <object>; → This is the object of our class and we can make many object.
<object>.d = 32
<object>.e = 18

assigning int d and e of public class their value but we cant do so for a, b, c as they are in private.

to assign value of a, b, c.

<object>.setdata(1, 2, 3);

↓

our function where we define value of a, b, c

<object>.getdata();

↓

our function where we cout all the value.

output :

a = 1
b = 2
c = 3
d = 32
e = 18

Chapter 22 oops recap & Nesting of Member function.

oops recap

classes - extension of structure in C++

structure & limitation

→ no security
→ no functions

classes → structure + more
classes → can have method and property.
classes → can make few member as private & few as public.

Page No.
Date / 20

→ by default everything in classes are private.
structure in C++ are typedef.

→ you can declare object with along class declaration like.

```
class <class name> {
    // definations
} harry, rohan, louis;
```

→ audit.salary make no sense if salary is private.
private variable can't be set directly.

Nesting of a Member function

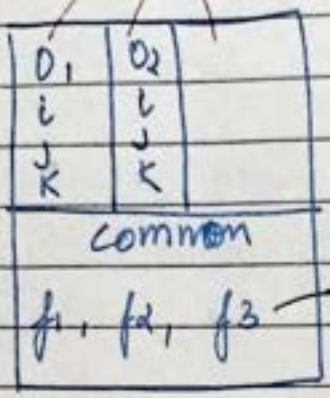
when we declare a function of a class inside the other function of a class is known as Nesting of a member function.

We can do nesting of our private member because we can't use them without declaring them.

Chapter 23 object memory allocation & using array

whenever we make a class memory is not assign to class but memory is assign when we create a object of that class.

diff thing of diff object take diff space for them



things that are common for all object are store that one place.

Memory diagram

How array are use in class

array are use normally in the classes you can add array in class with normal symbol.

if you add array in private you have to assign its value through function and you can normally assign value if you are taking array public.

Chapter-24 static Data Member & Method in C++

static Variable are defined outside the class.

Class employ {
static int count; ← can be any of private/public
};

not here.

int employee :: count; → if you want to start it from number other than 0 the initialize here

Page No.
Date / 20

By default a static variable is initialized by 0.

When we make a variable static in a class means that all object of the class are sharing that static value and doesn't have this own private value.

§ static variable can be used to refer to the common property of all object §

```
class employe
{ int id
  static int count  ---> private static var.

public
    void setData(void)
    cout<< "Enter the Id";
    cin>> id;
    count ++ ;
    void getdata (void)
    cout<< the id of  cout<< Id "and" and anothe number<< count
int employe::count;  ---> static variable formed.

int main()          3 object of class employe.
{ employe modi, komal, komu
  modi. setData ()  { During this the count varia
  modi. setData ()    ble is increased to 1
                    }

  komal. setData ()  § loop will start again fo
  komal. getData()   it but now value of
                     static will remain
                     initially and will go to 2)
```

If we dont use static then value of count will be 1 for all of the object.

static variable are also called object class variable.

Static member function

It can only access all the static member and if you try to give non static variable it will throw error.

```
static void Get count {
    cout << count;
}
```

if count is static it will work finely else it will throw error.

Chapter 25 Array of objects & passing object as function

```
class complex {
    int a;
    int b;
public:
    void setData (void int v1, int v2) {
        a = v1;
        b = v2; }
    void Setsum { complex o1, complex o2 }
        a = o1.a + o2.a   (o1 and o2 are object used as variable)
        b = o1.b + o2.b
```

class object.

Page No.
Date / 20

Hence we can use object as a function also.

Chapter 26 friend function in C++

properties of friend function :-

- Not in the scope of class
- Since it is not in the scope of the class it cannot be called from the object of the class
- can be invoked without the help of any object
- usually contains the argument as object.
- can be declare inside public and private section of the class.
- Cannot access member directly by their name and need object_name.member-name to access any member.

```
class complex{
    int a
    int b
    public:
    void set data (int n1, n2)
    {a=n1;
     b=n2;
    }
    void print data ()
    {
    cout<<a+ib;
```

friend syntax → friend complex sum complex (complex_O1, complex_O2)

complex → class name; sum complex → function name; complex_O1, complex_O2 → object of class

```
complex O3;
O3 = setNumber(a*O1 + a*O2, b*O1 + b*O2);
return O3;
}
main()
{
clrscr();
class c1, c2 sum;
c1.getdata(1,4);
c1.getdata();

c2.setdata(1,5);
c2.getdata();

sum = Sumcomplex(c1, c2);
sum.getdata();
```

Here friend function does not became the member of the class and neither we can access class objects.

Chapter 27: friend class & Member friend

~~forward declaration~~.

~~class complex;~~] → declaring the class from where we want to access private member.

```
class Calculator
{ public:
    int add(int a, int b)
    { return(a+b);
    }
    int sumRealcomplex(complex, complex);
};
```

declaring this function and will use at last after complex class it will error.

Page No.
Date / 20

```
class complex{
    int a, b;
    friend int Calculator::sumReal complex(complex, complex);
```

making the member of other class friend of this class.

```
public:
    void set number( int n1, int n2)
    { a=n1;
      b=n2; }
    void print number ( )
    { cout<< a+ib;
    };

int Calculator::sumRealComplex(complex O1, complex O2)
    {
      return (O1.a + O2.a);
    }
int main ( )
    complex O1, O2;
    O1.set number (6,7);
    O2.set number (7,9);

    Calculator C1;
    int sum = C1.callsumReal complex (O1, O2)
    cout<< sum;
```

→ forward declaration is necessary.
→ using firstly declaring the member and using the member at last is also imp.

you can also make the full class a friend.

