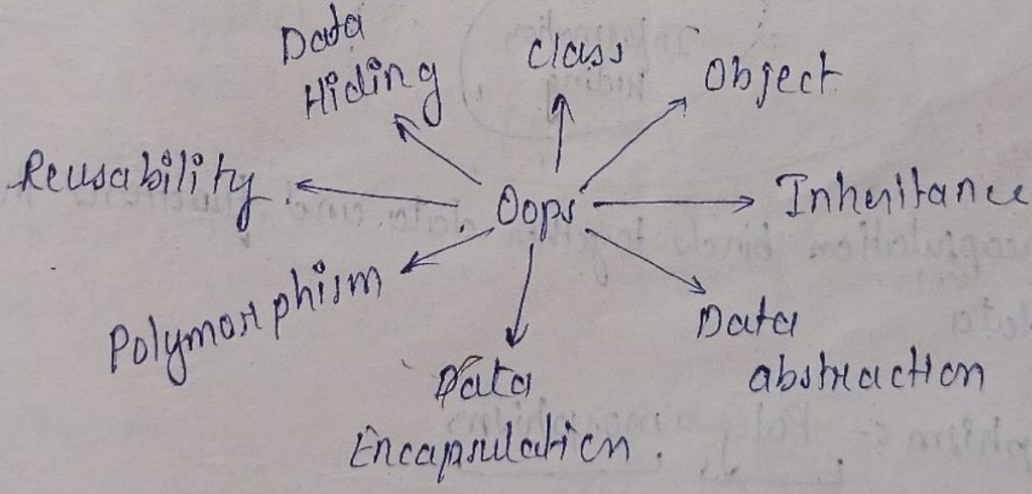
POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

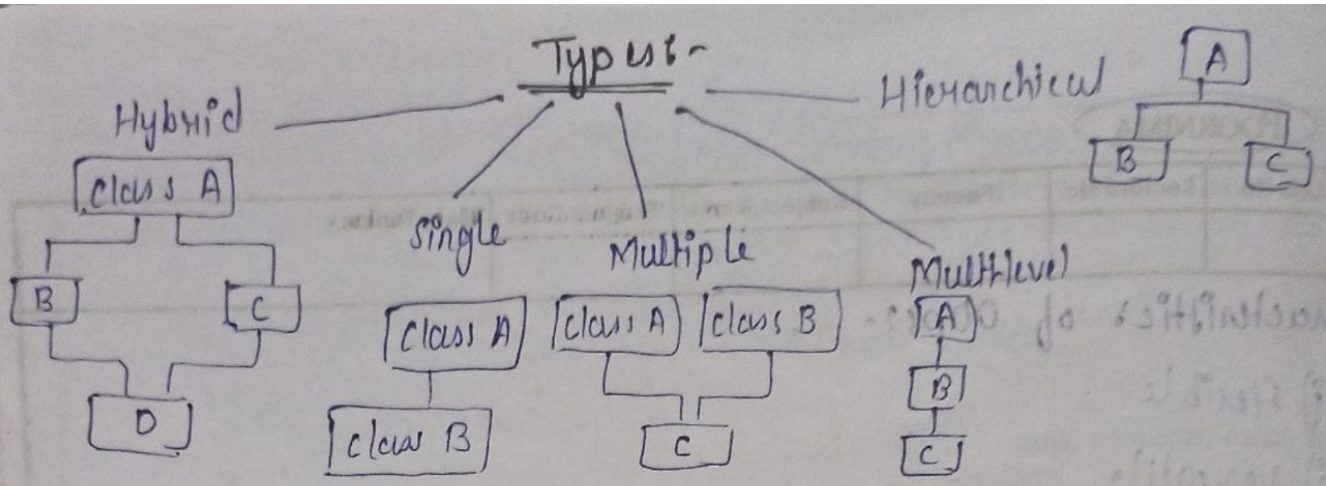
- Characteristics of OOPs:-
  - i) Flexible
  - ii) Versatile
  - iii) Simple
  - iv) Easy to learn
  - v) Extensible
  - vi) Secure
  - vii) Open source



- Class:- Group of objects with same attribute & behaviour
- Object:- It is instance of class.
- Inheritance:- It is the ability of by which a class acquire the properties of another class.

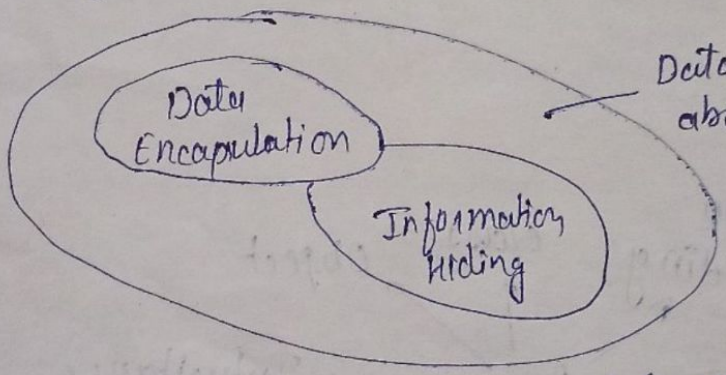
Main Ideas, Questions & Summary:

Library / Website Ref.:-



- **Data Abstraction:-** It is also called information hiding.

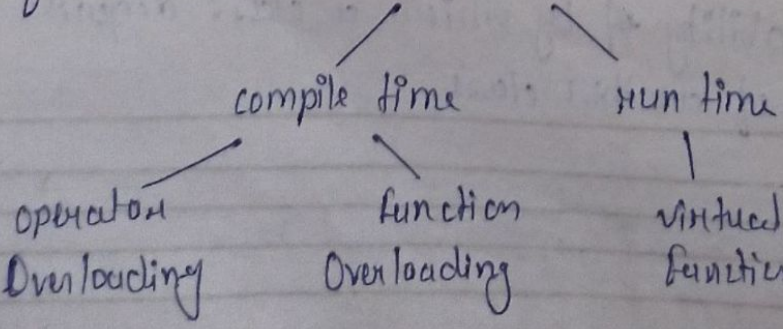
Provide only essential information to others and hiding the background details.



Data Encapsulation binds together data and function that manipulate data

Polymorphism :- Poly (many) morphism (form)

ability of object of different types to respond to function of same name.



POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

The variables which are declared in any class by using ~~array~~ any fundamental data types (like int, char, float, etc) or derived type (like class, structure, pointer etc) are data member

functions which are declared either in private section or public section are member function.

Types.

Private Member
(Member declared in private section of class)

Public Member.
(Member declared in public section of class).

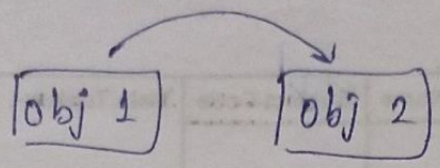
- Structure
  Public data member
  used by small amount of data structure
  Parameterized constructor

Class
Private data Member.
used by large amount of data
Constructor & Destructor.

Main Ideas, Questions & Summary:

Library / Website Ref.:-

Message Passing



object can communicate with each other by sending the function.

