

I N D E X

NAME: Divyanshu STD.: _____ SEC.: _____ ROLL NO.: _____ SUB.: C++ programming

[illegible]

Structure of Student.

To display Name, Roll NO, and marks.

```
#include <stdio.h>
```

```
struct student  
{  
    int RollNO;  
    char Name [20]  
    float marks;  
} s;
```

```
void main () {
```

```
    printf ("Enter name of student");  
    scanf ("%d %s\n", &s.Name);  
    printf ("Enter Roll NO of student");  
    scanf ("%d\n", &s.RollNO);  
    printf ("Enter marks of student");  
    scanf ("%f\n", &s.marks);
```

```
    printf ("\n %d\n %s\n %f", s.RollNO, s.Name, s.marks);
```

```
    return 0;  
}
```

for 7/11/28 goul

Array of structures.

```
#include <stdio.h>
```

```
struct student {  
    int rollNo;  
    char name[20];  
    float marks[3];  
};
```

```
void main() {  
    int i;
```

```
    for (i=0, i<3, i++)
```

```
    { printf("Enter name of student");  
      scanf("%s\n", &S[i].name);
```

```
      printf("Enter RollNo of student");  
      scanf("%d\n", &S[i].rollNo);
```

```
      printf("Enter marks of student");  
      scanf("%f\n", &S[i].marks); } &S[i].marks);
```

```
      printf("\n\t roll.no, \t name \t marks");
```

```
    for (i=0, i<3, i++)
```

```
    { printf("\n\t %d \t %s \t %f", S[i].rollNo, S[i].Name, S[i].marks);  
      }
```

```
    return 0;  
}
```

Sum 7/11/20
gib &S[i].rollNo

unit I

Introduction of Operating system.

- 1] Overview of oop concepts.
- 2] OOP characteristics
- 3] Comparison with procedural programming.
- 4] Advantages and disadvantages of oop
- 5] Introduction of classes and objects.

- 1] Overview of oop concepts.

Differentiate between C++ and C.

	C++	C
1]	C++ is multiparadigm It supports both procedural procedural and object oriented	C follows the procedural style programming.
2]	Data is less secured in C By using modifiers for class members to make it inaccessible for outside users.	Data is less secured in C.

3]

C++

It has bottom up approach.

4]

It support functions overloading

5]

We can use function in structures.

6]

It support reference variables.

7]

C in and C out used for input or output operations

8]

Operator overloading is possible.

9]

C++ program divided into program and classes

10]

C++ support the feature of name space.

11]

C++ provides exception handling using try and catch block

12]

C++ support inheritance

C

It has top down approach

It doesn't support function overloading

Structure doesn't support function in C

Doesn't support reference variables.

scanf function and printf() are used for input and output

Doesnot support operator overloading.

C program are divided in procedures and modules

C does not provide the feature of namespace.

Exception handling is not easy in C

C does not support inheritance

Q1] Explain the history of C++?

- C++ is a general-purpose programming language developed by Bjarne Stroustrup at Bell Labs in the early 1980s. It was designed as an enhancement to the C language, incorporating features of object-oriented programming (OOP) while maintaining C's efficiency and flexibility.
- 1] 1979: Stroustrup began working on 'C with classes' combining the strengths of Simula (object-oriented features) and C [performance].
 - 2] 1983: The language was renamed C++, signifying an increment over C, reflecting its evolution. Key features like classes, inheritance, and basic function overloading were introduced.
 - 3] 1985: The first edition of the book "The C++ programming language" was published and the first commercial release of C++ became available.
 - 4] 1990s: This language evolved with the addition of features like templates, exception handling and the standard template library (STL).
 - 5] 1998: The first international standard for C++ (ISO/IEC 14882:1998) was published.
 - 6] 2011 and beyond: Major updates like C++11, C++14, C++17, C++20, and C++23 introduced modern features such as lambda expressions, smart pointers and improved performance tools.

C++ remains widely used in software development for its balance of high-level and low-level programming capabilities.

Q3]

Enlist and explain the features of C++?