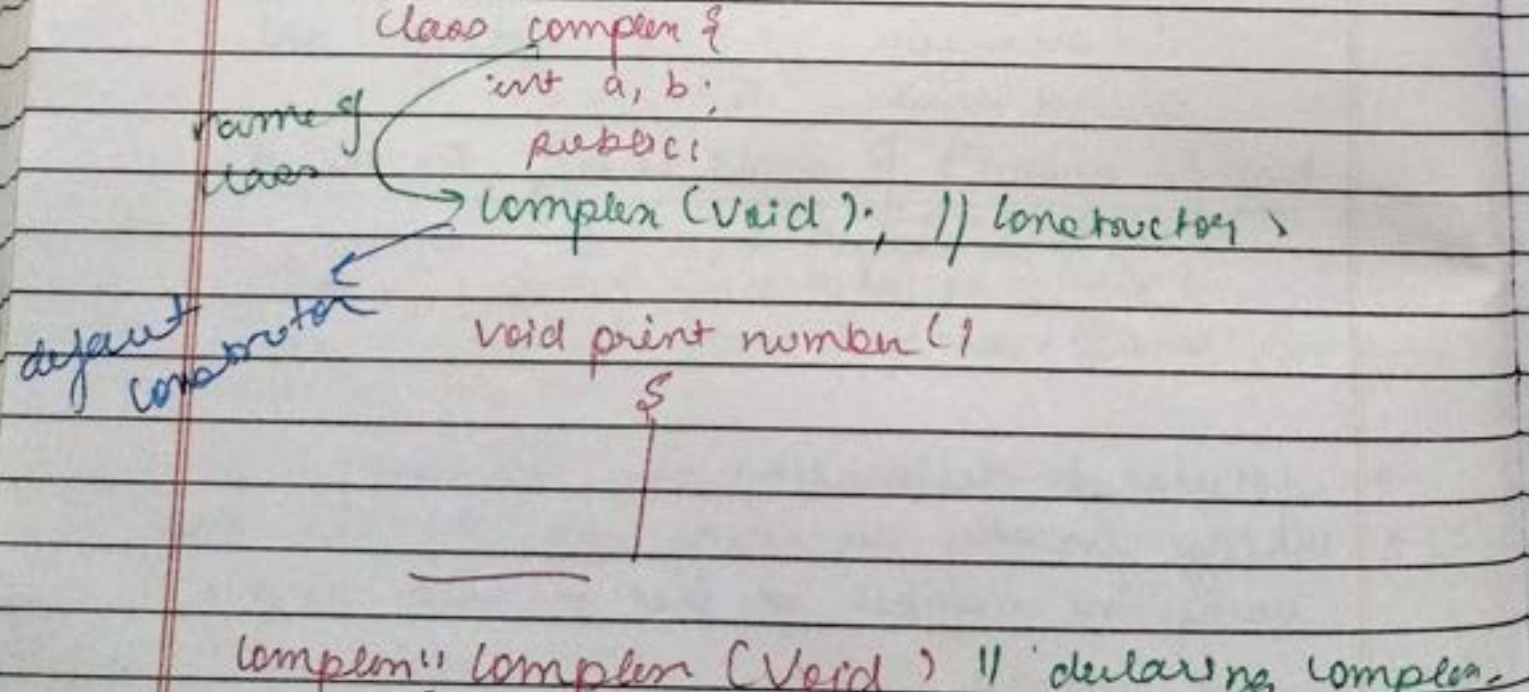
If you make whole class a friend then every member of that class will be able to access the private member of other class.

`friend class calculator;`] inside the class whose private member you want to access.

Chapter 28 More on C++ friend function.

Chapter 29 Constructor.

→ Constructor is a special member function with same name as of the class automatically invoked whenever object is created.



Page No.
Date / 20

```
int main()
{
|
complex c;
c.print number() → output = 10+20i
```

Properties of constructor

- Constructor which do not accept any parmeter are default constructor.
- It should be declare in Public section of the class
- They are automatically invoked whenever object is created.
- They cannot return value and donot have return type.
- It can have default argument.
- We cannot refer to the address.

Chapter 80 Parameterized & Default Constructor

there are two method to give parameter to constructor.

- Implict
- Explicit

These constructor are called parameterized constructor.

Implicit

Explict

```
complex a(4,6);
a.print();

complex b = complex(6,7);
b.print.
```

Chapter 31 Constructor Overloading

we can have more than one constructor in a class with same name, as long as each have a diff list of argument.

```
class complex {
    /
   /
public:
  complex() {                    ⟹ default constructor.
  a=0
  b=0;}
  complex(int x)                 one input
  {a=x;                          ⟹ Parameterized Constructor.
  b=0;}
  complex(int x, int y)          when two
  {a=x                           input are given
  b=y;}
```

Chapter 32 Constructor with default argument

default argument of the constructor are those which are provided in the constructor declaration.

default arrugment are passed in case if you dont provide the value the default argument value will be passed.

Page No.
Date / 20

Simple(int a=5, int b=6)
{
add X=a
Y=b }.

default argument.

If if you give 4 {2 then that value will be print else these 5 and 6 will be print

Chapter 33 Dynamic initialization of object using constructor

It means initialising the object at run time.

class Bank {

Bank( ) { }

other code

main( )
{

Bank bd1, bd2, bd3;

necessary to make this. if in case you only initialise two or one of them else it will throw error.

This is reason because when we make the constructor compiler search for the constructor for the object and if there isn't any empty constructor than it will throw error. We don't need to make any empty constructor if we initialize all object using other constructor.

Chapter 34 Copy constructor in C++.

Copy constructor is a constructor which make the copy of other object.

// when no copy constructor is found compiler its own copy constructor //

```
<class>(<class> &obj){
cout<<" copy constructor called ";
a=obj.a;
}
```

Syntax of creating copy constructor.

number z1(n); → copy const called.

number z2
z2 = n; → copy const. not called

number z3 = n; → copy const. called.

Chapter 35 Destructor in C++

Destructor full up all the memory taken by constructor and object dynamically.