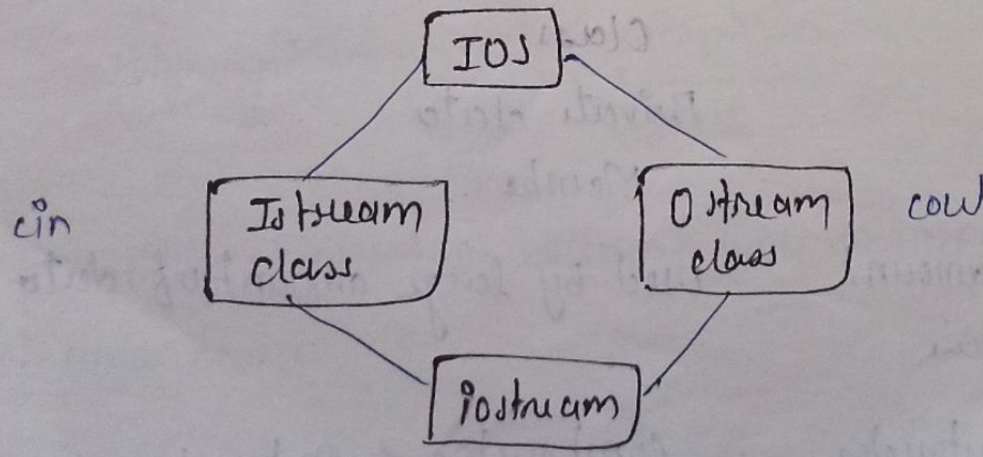
* To control I/P and O/P operation in c++ we need predefined objects which are controlled by streams.

There are 2 streams.

I/P O/P

Streams are flow of characters or flow of bytes which are having a ~~src~~ source or distination.

Streams have some predefined classes and objects to control the standard I/P and O/P operations.



• Programming Structure in C++

1) Documentation Session :-

Comments

//

/* ——
——
—— } comment block or multi line

2) Header files :-

#include <iostream.h>

#include <conio.h>

3) Global Variable

4) Class Declaration

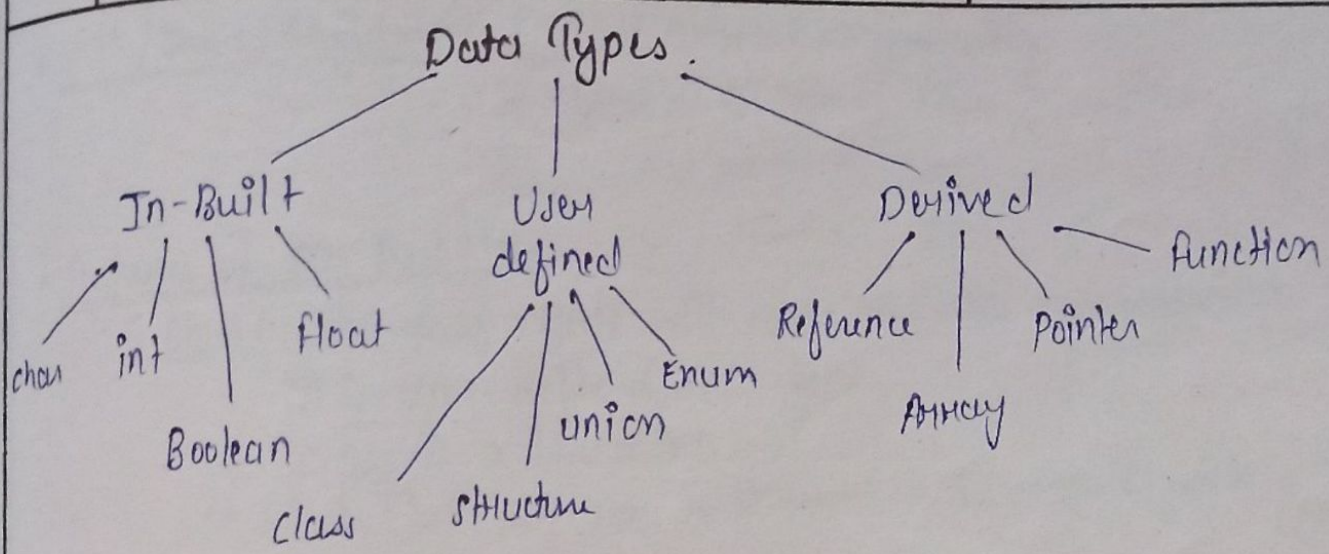
5) Function Declaration

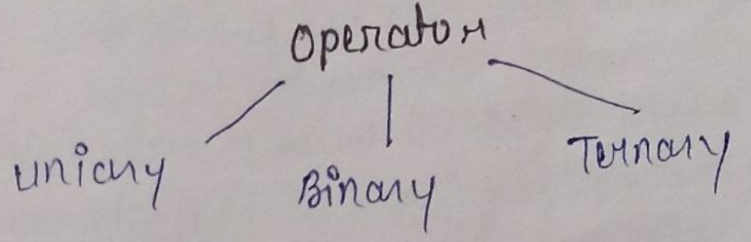
6) Main ()

7) Other # Statements.

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-





- New :- used for dynamic memory allocation
- Delete :- used for deletion / deallocation memory.
- endl :- ~~Else~~ used for new line
- ends :- Ending space

:: :- Scope resolution / Relationship operator / Scope operator / Global access operator.

*. :- Pointer to member (access operator)

* :: c

Main Ideas, Questions & Summary:

Library

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

- C++ / Java / Angular :- i) Object Oriented programming.
  ii) Object or module
  iii) Modular programming.

- Features of OOP in c++:-

i) Class :- Group of objects with same attributes and behaviour.
It is user-defined data type.
Class is a blueprint.
* Binds data member and member function together.

ii) Object :- Collection of methods and functions.
Variable of type class. It is instance of class.
Physical representation of class.
Object allow to access the data member & data function.

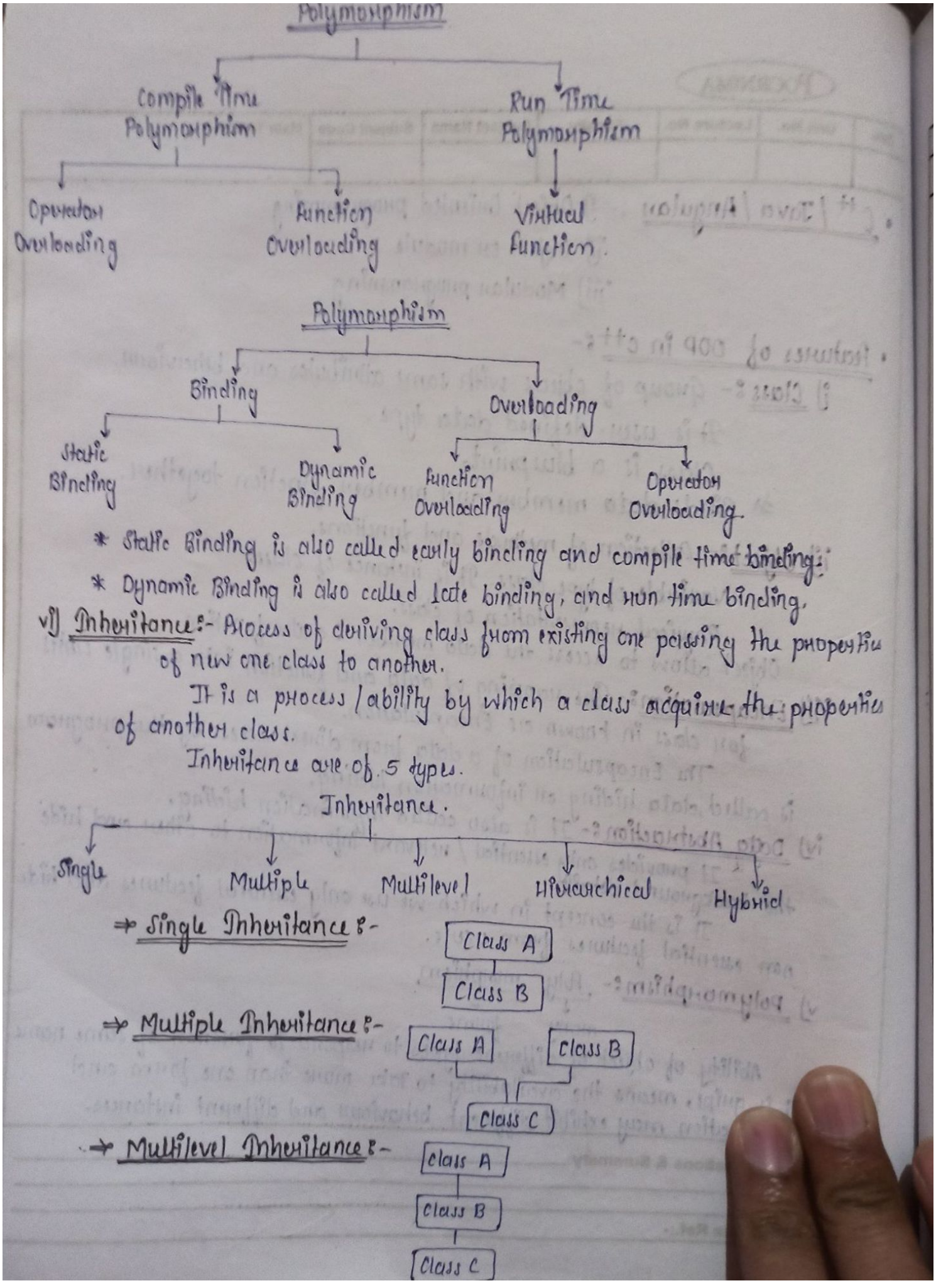
iii) Encapsulation :- The mapping of data and function into a single units for class in known as Encapsulation.
The Encapsulation of a data from direct access by the program is called data hiding or information hiding.