- 1] Simple
- 2] Abstract Data types [user-defined Data types]
- 3] Machine independent or portable.
- 4] Mid-level programming language.
- 5] Structured programming language.
- 6] Rich library
- 7] Memory management.
- 8] Quicker compilation.
- 9] Pointers
- 10] Recursions
- 11] Extensible.
- 12] Object-oriented.
- 13] Compiler based.
- 14] Reusability
- 15] National standards.
- 16] Errors and easily detected.
- 17] Power and flexibility.
- 18] Strongly typed language.
- 19] Redefine Existing operations.
- 20] Modeling Real-World Problem
- 21] Clarity.

Qn1 Explain any 10 features of C++.

→ The 10 features of C++ are as following.

1] Object-Oriented Programming [OOP]

C++ supports OOP principles like encapsulation, inheritance and polymorphism, enabling modular and reusable code.

2] Multi-Paradigm.

C++ supports multiple programming paradigms, including procedural, object-oriented, and generic programming.

3] Encapsulation.

Data and functions are bundled together within classes restricting direct access to some of the objects components.

4] Inheritance.

Allows one class to inherit properties and behaviour from another, promoting code reusability.

5] Polymorphism.

Enables functions or operators to behave differently based on the context (Eg. function overloading and operator overloading)

6] Templates
Provides generic programming by allowing functions and class to operate with different data types without resulting in code.

7] Standard Template Library [STL]
Offers a rich collection of pre-defined classes and functions like vectors, lists, and algorithms, simplifying complex programming tasks.

8] Exception Handling.

Provides mechanisms to handle runtime errors using try, catch, and throw keywords, ensuring robustness.

9] Platform Independence.

→ ~~Provides~~ Though not inherently platform-independent C++ programs can run on any platform with a compatible compiler.

10] Speed and Efficiency.

→ C++ is closer to hardware and offers faster execution making it suitable for system-level programming and performance-critical applications.

Q5] Explain the Advantages and Disadvantages of C++

Advantages

Disadvantages

1] Object-oriented programming supports concepts like encapsulation and polymorphism making code modular.

1] Complexity complex syntax and features can make it challenging for beginners.

2] High performance
Compiles directly to machine code, providing faster execution compared to interpreted languages.

Manual memory management requires explicit handling of memory which can lead to errors like memory leaks.

3] Portability
Code can run on multiple platforms with minimal modifications, provided a suitable compiler exists.

Lack of built-in garbage collection
No automatic memory management increasing development efforts.

4] Rich standard library offers a powerful standard template library [STL] for data structures and algorithms.

Security Risks features like pointers and direct memory access can lead to vulnerabilities.

5] Scalability
Efficient for developing both small and large-scale applications from games to operating systems.

Difficult debugging complex error messages and debugging can be time-consuming.

6] Low-level manipulation allows direct access to hardware resources making it suitable for system-level programming

7] Backward-compatibility Easily integrates with older C code, ensuring continuity in legacy projects.

6] Lack of runtime errors checking errors like segmentation faults are detected only during runtime.

7] Compiler Dependency Some advanced features are implemented differently across compilers, causing inconsistencies.

✓
20/11/25

NotesSociety

Introduction to classes and object.

Definition

Class: Class is a template from which objects are created and it is a group of similar objects.

A class is a blue print for creating objects. It defines properties or class defines properties [data members] and [behaviour] → [member functions]

Syntax

To declare the class

To declare the class we used keyword 'class' defines properties [Data member] and [behaviours] → [member functions]

<pre> class class_name { Data members ; member functions ; } ; </pre>	or	<pre> class class_name { private : Data members ; // Data members means Variable Declaration . } ; </pre>
---	----	---

<pre> class class_name { object ; } ; </pre>	or	<pre> public : = member function ; // function = declaration } object ; </pre>
--	----	--

here private and public are access specifiers.

There are three type access specifiers.

- 1) Public
- 2) Private
- 3) Protected.

By default all data members and member functions are consider as private

By default all the properties are private in short

Eg of Syntax

class class-Name { }

↑
Keywords

Class Student / Class-Name ;

Data members or Variables Declaration
 {
 int Roll NO ;
 char Name [10]
 float present ;
 } → Access Specifiers.

Public :

Accept () ;
 Display () ; → Members functions or functions declarations

} S ; → Class termination

↓
Object of class.

Objects

- 1] Objects are instance of class
- 2] Objects codes properties and behavior of class.
- 3] By using we can access all the properties of class.
- 4] An object is any entity that has state and behaviour.

Example : Chair, pen, mobile, laptop.