Destructor neither take any argument nether give/return any value.

Syntax to write destructor

```
~<Class name>(){
}
```

this will be execute whenever compiler find out that your have this object of this class will not be used further the destructor will call automatically and will full up memory.

If you write a object inside { } than after } at end the destructor will delete up the object as you have created that object dynamically only for that range.

Chapter 36 Inheritance & its different types
(part of oops)

As we Know:-

- Reusability is imp feature of oops

Page No.
Date / 20

- Reuse classes save time and money.
- Reusing already tested and debugged class will save a lot of effort of developing and debugging the same thing.

Inheritance means when you want the attribute of one class same to be in the attribute of other class without repeating all of them is called inheritance

example :-

Class employee
name
age
address
date of joining

Class manager

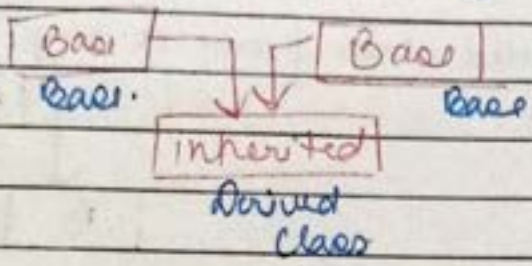
Now manger want same attribute as employee + we can add additional variable & member function -

this will be done using inheritance.

what is inheritance in c++

- 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 called as **Derived Class**
- Reusing classes save time and money.
- Diff type of inheritance in c++.

// we can do inheritance from two base class

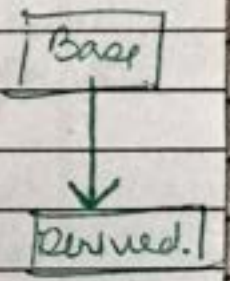


form of inheritance in C++

- Single inheritance.
- multiple inheritance.
- hierarchical inheritance.
- Multilevel inheritance.
- Hybrid inheritance.

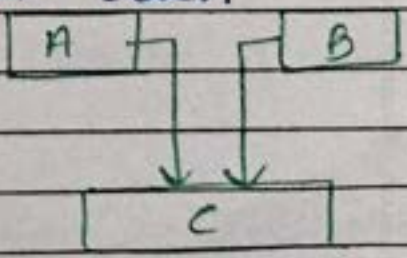
⇒ Single inheritance

a derived class with only 1 base class.



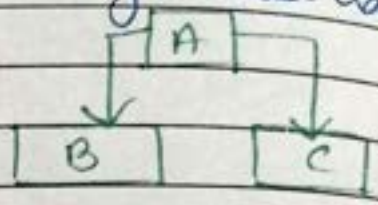
⇒ Multiple inheritance

derived class with more than 1 class.



⇒ Hierarchical inheritance

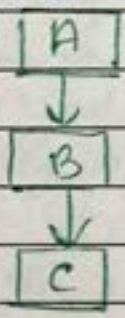
Several derived class from a single base class.



⇒ Multilevel inheritance

Deriving a class from already derived class.

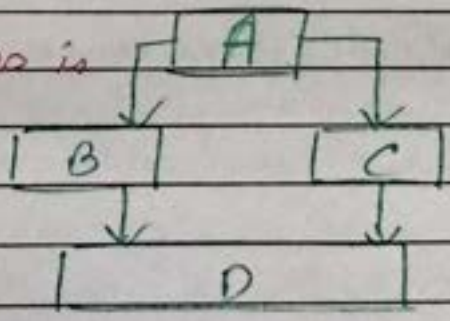
Here C drived from B which is already derived from A.



⇒ Hybrid inheritance

∴ Hybrid inheritance is combination of multiple inheritance and multilevel inheritance.

* one of the parent class is not the base class.



Page No.
Date / 20

chapter 37 Inheritance syntax & visibility -

Syntax

class <derived class name> : visibility - mode
< base class name >
{
member / methods etc.
}

It can be public or private / protected. This is visibility mode of inheritance.

if you inherit public then your base class public member will become public member of your derived class.

if you inherit private then your base class public member will become private member of your derived class.

By default visibility mode is private.

private member never get inherit in any case.

for inheritance there should a base class default constructor.

whenever you will create a Inherited class object base class default constructor will be called.

Page No.
Date / 20

Chapter 8 Single Inheritance Deep Dive:

```
class base {
    data1 ; → private
    public:
    data2;
    void SetData();
    int Getdata1();
    int Getdata2();
};
void base :: setdata(void)
{ Data1 =10;
  Data2 =20;
};
int base :: getdata1(){
  return
  };
```

visibility mode.

```
class Derived : public base{ → public inherit class from base,
    int data3;
    public:
        void process();
        void display();
};

void derived :: process(){
    Data3 = data2 * getdata1()
```

data2 can access directly

need a function :: it is a private variable.

void driver: display {

}

main() {

Driver dr;
dr. setdata();
dr. Process();
dr. display();

}

If you make visibility mode private ~~and still want to do this~~ then you will not be able to do dr. Setdata as by then it would have become private function

and to do that we have to set data ~~class~~ in the function & process by them we are using a private function in a class which is okay and will work fine.

Chapter 39: Protected Access Modifier

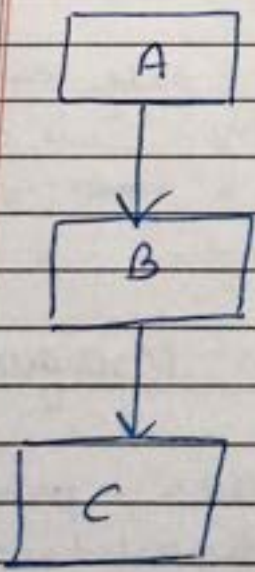
Protected variable are like private but we can inherit protected unlike private.

for a protected member.