iv) Data Abstraction :- It is also called information hiding.
It provides only essential / relevant information to other and hide the background details.
It is the concept in which we use only essential features and hide non essential features from users.

v) Polymorphism :- Poly (many) morphism (forms)
Ability of object of different types to respond to function of same name It is gripe, means the availability to take more than one form and operation may exhibit different behaviour and different instances.

Main Ideas, Questions & Summary:

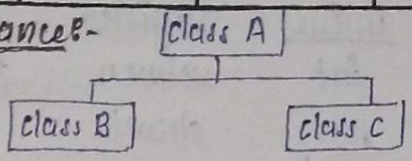
Library / Website Ref.:-


Polymorphism
Compile Time Polymorphism
Run Time Polymorphism
Operator Overloading
Function Overloading
Virtual function.
Polymorphism
Binding
Overloading
Static Binding
Dynamic Binding
Function Overloading
Operator Overloading.
* Static Binding is also called early binding and compile time binding.
* Dynamic Binding is also called late binding, and run time binding.
vi) Inheritance:- Process of deriving class from existing one passing the properties of new one class to another.
It is a process / ability by which a class acquire the properties of another class.
Inheritance are of 5 types.
Inheritance.
Single
Multiple
Multilevel
Hierarchical
Hybrid
⇒ Single Inheritance:-
Class A
Class B
⇒ Multiple Inheritance:-
Class A
Class B
Class C
⇒ Multilevel Inheritance:-
Class A
Class B
Class C

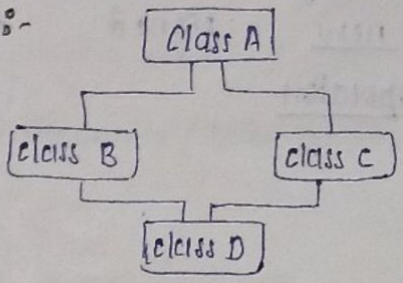
POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

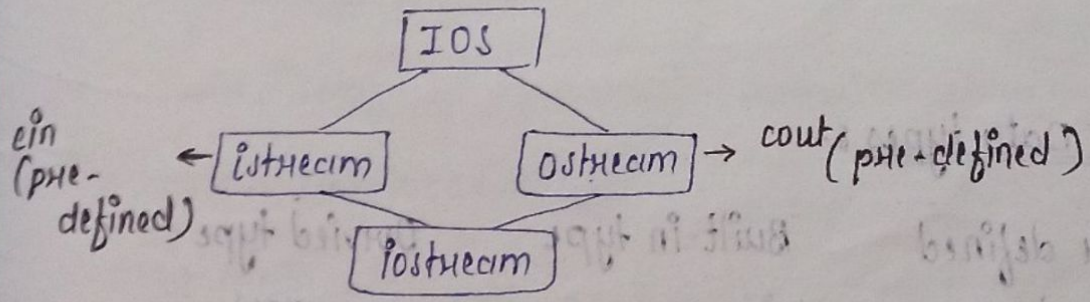
→ Hierarchical Inheritance:-



→ Hybird Inheritance:-



viii] Message Passing:- Object are able to communicate each other by sending the functions.



- C++ Programming:- It is object oriented programming language. It was developed by Bjarne Stroustrup in USA in 1980's (Bell Lab AT&T)

* Token in C++ Programming:- The smallest individual units in a program are known as tokens C++ as the following tokens are:

1] keywords
2] identifiers
3] Constant
4] String
5] Operators.

Main Ideas, Questions & Summary:

Library / Website Ref.:-

* keywords :-

auto	Default	friend	Private	sizeof
break	delete	goto	Protected	static
case	do	if	Public	struct
catch	double	inline	register	switch
char	else	int	return	template
class	enum	long	short	this
constant	extern	new	signed	throw
continue	Float	operator		try
typedef	while			
union	for			
unsigned				
virtual				
void				
volatile				

- **C++ Data types :-**

user defined	Built-in type	Derived type
Structure	void	array
class	integer → int, char	Function
union	Floating → Float, double	Pointer
enumeration		Reference

- **C++ Operators :-**
  - << → Insertion Operators
  - >> → Extraction Operators
  - :: → Scope Resolution operators
  - :: → Pointer to member declaration / declaration
  - →* → Pointer to member operator
  - .* → Pointer to member operator
  - delete → Memory deallocation
  - new → Memory location

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

Program :-

```
#include <iostream.h>
using namespace std;
class Box
{
int length, width;
```

length, width → Local Parameter / Data member.

Data member

```
Public:
void assign (int x, int y)
```

int x, int y → Formal Parameter

```
{
length = x;
width = y;
}
```

length = x; width = y; } Functions

```
void display ()
{
cout << length << width;
}
int main ()
```

int main () → static Local Parameter

```
Box B1
B1.assign (10,20)
```

10, 20 → Local Parameter.

```
B1.Display ();
return (0);
}
```

- **Access Specifiers :-**

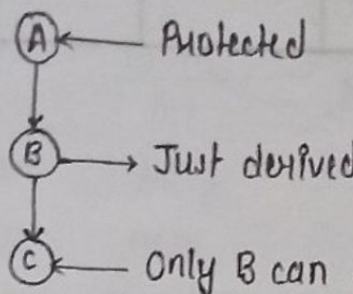
1) **Public :-** In this Access specifier inside class function, outside class function and anyone can ~~excess~~ access which is ~~ex~~ accessed in the real world the public members.

Main Ideas, Questions & Summary:

Library / Website Ref.:-

ii) Private:- In this Access Specifier only inside class function can access private member.

iii) Protected:- In this specifier only just derived class can access protected member.