- Collection of objects is called class
- Object can be logical and physical
- Object is a real world entity.
- Object is a runtime entity and it created at runtime.
- All the members of the class can be accessed through object.

NotesSociety

Syntax :- To declare objects.

Class name Objects name;

Eg : Student S; Car C;

here Students in the class and S is a object.

Name Space in C++

Syntax

Using namespace [STD];

This statement declare outside of main () and before void main ()

STD means standard used to assign the scope of object variable functions.

Name space gives space or allocate the space to object, variables, etc.

NotesSociety

Eg

STD: Function();
[scope resolution]

STD: [Scope resolution] Variables;

STD :: numbers.

Unit VII Class and Objects

- 1] Defining class in C++
- 2] Declaring objects
- 3] Access specifiers.
- 4] Member functions and data members.

*] Defining Classes in C++.

Q1] Write a C++ program to accept and display following properties of students :- Roll No, Name, marks

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

```
class student
```

```
{
```

```
public:
```

```
int Roll-No;
```

```
char Name[20]
```

```
float Percent;
```

```
};
```

```
void main()
```

```
{ student s;
```

```
cout << "Enter your name:";
```

```
cin >> s.Name;
```

```
cout << endl << "Enter your Roll NO,";
```

```
cin >> s.Roll.No;
```



```

cout << endl << "Enter your percentage :";
cin >> s_percent;
cout << endl << "Roll NO: " << s_roll NO << endl << "Name" << s_name
<< endl << "Percent" << s_percent;

```

Q] Write a C++ program to accept and display following properties of employee :- salary, Employ ID, Age, position.

NotesSociety

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

```
class student {
```

```
private
    int Roll-NO;
    char name[20];
    float percent;
}
```

```
public
    void accept()
```

// inline function definition -

```
{
    cout << "Enter your name:";
    <in>> Name;
    cout << endl << "Enter your Roll NO:" << endl;
    <in>> Roll NO;
    cout << "Enter your percentage:";
    <in>> Percent;
}
```

```
void display()
```

```
{
    cout << endl << "Roll NO:" << roll_NO;
    cout << endl << "Percent:" << percent;
}
```

```
};
```



```
void main ( )  
{  
    student s;  
    s.accept ( );  
    s.Display ( );  
}
```

✓
for
20/1/21

NotesSociety

→ ② Create Declare and label furniture class

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

Class furniture.

Private:

```
int length;
int width;
char material;
float price;
```

Public:

void accept()

```
{
    cout << "Enter your name the length,"
    << cin >> name length;
    << cout << endl << "Enter width," << endl;
    << cin >> width;
    << cout << "Enter material name:";
    << cin >> material name;
    << cout << endl << "Enter the price,"
    << cin >> price;
}
```



```
void Display()
```

```
{
    cout << endl << "length:" << length << endl;
    cout << endl << "width:" << width << endl;
    cout << endl << "material:" << material << endl;
    cout << endl << "price:" << price << endl;
}
```

```
void main()
```

```
{
    Furniture F;
    F.accept();
    F.display();
}
```

NotesSociety

Q2]

Create class of car.

include ~~<io stream>~~
using namespace std;

Class Car:

Private:

int carmodel no;
char car company;
float price;

Public:

void accept()

```
{  
    cout << "Enter Carmodel no,"  
    <in >> model no;  
    cout << "Enter Car Company";  
    <in >> Car company;  
    cout << "Enter price";  
    <in >> price;  
}
```



```
void Display () {
```

```
    cout << endl << "Model number" << modelNo ;
    cout << endl << "Car Company" << carCompany ;
    cout << endl << "Price" << price ; }
```

```
void main ()
```

```
{ Car c ;
  c.accept () ;
  c.display () ;
```

```
}
```

NotesSociety

Q8] Create course class .