Inherit form →	Public deri	Private deri	Protected Deri
Private member	not inherited	not inherited	not inherited
Protected member	Protected	Private	Protected
Public member	Public	Private	Protected

protected variable can also be not use directly in main class.

chapter 40 :- Multilevel Inheritance Dup Dive

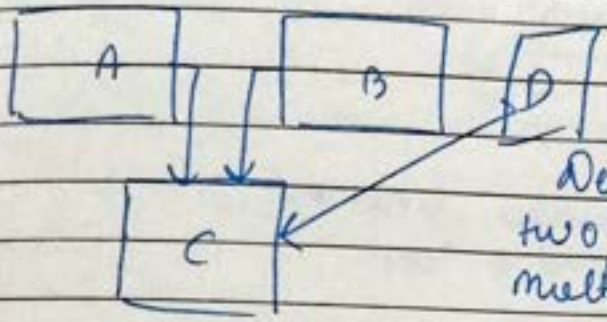


inheriting a class from already inherited class is called multilevel inheritance.

→ Remember this while you do multilevel inheritance.

Page No.
Date / 20

Chapter 41 : Multiple Inheritance with Deep Dive



Deriving a class from two or more Base class is called multiple inheritance.

Class Derived : <visibility-mode> base1, <visibility-mode> base2 {

}

Chapter 42 : Exercise on C++ inheritance

// Code done on laptop of simple, Scientific and hybrid Calculator. //

Chapter 43 : Ambiguity Resolution in inheritance

Ambiguity is defined as when one class is derived from two or more base classes then there are chances that base class have function with same name. This is called ambiguity.

to resolve this issue you have to tell the compiler which function you want to use in that derived class.

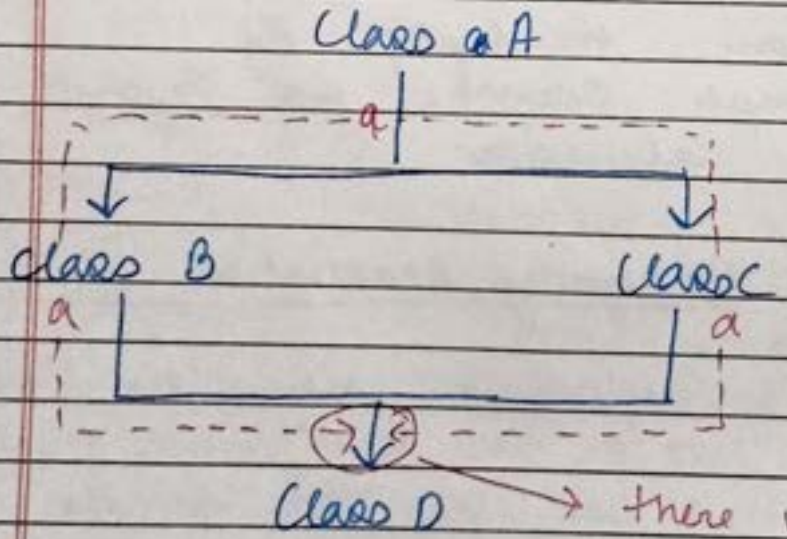
for multiple inheritance

```
class derived: public base1, public base2 {
public:
    void greet() {
        b<class>::greet();
    }
};
```

Class whose function you want to use.

for single inheritance Derived class function overrides the base class function.

Chapter 44 Virtual Base Class in C++



→ there will be two a now and it will cause ambiguity.

to avoid this we make B and C as virtual class.

Page No.
Date / 20

Class B : virtual public A {

};

this shows A is virtual base class.

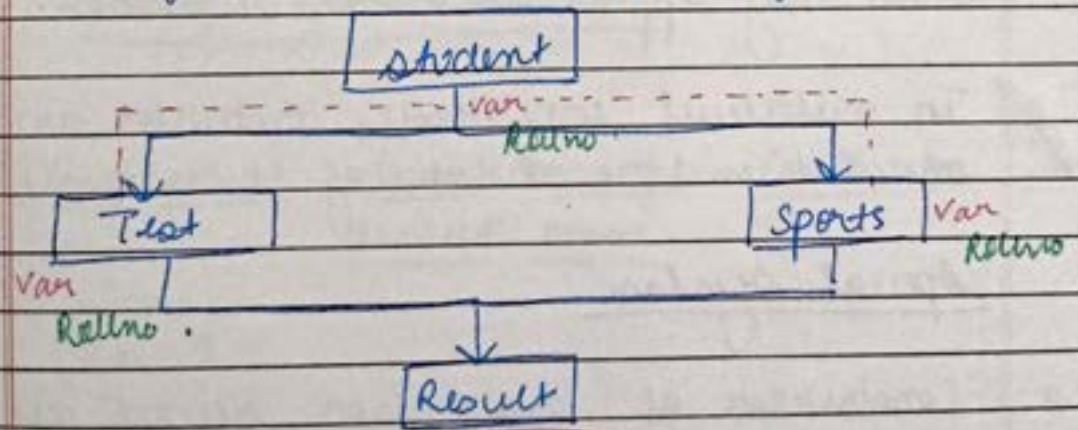
Class C : virtual public A {

};

virtual class will make sure there will be no ambiguity.

Chapter 45 Code example demostration Virtual Base Class

Ok gona do the example of



Class test : virtual class public student {

};

Can do in either of way

Class sport : public virtual student {

};

Page No.
Date / 20

Chapter 46 Constructor in Derived class

- We can use constructor in derived class in C++
- if Base class do not have any argument then there is no need to use constructor
- But if there is one or more argument in base class constructor, then

derived class need to pass argument to the base class constructor

- If both base and derived class have constructor (base) will be executed first

Constructor in multiple / multilevel inheritance

- base class are constructed in the order in which they appear in class declaration.
- In multilevel inheritance, constructor are executed in the order of inheritance.

Special Syntax

- Constructor of base class receive all the argument at one and then will pass the calls to the respective base classes.
- body is called after all constructor are finished executing.