(Multilevel Inheritance)

- Array of Object:- Suppose we have 50 students in a class and we have to input the name and marks of all the 50 students then creating 50 different objects and inputing the name and marks of all those 50 students is not a good option. In that case, we will create an array object.

Program:-

```
#include <iostream>
#include <string>
using namespace std;
class student
{
 string name;
 int marks;
Public
 void getname()
 {
  getline(cin, name);
 }
 void get marks()
 {
cin >> marks;
 }
void display info()
{
cout << "Name" << name << endl;
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
cout<<"marks "<<marks<<endl;
}
};
int main()
{
student st[3];
for(int i=0; i<3; i++)
{
cout<<"student "<<i+1<<endl;
cout<<"Enter Name"<<endl;
st[i].getname();
cout<<"Enter Marks"<<endl;
st[i].getmarks();
}
for(int i=0; i<3; i++)
{
cout<<"student "<<i+1<<endl;
}
st[i].display info();
return 0;
}
```

Main Ideas, Questions & Summary:

- **Difference between C and C++**

C	C++
i) Header files stdio.h	i) iostream.h
ii) Procedure Oriented	ii) Hybrid language / object oriented
iii) Char structure	iii) Class
iv) C based on process	iv) C++ is based on data
v) Memory allocation and deallocation using malloc, calloc	v) Memory allocation new using and deallocation using delete
vi) Printf and Scanf for input and output.	vi) C in for input and cout for output.
vii) we do not use insertion	vii) We have insertion and extraction operators.
viii) We use formal specifiers	viii) We do not use.
ix) We don't use namespace	ix) We use namespace.
x) No information hiding	x) We use the concept information hiding.
xi) Superset of B	xi) subset of C
xii) Variables are required.	xii) Declaration are required.
xiii) Complication are by gcc/Tc.	xiii) g++ /Tc++
xiv) Variable Local	xiv) By default members are private
xv) It supports in built-in and primitive data type.	xv) It supports built in and user-defined data types.
xvi) We can't use inheritance and structure to other.	xvi) We use inheritance properly for class.
xvii) We can't use constructor & destructor.	xvii) We can use constructor & destructor.

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

- Difference between Structure & Class :-

Structure	Class
i) We use struct keyword.	i) We use Class.
ii) We don't use function inside the structure	ii) We use function inside & outside class.
iii) We don't use the concept information hiding.	iii) We use the concept information hiding.
iv) don't use inheritance.	iv) One class inheritance another class.
v) We don't use protected ~~public~~ private access specifier.	v) Public, Private and protected access specifier.
vi) We don't use reference variable	vi) We use references variable.
vii) Structure is a collection of data member.	vii) Class is collection of data member and member function.

- Programming Structure in c++ :-
  - i) Header file
  - ii) Global Variable
  - iii) Class declaration
  - iv) Class definition
  - v) Function declaration - Optional
  - vi) main

Main Ideas, Questions & Summary:

Library / Website Ref.:-

- **Reference:-** Both reference and pointer can be used to ~~cree~~ check local variable of the function to inside another function. Both of them can also be used to saved copying of big objects when passed as arguments to function or return from function to gain efficiency

⇒ **Reference Variable in c++:-** Reference is also derived data type.

→ reference

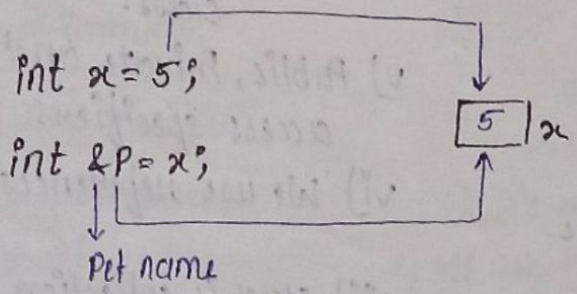
&a

↳ reference variable

For example :- int x = 5

x → at the position

5 → memory



P is a reference variable of x.

Pointer is less secure than reference variable.

⇒ **Rules of reference Variable:-**

i) **Using for references:-** reference variable should be initialized.

i) int &P X

ii) int &P=x ✓

ii) It is a fixed connection and you can't reassign the reference variable unlike pointers.

iii) Reference variable cannot be assigned null

int &P= Null X

iv) Cannot create array of reference unlike pointers.

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

i) int x=y;
int &P;
&P=x ✗

ii) int x=y;
int &P=x; ✓

v) Reference is a new data type in c++ and it is a derived data type.

vi) Reference variable are useful in Copy Constructor & Operator Overloading concept

viii) There are also called the react only pointer.

Syntax:-

data type & variable = variable;

Example:- void main()
{
int a=100;
int &b=a;
int &c=b;
clrscr()
cout<<a<<b<<c; {o/p 100, 100, 100}
c=200;
cout<<a<<b<<c; {=200, 200, 200}
getch();

→ Application of References:- When a variable is declared as reference it becomes a automobile name for an existing variable. A variable can be declared as references by putting '&' in the declared.

Main Ideas, Questions & Summary:

Library / Website Ref.:-

Example:-

```
int main
{
 int x=10;
 int &ref = x;
  ref =20;
cout << "x= " << x << endl;
  x= 30;
}
cout << "ref = " << ref << endl;
```

→ Application of references:-

i) Modify the passed Parameters in the function:-

```
# void swap (int &first int &section)
                          second.
 {
  int temp = first;
    first= second;
    second = temp;
 }
 int main ()
 {
  int a=1; b=3
 swap (a, b);
cout << a << " " << b;
 return 0;
 }
```

ii) Avoiding copy of large structure:- Imagine a function that has to receive a large object if we pass it without references, a new copy of it is created with passes a wastes of CPU time and memory. We can use references to avoid these.

Program:- struct student

```
{
    string name;
    string address,
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
    int roll no;
  }
  void print (const student &s)
  {
cout << s.name << " " << s.address << s.roll no;
  }
```

iii) In for each loops to modify all objects :- We can use reference in for each loop to modify all objects / elements.

```
  for (int &x : vect)
    {
      x = x+5;
  // Printing Elements.
    }
  for (int x : vect)
    {
  cout << x;
    }
```

- Dynamic Memory Allocation :- C programming uses Malloc & Alloc function to allocate memory dynamically and run time. Similarly it uses the function free dynamic Memory Allocation.

C++ Programming also defines to unary operators new and Delete that perform the task of allocating and deallocating of memory in a better and easier way.

An object can be created by using new operator and destroy by using delete operator as and when required.

A data object created inside a block with new will remain in existance until it is explicity destroy by using delete. Thus, the life time

Main Ideas, Questions & Summary:

Library / Website Ref.:-

of an object is directly under our control and unrelated to the block structure of program.

- P = new int;
- q = new float;
- int * P = new int;
- int *q = new float;
- int * p = new int (25);
- int *q = new float (7.5);
- int * p = new int [10];

↙ create a memory space for an array of 10 integer.

Ptr = (node *) malloc (sizeof P, Free [P] node)).

When a data object is no longer needed it is destroyed to release the memory space for reuse.

Syntax:- delete P;

delete q;

↙ delete [] P;

It will delete the interior array pointed by

- **Function Overloading in c++** :- It is a process which are polymorphism. We can have multiple defination for the same function and in the same scope.

The definition of the function must defer from each other by the types &/or the no. of arguements in the argument list.

We cannot overload function declaration that defer only by return type.

Example:-
```
#include <iostream>
using namespace std;
class print data
{
Public
void print (int i)
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
{
cout<<"Printing int"<<i<<endl;
}
void print(double f)
{
cout<<"Printing float"<<f<<endl;
}
void print(char c);
cout<<"Printing character"<<c<<endl;
}
};
int main()
{
Print(data Pd);
Pd.Print(5);
Pd.Print(500);
Pd.Print('A');
return 0;
}.
"Printing character"<<c<<endl;
```

- **Default values for parameters in functions:-** When we define a function, we can specify a default for each of last parameters this of value will be used if in the corresponding arguments is lack blank when calling to the function.

This is done by using the assignment operators and assigning values for the arguments in the function definition if the value for that parameter is not passed when the function is called, the

Main Ideas, Questions & Summary:

Library / Website Ref.:-

default given value is used, But the value is specified, this default value is specified, this default value is ignored and passed value is used instead.

Program :-
```
int sum (int a, int b = 20)
  {
  int result;
  result = a+b;
  return (result);
  }
int main ()
  {
  int a = 100;
  int b = 200;
  int result;
  result = sum (a, b);
cout << "Total value is :" << result;
 result = sum (a);
cout << "Total value is :" << result << endl;
  return 0;
  }
```

- **Passing class parameter (object) into function :-**

```
#include <iostream.h>
using namespace std;
class sample
  {
  Private:
  int a;
  Public:
    void get a ()
  {
  cout << "Enter value of a:";
  cin >> a;
  }
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
void put a ()
{
cout << "value of a is : " << a;
}
};
void main ()
{
sample s1 . s2
s1 . get a ();
s2 . get a ();
s1 . big (s2);
}
void big (sample s2)
{
if (a > s2.a)
{
cout << "s1. a is big";
}else if (a < s2.a)
{
cout << "s2 is a big s2.a is big";
}
else {
cout << "both are equal";
}
};
```

Summary:

- **Member function with class type return value:-**

* If we want to return all the data members of a class at a time then we use member function with class type return.

```
#include <iostream.h>
using namespace std;
class test
{
 Private:
   int a, b;
 void getdata ();
 void putdata ();
 test sum (test);
};
 void test :: getdata ()
{
cout << "Enter value of a & b ";
cin >> a >> b;
}
void test :: putdata ()
{
cout << Value of a & b are: " << a << b;
}
test. test :: sum (test t2)
 {
 test t3;
t3. a = a + t2. a;
t3. b = b + t2. b;
 return t3;
}
 t2. Putdata () void main ()
{
  test t1, t2, t3;
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
t1.getdata();
t2.getdata();
t3 = t1.sum(t2);
t1.Putdata()
t2.Putdata()
cout << "t2 object data is:" << endl;
t3.Putdata()
```

- **Constructor:-** Automatic initialization the object

It is a special function that is automatically invoke when object is created.

Constructor does not have any data Type

Constructor name and class name are same.

Case-sensitive

Always public.

Program:-

```
class test
{
  int a, b;
 Public:
  test()    // constructor
 {
  a = 10;
  b = 20;
 }
  void show-data()
 {
  cout << a << b;
 }
```

Main Ideas, Questions & Summary:

Library / Website Ref.:-

```
};
main()
{
  ~~test t1();~~ test t1;
  t1.showdata();
}
```

- **Rules:-** i) constructor does not have any return type. including void
  ii) Constructor name & class name should be same
  iii) Constructor & destructore should always be public
  iv) constructor may or may not have argument hence it is possible to overload the constructor.
  v) Constructor ~~AA~~ never participant in inheritance.
  vi) we are not able to find out the address of the constructor and they will make implicit calls to new and delete operators.
  vii) Constructor may have default arguments
  viii) Constructor is only called once when object is created. further calling is not allowed.
  ix) Constructor never invoke with object name.

- **Types of Constructor:-** There are various type of constructor.
  i) Default Constructor
  ii) Parameterize Constructor
  iii) Copy Constructor.

⇒ **Default Constructor:-** without any argument

- Compiler designed or system designed
- user-defined

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

★ This Operator in c++ :- This pointer is used for returning the object address.

⇒ It is also to distinguished are data member from local variable when both are declared with the same name.

⇒ whenever data member and function argument are declared with the same name compiler gets confused that which one is the data member and which one is the argument to avoid this pointer problem and identify data members, we use this operator.

Example:-

```
class test
{
    int a, b;
public:
    void show(int a, int b)
    {
this -> a = a;  or (*this).a = a;
this -> b = b;  or (*this).b = b;
    }
main()
test t1;
t1.show(20,20);
```

• Construction Overloading :-

```
class Circle
{
    float r;
Public:
    circle()
```

eas, Questions & Summary:

bsite Ref.:-

```
{
    r = 5;
}
circle (float r)                    // Parameterized Constructor
{
(*this).r = r;
}
void showdata ()
{
cout << "Area : " << 3.14 * r * r << endl;
}
main ()
{
 circle c1;
c1.show_area ();
circle c2(10.5);
c2.show_area ();
 getch ();
}
```

Example II :-

```
#include <iostream.h>
#include <conio.h>
#include <string.h>
class stu
{
  char name[20], course[20];
  float fee;
  Public:
  stu (char name[20], char course[20])
{
 strcpy (this → name, name);
 strcpy (this → course, course);
  fee = 0;
}
stu (char name[20], char course[20], float fee)
{
stu::stu (name, course);    // constructor calling in constructor.
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
    this -> fee = fee;
  }
   void show ()
 {
cout << "name is :" << name << endl;
cout << "course is :" << course << endl;
cout << "fee is :" << fee << endl;
};
void main ()
{
  clrscr ();
stu s1 ("Roma", "C++");
stu s2 ("Varsha", "C++", 5000);
 s1.show stu ();
 s2.show stu ();
}
```

- **Copy constructor**:- Copy constructor is a constructor which is used to initialized the current object value with another value

ii) It is used in operator overloading concept.

iii) copy constructor are having reference type parameters.

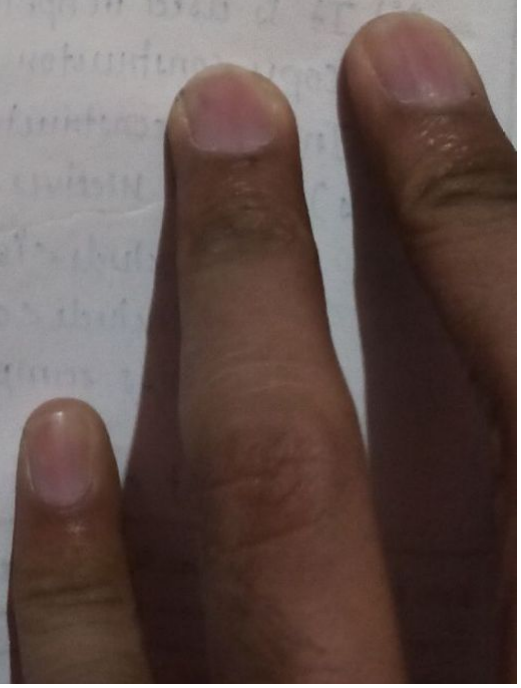
iv) In copy constructor we are having class type parameters (objects) i.e., it receives another object to initialized the current object.

```
#include <iostream.h>
#include <conio.h>
class sample
{
int a, b;
```

s, Questions & Summary:

ebsite Ref.:-

```
Public:
Sample(int a, int b)
{
 (*this).a = a;
 (*this).b = b;
}
Sample (sample & old)      // copy Constructor.
 {
   a = old.a;
   b = old.b;
 }
void print ()
 {
cout << "a is " << a << endl;
cout << "b is " << b << endl;
 }
};
void main ()
{
  sample s1 (10, 20);
  sample s2 (s1)
   s1. Print ();
   s2. Print ();
}
```



POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-
21. Aug.						