Page No.
Date / 20

Drived - const (arg1, arg2 ...) : Base1 - const (arg1, arg2), Base2 const (arg3, arg4)

{

}; Base1 const (arg1, arg2)

Special Case of virtual Class

- invoked before any non virtual base class invoked in order they declare
- any non-virtual base classes are then constructed before the derived class const. is executed.

Chapter 47 Solution to Exercise on inheritance

Coding of chapter 42

Chapter 48 Code Example Constructor in derived class.

Case :- 1

class B : public A { → order of execution of constructor
};

first A() then B()

Page No.
Date / 20

Case :- 2

Class A: public B, public C {

};

order = B() → C() → A()

∵ because B is written first.

Case :- 3

Class A: public B, virtual public C {

};

order = C() → B() → A()

because virtual class is declare before any other non-virtual class.

Code :-

This matter and all to it Base1 constructor will be called first.

class Derived: public Base1, public Base2 {

int D1, D2;

public:

This order

Derived(int a, int b, int c, int d): Base2(a)

Base1(b) {

D1 = c;

D2 = d;

}

do not matter

Chapter 49 Initigation list in C++

help to initialize variable in a const toucter.

Syntax

```
Constructor (arg-list) : initiligation-sections
{
  assignment + other code;
}
```

example

```
Class Test {
    int a
    int b
      public:
      Test (int i, int j) : a(i), b(j)
      {
      }
};
```

a is declare first

by this a will become equal to (i) and b will equal to (j)

You can also do `b(i+j)`
`b(i-j)`
`b(2*j)`

The variable which is declare first get initialize first as well. which mean

if I try to use b in a then it will give me error value.

a(i+b), b(j) ✗

This give error as b is not initalized and we are using b in a.

b(j), a(i+b) ✗

This also get error as we have define a first and while defining the variable So a is will be initialized first as well and b will not be declare that time cause error

Chapter 50 Revisting Pointer: new and delete keyword

for after chapter 12

New Keyword

basically

```
int a = 4
int * ptr = &a;
```

cout << *(ptr); ⟶ gives 4

and

cout << ptr; ⟶ gives address of a

Page No.
Date / 20

New keyword

Dynamically → int * p = new int (40)
cout << *p → give 40
cout << p → address of p

Creating array dynamically

int * array = new int [3] → create new array with 3 element ie index storage [0,2]

array [0] = 10;
array [1] = 20;
array [2] = 30;

you can also write *(array+1)

cout << "The Value of array[0] is " << array[0];

↓

give output (10)

Delete Operator

↓

use to free dynamically initialized array to free up the memory.

delete <variable>;
to delete variable.

delete [ ] <array>;
to delete complete array list.

Page No.
Date / 20

Chapter 51. Pointers to object and Arrow Operator

```
Complex c1;
Complex *ptr = &c1;
```
→ pointer object created

```
(*ptr).SetData(a, b);
(*ptr).GetData();
```

These () are important as (.) operator have greater precidence than (*) operator.

also `Complex *ptr1 = new Complex;`

This syntax is also valid

Arrow Operator (->)

using this you can directly the d regions function of a pointer.

```
Complex *ptr = &c1;
ptr -> SetData(a, b)
ptr -> GetData()
```
→ arrow operator

They can only be use on pointer.

→ work as (*).

→ pointer variable name.

-> means ptr jis object ko point kr rha hai uska set Data run kro uska Get Data run kro.

* How can we make array of objects.

```
Complex * ptr = new Complex [3]
ptr -> SetData (1,3);
ptr -> GetData ();
```
→ 0 index

```
(ptr+1) -> SetData (4,5);
(ptr+1) -> GetData ();
```
→ 1 index

Can create more like this.

Chapter 52 Array of object using pointer

As we use to do for the int

```
int * ptr = new int [34];
```
→ square Bracket indicate array.

↳ In this compiler give us the memory to store 34 int in an array similarly we do it for object

```
shop * ptr = new shop [2];
```
↳ shop* type object

* Here we use

```
shop * ptr = new shop [3];
shop* ptrTemp = ptr;
```

Through code we make a temp object name ptrTemp. that point to ptr.

item size = 3;

Shop* ptr = new Shape[3];
Shop* ptrtmp = ptr;

int p;
float q;

for(i=0; i<size; i++)
{
Cout<< "Enter ID & price of item "<<i+1<<endl;
Cin>>p>>q;
ptr -> SetData(p,q)
ptr++; → we moved ptr to use all 3 object of array that we create.

for(int i=0 ————)
{ ptrTemp → GetData();
ptrTemp++;

Here we use ptrTemp rather than ptr because through last ptr the value of ptr have been increased upto 4 and if we make a ptrtemp variable.

Chapter 53 this pointer in C++

Class A {
int a;
public:
Void SetData(int a)
a = a; → This throw error because both have same name

and compiler get confuse as which variable did we need to take.

int [a = a;]

priority in c++ is given to local variable first ie [a] = [a]
[a] in arg
[a] object variable.
→ this make no sense Here compiler throw garbage value.

To Solve this issue we use pointer Variable.

this → (a) = (a) → argument Variable
now this a object Variable

this a keyword which is a pointer which points to the object which is called invoked member function.

You can also use this pointer on methods / function.

Class name ← (A) & setData (int a) {
this → a = a;
returns * this }
→ now this return an object type Variable.

returning object type means we can directly use.

a.SetData().GetData();

as this part will return an object

Chapter-54 Polymorphism in C++

Polymorphism => many form of one thing.