• Destructor in C++ :- Whenever object is created it occupied some memory.

~~Whenever~~ a constructor is executed when an object is defined so memory is allocated to release this memory destructor is used.

It is also a special member function

It destroy the memory

★ Rules :- Destructor name and class name should be same. and preceeded by a ~ operator.

ii) It should also declared in Public area.

iii) Destructor does not have any argument.

iv) Destructor Overloading is not possible.

v) Destructor never return any type.

vi) It ~~make~~ make implicit call to new and delete operator.

Main Ideas, Questions & Summary:

Library / Website Ref.:-

• Friend Function :- Friend function can be declare with any access specifier. (Public, Private, ~~st~~ Protected)

Friend function should be declare within the class using Friend keyword

Defination of friend function should be given outside the class without using the friend keyword.

We can introduce different function with in any no of classes.

Friend function have object as their argument.

Friend function is called like a normal function.

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-
20. Aug						

Example:- class sample

```
{ int a;
  int b;
friend void print(sample);
};
void print (sample s)
{
  s.a = 10;
  s.b = 20;
cout << "a is : " << s.a << endl;
cout << "b is : " << s.b << endl;
}
void main ()
{
  sample s;
  Print (s);
}
```

Example ii):- class test1; // forward declaration

```
class test2
{
  int a;
public:
void get_a ()
{
cout << "Value of a ";
cin >> a;
} friend void big (test1, test2);
};
class test1
{
```

Main Ideas, Questions & Summary:

Website Ref.:-

```
    int b;
   Public:
   void get_b()
   {
    cout << "value of b";
    cin >> b;
   } friend big (test t1, test t2);
};
main()
{
 test1.t1;
 test2.t2;
t2.geta();
t1.get_b();
    big (t1, t2); }
 void big (test1 t1, test 2 t2)
   {
     if (t2.a > t1.b)
   cout << "a is greater";
    else if (t1.b > t2.a)
   cout << "b is greater";
    else
   cout << "both are equal;
}
```

Ques. Write a program that adds two no. belonging two different classes. Demonstrate the use of friend function in one class, that is the member function of second class?

```
#include <iostream>
using namespace std;
class Test 2;
class Test 1
{
 int a;
 Public:
 void get_a()
{
  cin>>a;
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
    }
    friend void Test :: add (Test1)
    };
Class Test2
{
   int b;
Public:
void get-b()
{
  cin >> b;
}
void add (Test1)
};
void Test2 :: add (Test1 t1)
{
 cout << "sum=" << t1.a + b;
}
void main ()
{
 test1 t1;
 test2 t2;
 t1.get a();
 t2.get b();
 t2.add (t1);
}
```

Example:-

```
Class Distance
{
 int km;
 int m;
```

Main Ideas, Questions & Summary:

Library / Website Ref.:-

```
    friend void convert (distance d1)
};
    void convert (distance d1)
{
    cout<<"Enter the value of km";
    cin>>d1.km;
    cout<<"Enter the value of m";
    cin>>d1.m;
    cout<<"distance in meter:"<<d1.km x 1000+d1.m;
}
void main()
{
    distance d1;
    convert(d1);
}
```

- **Inheritance:-** It is a process of deriving new classes or class from existing old class or classes.

In other word, it is process of passing the property of one class to another. In this situation, the old class is referred as base class or super class and new created class as derived or sub class.

⇒ Syntax for inheritance:-

Derived class: Public / Private / Protected

visibility mode

- base class

When derived class object is created, it also creates the memory for the base class object.

Derived class's object size is sum of derived class data members and base class data members.

For Example:-

```
Class A
{
  Public:
  int x;
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
    float y;
};
    Class B: Public A
    {
      int N;
    };
main ()
{
  B.b;
```

{* Note:- Private data number of base class are not directly not available to the derived class. Hence, they do not participant in inheritance because of the data hiding concept. }

II nd Example:-

```
#include <iostream>
using namespace std;
Class B
{
  int a;
  Public:
  int b;
  void get_data1();
  void get_data2(void);
  void show (void);
};
Class D: Private B
{
 int c;
Public:
 void mul (void);
void display (void);
};
void B::get_data()
{
cout << "Enter value a & b";
cin >> a >> b;
}
int B::getdata2 ()
{
return (a);
}
void B::show ()
{
void D::mul ()
{
get_data1 ();
c = b * get_data2 ();
}
show ();
cout << " " << c;
}
int main ()
{
 Dd;
 d.get_data1 ();
```

(a will not work)

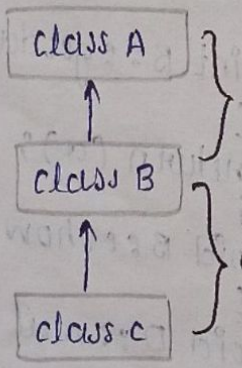
Main Ideas, Questions & Summary:

Library / Website Ref.:-

```
d.mul();
d.show();  (✗ will not work)
d.display();
d.b = 20;  (✗ will not work)
d.mul();
d.display();
return 0;
}
```

Output: Enter a & b: 5 10
c = 50
Enter a & b: 12 20
c = 240

- **Multilevel Inheritance:-** If a class is derived from another derived class then it is called multilevel inheritance. So, in c++ multilevel inheritance, a class has more than one parent class.



Class B derived from one base class A

Class C derived from derived class B

Syntax:-
```
class A  // base class.
{
-----
};
class B : access specifier A  //derived class
{
-----
};
class C : access_specifier B //derived from derived class B.
{
-----
};
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

For Example:-

```
#include <iostream>
class student
{
protected:
 int roll no;
public:
 void get_no(int);
 void put_no(int);
};
void student :: get_no(int a)
{
roll no = a;
}
void student :: put_no()
{
cout << "Roll number: " << roll_no;
}
class test :: public student
{
protected:
float sub1;
float sub2;
public:
void get_marks(float, float);
void put_marks(void);
};
void test :: getmarks(float x, float y)
{
sub1 = x;
sub2 = y;
}
void test :: put_marks()
{
cout << "Marks in sub1 = " << sub1;
cout << "Marks in sub2 = " << sub2;
}
class result : public test
{
float total;
public:
void display(void);
};
void result :: display(void)
{
total = sub1 + sub2;
 Put_no();
 Put_marks();
cout << "Total = " << Total;
}
int main()
{
result student1;
student1.get_no(111);
student1.get_marks(75.0, 59.5);
student.display();
return 0;
}
```

Main Ideas, Questions & Summary:

Library / Website Ref.:-

Output: Roll no. 111
Marks in sub1 = 75.0
Marks in sub2 = 59.5
Total = 134.5

- **Single level Inheritance:-**

```
#include <iostream>
using namespace std;
class stu
{
  private:
  int id;
  char name[20];
 public:
   void getdata()
 {
  cout << "Enter the Id:";
  cin >> id;
 cout << "Enter the name:";
  cin.ignore();
  cin.getline(name, 20);
 }
 void getstu()
{
 cout << "id = " << id;
 cout << "Name = " << name;
 }
};
class phy : Public stu
 {
  float hight, weight;
Public:
 void get~~data~~detail()
  {
   cin >> height >> ~~weight~~ ~~height~~; weight;
  }
 void show~~data~~detail()
  {
   cout << height << ~~height~~; weight;
  }
};
  main()
{
 phy P;
 P.getdata();
 P.putstu();
 P.getdetail();
 P.showdata();
}
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

- Static Function:- A static member function is a special member function which is used to access only static data member, any other normal data member cannot be accessed through static member function. Just like, static member function is also a class function; it is not associated with any class object.

```
Class Test
{
static int count;
  int num;
Public:
void getdata (int a)
{
  num = a;
  ++count;
}
void getcount ()
{
  cout << count;
}
};
int Test::count;
int main ()
{
  Test a, b, c;
a.getdata (10);
b.getdata (80);
c.getdata (90);
a.getcount ();
b.getcount ();
c.getcount ();
Return ();
}
```

Static me last count ki value
jo rhti hai baki sb me print hogi

Main Ideas, Questions & Summary:

Library / Website Ref.:-

• Operator which cannot be overloaded:- i) Scope ::

ii) sizeof iii) number (.) iv) pointer selector (*)

v) Ternary (?:)

```
class Test
{
Private:
  int num;
Public:
  Test ():num(8), i(2)
}
void operator ++ ()
{
 num= num+5;
}
void print()
{
cout << "num" << num;
}
};
int main()
{
 Test t1;
  ++t;
 t1.print;
return 0;
```

• Operator Overloading:-

```
Class Test
{
 Private:
 int n;
 Public:
void operator + (Test t)
 {
 int H = n++ ;
  cout << H;
}
Test (int i)
{
  n= i;
}
};
  int main()
 {
  test t1(5);
  Test t2(4);
  t1 + t2;
return 0;
}
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

• Function Overriding :- is a feature that allows us to have a same function in child class which is already present in the parent class. A child class inherits the data members and member functions of parent class, but when you want to override a functionality in the child class than you can use function overriding. It is like creating a new version of an old function, in the child class.

To override a function you must have the same signature in child class. By signature I mean the data type & sequence of parameters.

```
#include <iostream>
using namespace std;
class Base {
Public:
void display()
{
cout<<"Function of Parent class";
}
};
class derived : Public Base
{
Public:
 void display()
{
cout<<"Function of child class";
}
};
 int main()
{
 derived d = derived();
 d.display();
 return 0;
}
```

Main Ideas, Questions & Summary:

Library / Website Ref.:-

• Virtual function:- Virtual function is a member function in the base class that you redefine in a derived class.

It is declared using the virtual keyword.

It is used to tell the compiler to perform dynamic linkage or late binding on the function.

Virtual functions ensure that the correct function is called for an object, regardless of the types of reference (or pointer) used for function call.

Virtual functions should be accessed using pointer or reference of base class type to achieve run time polymorphism.

Pure Virtual function:- It is also called as abstract function. In c++, pure virtual function is a virtual function for which we don't have implementation, we only declare it.

It is declared by assigning 0 in declaration.

A pure virtual function is implemented by classes which are derived from a abstract class.

Example:-

```
// An abstract Class.
class Test
{
    // Data members of class.
public:
    // Pure Virtual function
    virtual void show() = 0;
    /* Other members */
};
```

Virtual Destructor: Deleting a derived class object using a pointer to a base class that has a non-virtual destructor results in undefined behaviour. To correct this situation, the base class should be defined with a virtual destructor.

For example:

```
#include <iostream>
using namespace std;
class base {
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
    Public:
        base ()
    {
cout << "Constructing base \n";
  ~ base ()
    {
    cout<<"Destructing base \n";
    }
};
class derived: Public base {
  public:
  derived ()
{
 cout << "Constructing derived \n";
}
 ~derived ()
{
 cout << "Destructing derived \n";
}
};
 int main (void)
 {
  derived * d = new derived ();
   base * b = d;
   delete b;
   getchar ();
   return 0;
}
```

- **Templates** (Generic Programming): Generic means independent of datatype
  keyword used is template.

Ma...s, Questions & Summary:

...Website Ref.:-

Templates are also called as generic programming.

Templates are features of c++ programming language that allows functions and classes to operate with generic type. This allows functions or class to work on many different data types without being rewritten for each one.

It is of great importance when combined with multiple inheritance and operator overloading. c++ standard library provides many useful functions within a framework of connected templates. Instead of execution, a program is written to be compiled in templates.

Templates can be used in 2 different ways:

(Function Templates)

Function template works is a similar way to a normal function with one key difference. A single function template can work with different data types at once but a single normal function can only work with one set of data types.

(class Templates)

In class template, we need a class implementation that is same for all classes only the data types used are different.

It makes easy to reuse same code for are data types.

Example: (function Template)

```
#include <iostream>
using namespace std;
template <class T>
T large (T n1, T n2)
{
return (n1 > n2) ? n1 : n2
}
int main ()
{
  int i1, i2;
float f1, f2;
char c1, c2;
cout << "Enter 2 integers : \n";
cin >> i1 >> i2;
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
cout << large (i1, i2) << "is larger" << endl;
cout << "Enter 2 float pointers : \n";
cin >> f1 >> f2;
cout << large (f1, f2). << "is larger" << endl;
cout << "Enter 2 characters : \n";
cin >> c1 >> c2;
cout << large (c1, c2) << "has large ASCII value" << endl;
return 0;
}
```

Example : (Class Template)

```
template < typename T; int N>
class Array
{
 T arr [N];
Public:
 int getsize ()
{
 return N;
}
};
int main ()
{
Array <int, 6> ar;
 cout << ar. getsize ();
 return 0;
}
```

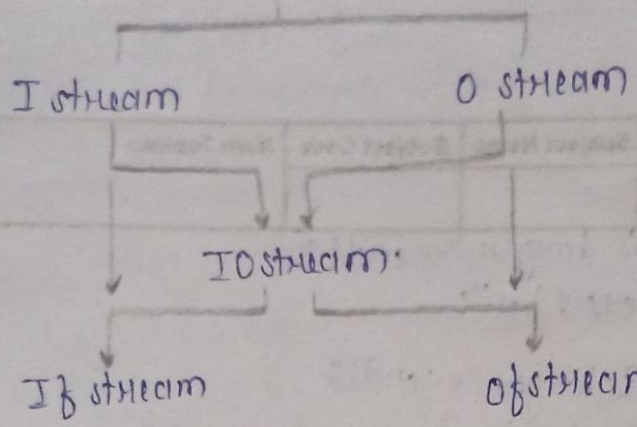
- File Handling

⇒

In Ideas, Questions & Summary:

Library / Website Ref.:-

4 Stream Classes



Stream can be represented as a source or destination of characters of indefinite length. A stream is an abstraction that represents a device on which operation are performed.

In c++ we have a set of file handling methods. These includes iostream, ostream and fstream.

Files are used to store data in a storage device permanently.

File handling provides a mechanism to store O/P of program in file & perform various operation.

⇒ Ifstream: Stream class signifies the input file stream and is applied for reading information from files.

⇒ Fstream: This stream class can be used for both read & write from/to files.

⇒ Ofstream: This stream class signified the output file stream and is applied to create files for writing information to files.

File Handling Operations: i) Creating a file: Open()
ii) Reading data: read()
iii) Writing new data: Write()
iv) Closing a file: close()

Default Open modes:
ifstream: in
ofstream: out
fstream: in and out

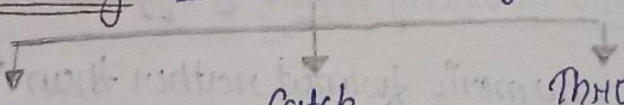
Example:
```
#include<iostream>
#include<fstream>
using namespace std;
int main()
{
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
fstream new_file;
new_file.open("new-file", ios::out);
if(!new-file)
{
cout<<"file Creation failed";
}
else
{
cout<<"New file Created";
new-file.close();
}
return 0;
}
```

- Exception Handling: Exception Handling



Catch keyword: Indicates the catching of an expression

Try keyword: A try block identifies a block of code for which particular exception will be activated.

Throw keyword: A program throws can exception when a problem shows up.

Exception is a problem that aries during the execution of a program.

Exception provide a way to transfer control from 1 part of a program to another.

Example: #include <iostream>
using namespace std;
double division (int a, int b)
{ if (b==0)

Main Ideas, Questions & Summary:

Library / Website Ref.:-