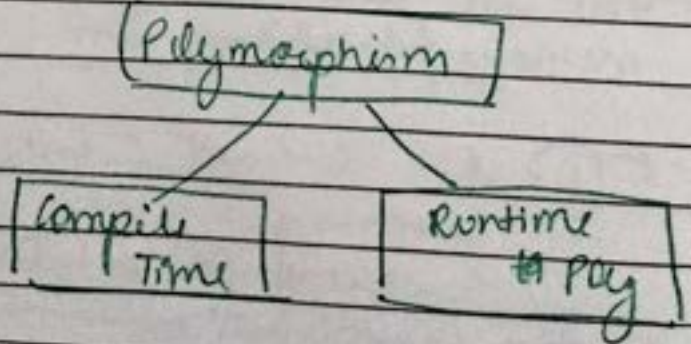
more form

one name multiple forms

example :- function overloading.

Types of polymorphism

→ Compile Time
→ Run Time



Page No.
Date / 20

Compile Time polymorphism :- The decision is taken in the compile Time that which function is going to run.

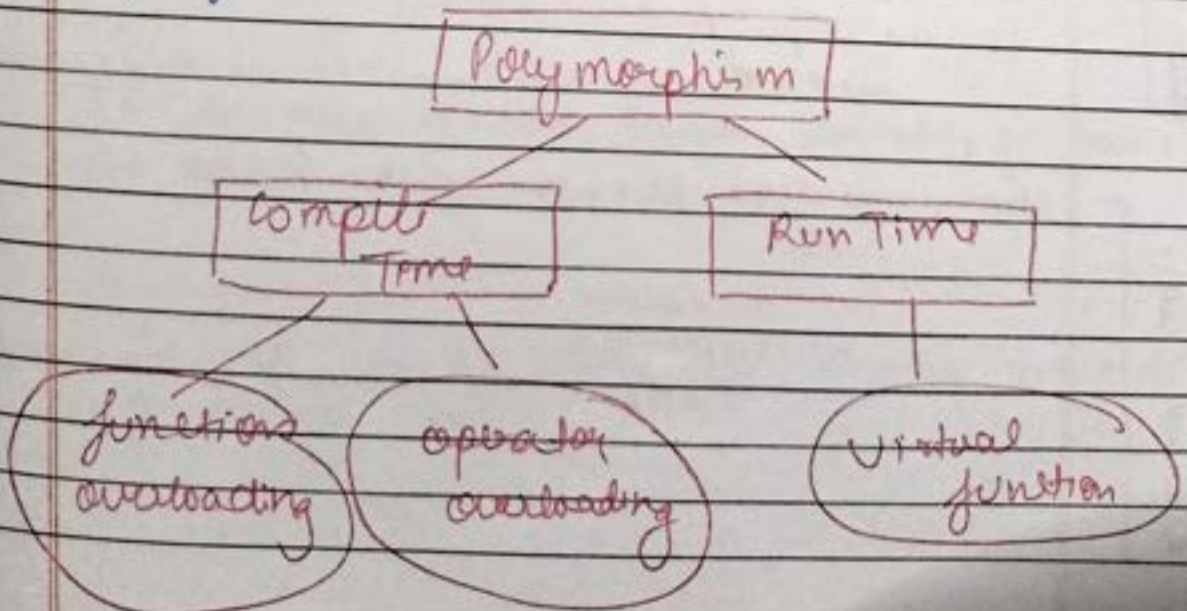
Also known as Early bind and Static Bind

How to achive compile type polymorphism

→ function Overloading.
→ Operator Overloading.

Run Time Polymorphism :- achieve in C++ with help of virtual functions.

In Run Time → compiler diff the decision of function call.



Chapter 55 Pointer to derived classes in C++

In C++ base class pointer can point to the object of derived class and it is legal.

but the function will be called of Base class and not of derived class.

This is known as Late Binding.

Agar apne Base class ke pointer ko derived class se bhi point karvadiya tab toh usse bad agar function run hoge toh woh base class ka run hoge.

Base Class

```
class BaseClass {
    Public:
    int var_Base;
    cout<<"Display Base class variable var_Base"<<var_Base
    void Display() {
    cout<<"Display Var_Base "<<var_Base<<endl;
    }
};
```

Derived Class.

```
class DerivedClass : public BaseClass {
    Public:
    int var_Derived;
    void Display() {
    cout<<"Display Var_Derived"<< var_Derived;
    }
};
```

Base Class * Base_Class_pointer ;→ Base class pointer
Base Class Obj-base1; → Base class obj
Derived Class obj-derived; → Derive class obj

Base_Class_pointer = &obj_derived ;
↳ Base class pointer pointing to the object of derived class.

Base_Class_pointer → Var_base = 34; ✓
you can access value of base class through pointer pointing to derived class

Base_Class_pointer → Var_derived = 10; ✗
but you can't access to the value of derived class variable through that pointer

Base_Class_pointer → Display ( );
The function of the Base class will be executed and not of derived class.

Derived Class * Derived_class_pointer; → Derived class pointer.
Derived_Class_pointer = & obj_derived ;
derived_class_pointer → Var_base = 10 ;
derived_class_pointer → Var_derived = 20;

You can access to both variable of base and derived class using derived class pointer.

derived_class_pointer → Display ( );
It will display value of Display function of derived class.

Page No.
Date / 20

Chapter 56 :- Virtual function in C++

```
Class BaseClass {
public:
    int Var_Base = 1;
    virtual void display() {
    cout<< " Of :- ---------- "<<
    }
};

Class Derived {
public:
    int Var_Derived = 2;
    void display() {
    cout<<" ---------- ";
    }
};

main
BaseClass * Base_Class_pointer;

BaseClass   obj_base;
Derived     obj_derived;

Base_Class_pointer = & obj_derived;

Base_Class_pointer -> display();
```

→ Since we are using virtual function in base class which mean it will only initiate with obj of base class and not of derived class.

So in case the value of derived class will be executed here.

Chapter 57 :- Virtual function example + Creation Rule in C++

* Rules for virtual function

1. They cannot be static
2. They are accessed by object pointer only.
3. Virtual function can be friend of other base
4. A function in base class can might not be used.
5. If Virtual func. define in Base Class then is no necessary of redifing it in derived classes.

```
class CWH { → Base class.
    Protected:
    String Title;
    float Rating;
    Public:
        CWH (String T, float R) { → Base class constructor.
            Title = T
            Rating = R
        }
        virtual void display () { }.
};
```

using virtual function.

```
class derived: Public CWH { → Derived class.
    Protected:
    float videoLength;
```

Page No.
Date / 20

Public:
CWH Video (string s, float r, float VL) {
Title = s;
Rating = R;
VideoLength = VL; }

Void display ( ) { → If we point obj to this class but it doesn't have that function then base class virtual function get called.

}

main ()

title = ---- ;
Rating = ---- ;
VideoLength = ---- ; } using this over here
CWH Video = obj-video (title, Rating, VideoLength);

CWH * tuts[2]; → making a array of object.

tuts[0] = & obj-video; → pointing obj to obj of CWH video

tuts[0] -> display()

Now if there were two derived class and there was a function display() in both of them then this will error if we don't use virtual in base class.

Page No.
Date / 20

Chapter 58 Abstract Base class & pure virtual function.

Syntax for pure virtual function.

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

→ This is also known as do nothing function

→ This make sure that you cannot a derived class now which donot contain display function into it.

Abstract Base Class

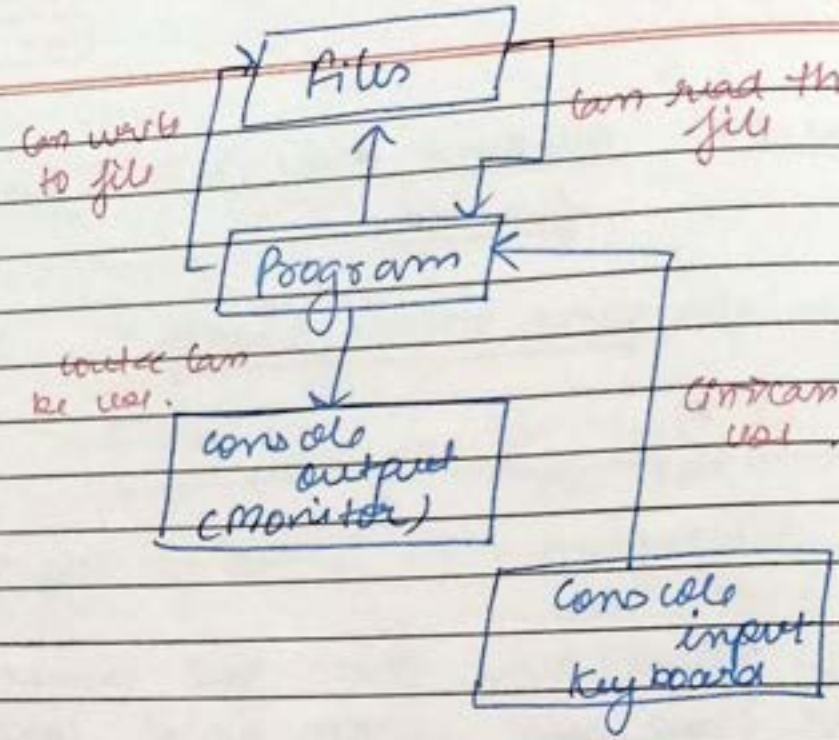
Abstract Class is a class which contains one or more abstract method / pure virtual function.

→ They seperate the interface from the implementation

Both Abstract Base Class and Pure Virtual function are related to each other.

Chapter 59 files I/O in C++

files => It is a pattern of data that is stored in disk



Chapter 60 Files I/O in C++ Reading and writing

we will use `#include <fstream>`

- The useful classes to working with files in C++ are :-
  1. fstreambase
  2. ifstream
  3. ofstream

) → These both are derived from fstreambase.

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

- using the constructor
- using member function open() of the class.

Page No.
Date / 20

→ opening file using constructor and writing it

```
# include <fstream>

main()

string st = "abc"; ] → text we want to write.

ofstream out("<file name>");
```

out → creating object of ofstream file

("<file name>") → file name where we want to write

out << st; → this will add our string to the files.

→ opening file using constructor and reading it

```
# in -- <fstream>.

main()

ifstream in("<file name>");
```

in → creating object of istream.

("<file name>") → file name that we want to read.

```
string str;
```

str → creating empty string

```
getline(in, str);
```

in → object, str → string.

Page No.
Date / 20

Get line function is use to get a complete line we don't use get line function than only character before first space and rest line will be print.

cout << str; → casting the str in which we store data now.

Chapter 61 File I/O in C++ Read/Write in same program & closing file

Syntax of close function.

<object>.close()

It is important to use the close stream in the object as it will allow the compiler to free up the memory. that is being occupied by the object in the memory.

Chapter 62 file I/O in C++ open() and eof() function

opening the file using open()

ofstream out;
out.open("<file name>");
→ open method is already defined in ofstream.

out << "_____";
out << "_____";

out.close(); → closing an object is also necessary.

Use of eof() method in the code.

```
ifstream in;
in.open("<file name>")

string str;
while (in.eof()==0){
    getline(in, str); → using getline to get a line
    cout << str << endl;
}
in.close()
```

The above syntax is used to print all the text till the last line.

Chapter 63 C++ Templates: Must for competitive

We know that

'class' → object

object are created with the help of class.

Page No.
Date / 20

Template → class

class are made with the help of template.

❖ why we need to use template :-

✓ Template are also known as parameterized classes.

→ we use template to follow our (DRY) principal (Do not Repeat Yourself)

→ we can do (generic) programming with the help of the template.