```
    {
      throw "Division by zero condition!";
    }
    return (a/b);
}
int main ()
{
 int x=50;
 int y=0;
double z=0;
try {
  z = division (x,y);
cout << z << endl;
}
catch (const char* msg)
{
cerr << msg << endl;
}
return 0;
}
```

⇒ Advantages of saving data in binary form: i) Efficient
ii) Reliable
iii) If we will store data in numeric formed rather than in text character it tends to use less memory.
iv) It offers advantages in terms of speed of access
v) Binary files are faster and easier for a program to read & write than text files.
vi) Best way to store program information.

⇒ Advantages of saving data in text form: i) It can be read by people.
ii) It can be ported to different type of system.

- Member function: A member function of a class is a function there has its ~~funct~~ definition or prototype within class definition like any other variables. It operator on any object of class of which it is a member and has a access to all the members of a class for that object.

Member function can be defined within the class definition or separately using scope resolution operator.

Example:
```
#include<iostream>
using namespace std;
class box
{
```

POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

```
public:
    double length;
    double breadth;
    double height;
double getVolume(void);
void setLength(double len);
void setbreadth(double bre);
    setHeight(double hei);
};
double box::getvolume(void)
{
  return length * breadth * height;
}
void box::setLength(double len) {
    length = len;
}
void box::setbreadth(double bre) {
    breadth = bre;
}
void box::setheight(double hei) {
    height = hei;
}
int main()
{
  Box Box1;
  Box Box2;
 double volume = 0.0;
Box1.setlength(6.0);
Box1.setBreadth(7.0);
Box1.setHeight(5.0);
Box2.setLength(12.0);
Box2.setBreadth(13.0);
Box2.setheight(10.0);
volume = Box1.getvolume();
cout << "volume of Box1 :"
    << volume << endl;
volume = Box2.getvolume();
cout << "Volume of Box2 :" << volume <<
    endl;
 return 0;
}
```

Output:

Volume of Box1 : 210

Volume of Box2 : 1560

Main Ideas, Questions & Summary:

Library / Website Ref.:-

- **Function Overriding:** If derived class defines same function as defined in its base class it is known as function overriding in c++.

It is used to achieve runtime polymorphism.

It enables us to provide specific implementation of function which is already provided by its base ~~also~~ class.

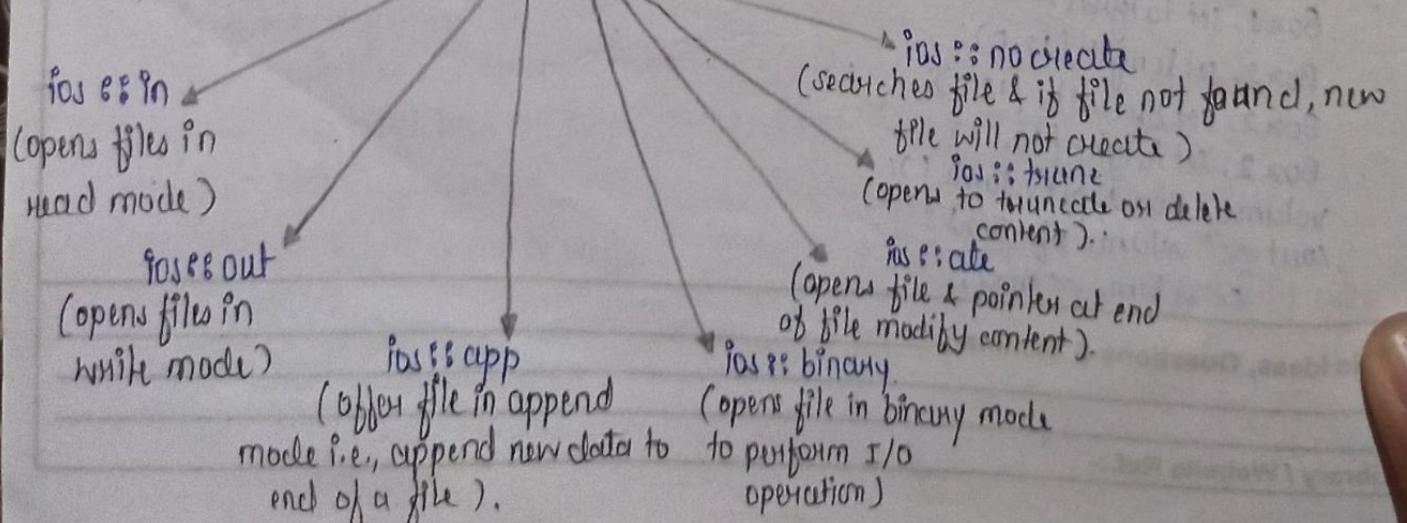
Example:
```
#include <iostream>
using namespace std;
class Animal {
public:
void eat() {
cout << "Eating";
}
};
class Dog : public Animal
{
public:
void eat()
{
cout << "Eating bread";
}
};
int main(void) {
dog d = Dog();
d.eat();
return 0;
}
```

- **File Mode:** In c++, for every file operation there are specific files modes.

These file modes allows us to create, read, write or modify a file.

File modes are defined in class

Types of files Modes



POORNIMA

Date	Unit No.	Lecture No.	Faculty	Subject Name	Subject Code	Main Topics:-

Method Overloading	Method Overriding
i) Occurs within same class.	i) Occurs b/w 2 classes (subclass & superclass).
ii) Inheritance not involved.	ii) Involved.
iii) Parameter must be diff.	iii) must be ~~diffe~~ same.
iv) Return type may or may not be same.	iv) Must be same.
v) Acces ~~Occurs~~ modifier & non ~~occurs~~ Access modifiers can be changed.	v) Non access ~~occurs~~ modified Final Static Access modifiers must be same.
vi) One method doesn't hide other.	vi) Child method hides that of parent class method.

Function Overloaded	Templates.
i) Duplicate code	i) Reuse code
ii) Maintaing is difficult	ii) Cause code block
iii) Each must be changed if there is need.	iii) They are like Macros.

- File Handling: Files are used to store data in a storage device permanently. File handling provides a mechanism to store the output of a program in a file and to perform various operations on it.

There are various modes of file handling:

i) In: Opens the file to read (default for ifstream)
ii) Out: Opens the file to write (default for outstream)
iii) Binary: Opens the file in binary mode.

Main Ideas, Questions & Summary:

Library / Web Ref.:-

iv) app: Opens the file and appends all the outputs at the end.

v) ate: Opens the file and moves the control to the end of the file.

vi) trunc: Removes the data in the existing file.

vii) nocreate: Opens the file only if it already exists.

viii) noreplace: Opens the file only if it does not already exist.

C++ provides us with the following operations in File Handling:

- Creating a file: Open()
- Reading data: read()
- Writing new data: write()
- Closing a file: close()