Syntax of template

```
template <class T> → T can be int, float, char &
class Vector {
    T* arr;
    Public:
        Vector(T* arr)
        { ——————
          ——————
        }
        ——————
        ——— many other
        —— method ..
};
|
main().
```

Vector <int> myVec(5);
Vector <float> myFVec(ptr1);

Chapter 64 Writing our first C++ template

#include
<
<

template <class T> → initialising Template
class vector {
public:
T* arr; → using Template as our array can be float or int
int size;

vector(int n) {
size = n;
arr = new T[size]; mean it could be new int[size] new float[6] →
}

(T) DotProd(vector &v) { → T int, float, double type function.
T d = 0; → int d = 0, float d = 0
for(int i = 0; i < size; i++)
{
d += this→arr[i] * v.arr[i];
}
array of this object
array of other object.
return d;
}

```
int main() {
    |
    |
    T
Vector <float> V1(3);   → data type of template
V1.arr[0] = 4.9;
V1.arr[1] = 5.1;
V1.arr[2] = 1.2;

Vector <float> V2(3);
```

float d = V1.dotproduct(V2)
↳ for storing float number.

```
cout<<d<<endl;
```

Chapter 65 Template with multiple parameter

Syntax for multiple parameter

```
template <class T1, class T2>
class <name> {
    // body
};
```

T1 → can be used as float
T2 → can be an int or

Very same class.

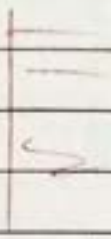
Code on laptop

Page No.
Date / 20

Chapter 66 Template with default parameter :-

using default parameter in C++.

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



```
int main(){
```

<Class name> < > → that can be left alone as we have already defined our default parameter.

If there is an exception case where we don't want to use default template we can also do that.

```
int main () {
```

<Class name > < float , char > object (values).

defining the new template values.

Chapter 67 function Template & function template with parameter.

other method use for cout the output in C++.

printf(" ——— %f ", a) → valid.

%f → for a = float
%d → for a = int
%c → for a = char.

function template syntax:

```
template < class t1, class t2 >
float Average (t1 a, t2 b)
{
    float avg = (a+b)/2;
    return avg
}
```
→ function.

we can use the template in function the same way we use template in the class.

template in main function

< function name > < template > ();

Chapter 68 Member function template & overloading

Defing the function template outside the class.

```
template < class T>
void <class name> :: <T> :: <function> { }
```

we can also do function overloading with template.

but always the exact match function will get execute.

Important

Chapter 69 The C++ Standard template Library (STL)

STL :- Standard template (Library)

↓

Library of generic classes & function.

Why use STL?

→ we use well tested component (classes & function).
→ Time saving.

STL Component

- Containers → [(store data) * use template classes
- Algorithm → [Sort, Searching, merging etc.
- Iterators → object point to an element handle just like pointer.

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

Page No.
Date / 20

Chapter 70
Container in C++ STL

Container are object which store data

Container are of three types

- Sequence Container (Linear fashion)
- Associative Container (direct access)
- Derive Container. (Real world modeling)

⇒ **Sequence Container**

They are (Linear fashioned)
example :- vector
list
Deque (double queue).

⇒ **Associative Container.**

They are (direct access)
example :- set,
map
multimap

use for fast searching of an element in.

⇒ **Derived Container**

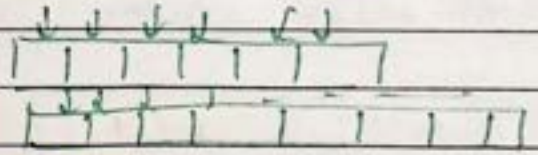
can be derived from sequence container or Associative Container.

example:- stack → LIFO (last in)
queue → FIFO (first in)
priority queue.

Sequence Container

* Vector → Random Access → fast.
Insertion / Deletion → slow.
Insertion at end → fast

Room



Resize

* List → Random Access → slow.
Insertion in Middle → fast
Insertion at end → fast.

Associative container

All operation → fast
except R.A

Derived container

depends → data structure

Page No.
Date / 20

Chapter 71 Vector in C++ STL

we need to add

```
#include <vector>
```

You can access vector method by google at

C++ .com documentation

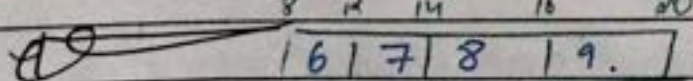
Syntax to make vector object

```
vector <data type> object ;
```

many method are already pre-defined and we just need to use them properly.

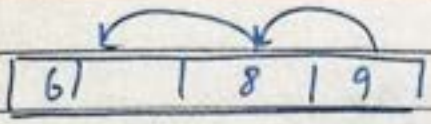
Chapter 72 list in C++ STL

It is bidirectional linear list that shows insertion and deletion operation.

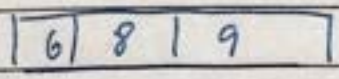


In default array store element in contigious block of memory.

Now if we want to remove (7)



then we have to shift both 8 and 9 in a array store element in contigious block of memory.

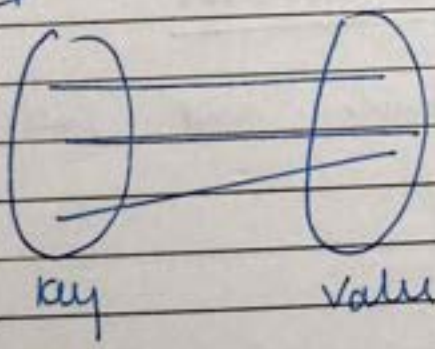


this is a very difficult task so we use list

<Code> and gogle documentation.

Chapter 73 Map in C++ STL

Map is use to store key Value pair.



Syntax to initialize map

```
map<string, int> mapmark;
mapmark["A"] = 60
```

for its output we will use iterator.

using iterater for map

```
map<string, int>::iterator itr;
for( itr = mapmark.begin(); itr != mapmark.end();
                                    itr++ ).
{
cout << (* itr ).first << (* itr ).second;
```

to print first element at that itr ie. [name]

to print second element at that itr ie [marks.]

chapter 74 function objet in STL

function objet → functor

more form reference and code.