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

```
Course class {
```

```
private :
```

```
    int duration ;
    char degree name ;
    float price ;
```

```
public :
```

```
void accept ()
```

```
{
```

```

    cout << "Enter course duration",
    cin >> Course duration;
    cout << "Enter degree name";
    cin >> degree name;
    cout << "Enter prize"
    cin >> prize;
}

```

```

void display () {

```

```

    cout << endl << "Course duration" << duration;
    cout << endl << "Degree name" << degree name;
    cout << endl << "Prize" << prize;
}

```

```

void main () {

```

```

    { Course duration. C

```

```

    C. accept ();
    C. display ();
}

```

for

Write a C++ program to accept and display following of book property class

1. Book class: price, publisher accept and display by data member name and other.

// include <iostream>
using namespace std;

Private

class Property of Books.

int Char Book name;
float price of Book;
Char Book author name;
Char publisher name;

Wrote again? sure

Public:

void accept()

```
{
    cout << "Enter book name ";
    cin >> Book name;
    cout << "Enter price of Book ";
    cin >> price of Book;
    cout << "Enter book author ";
    cin >> Author name;
    cout << "Enter publisher name ";
    cin >> publisher name;
}
```

void Display () {

```
cout << endl << "book name" << book name ;  
cout << endl << "price of book" << price ;  
cout << endl << "author name" << author name ;  
cout << endl << "publisher name" << publisher name ;
```

void main ()

```
{ course duration  
  Book Property B ;
```

B.accept () ;

B.Display () ;

}

NotesSociety

Write C++ for class library with data members
total number of books, total cost of library book

→ #include <iostream>
using namespace std;

Private
Property of Library.
char

char Int Total number of book;
Double Total cost of book;

Public:

void accept ()

NotesSociety

```
{
    cout << "Enter total no of Book";
    cin >> total Books.
    cout << "Enter total no of cost";
    cin >> price
}
```

void display () {

```
    cout << endl << "Total Book" << All Books.
    cout << endl << "Total cost" << Total cost; }
```

void main ()

```
{
    Property of library. l
    l. accept ();
    l. display ();
}
```

3-

CLASSMATE
Date _____
Page _____

Syntax for outside class definition

{function define outside of class}

```
prototype class_name :: Function_name ()  
{  
    // block of code  
}
```

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

```
class student
```

```
{  
    private :
```

```
        int Roll No;
```

```
        char name[20];
```

```
        float percent;
```

```
    public :
```

```
        void Accept ();
```

```
        void Display ();
```

```
};
```

```
void student :: Accept ()
```

```
{
```

```
    cout << "Enter your name : ";
```

```
    cin >> name;
```



```
cout << endl << "Enter your roll NO" << endl;
<in >> roll.No;
cout << "Enter your percentage: ";
<in >> percent;
```

```
void main()
```

```
{
    Student S;
    S.accept();
    S.Display();
}
```

NotesSociety

Functions in C++

- Function: It is a small set of executable code.
- 2] Functions written a sign from main function.
 - 3] There are main two type of function.
Pre-define function
User define functions.

A] User-Define function.

→ 1] Based on problem statement user can create there own functions are called user define functions.

- 2] A function is block of code which only runs when it is called.
- 3] We can pass data [parameters] into a functions.
- 4] Functions are use to perform certain action and they are important for reusing code. — define the code once and use it many times.
- 5] Functions are also known as procedure or subroutines in other programming language.
- 6] It provides modularity and reusability.
- 7] Functions make the code optimised so. No need to

(A) Inside class definition:

→ In this way we can define function inside the class.

Ex

```
class student {  
    { public;  
    void accept ()  
    { cout << "Hello";  
    }  
};
```

Outside class definition

→ In this way we can define the function outside the class by using scope resolution operator [::]

Prototype class_name :: function_name ()

```
{  
    ---  
}
```

Ex

void student :: Accept ()

```
{  
    ---  
}
```


→ Define function outside of class with public specifier

SA

Class Student

{ Private :

```
public:
    void Accept();
    ~~~~~
};
```

Q. Write a C++ code to create a different classes with outside class definition to accessing there data member.

* Friend function

- 1] A friend function is a function of the class define outside the scope but it has the write to access.
- 2] It has the ~~same~~ right to access all the private and protected by of member of class.
- 3] The friend functions appears in the class definition but the friends are the member function.
- 4] It can be invoked like a normal function without using function [Object].
- 5] It can be declared either in private or public part.
- 6] The situation that the outside functions need to access the private member of the class so in that case friend functions come into the picture.
It is used
The friend function declared using

Friend key word :-

Syntax of declaration

class class_name

{
.....
}

return type function_name (parameter).
==>

Lab Assignment : III .

Create a class named weatherReport that holds a daily weather report with data member's day of month, hightemp, lowtemp, amount Rain and amount snow. Use default constructor to initialize the objects. Also include a function that prompts the user and sets values. Write C++ program to enter data and generate monthly average report and display it.

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

```
Class Weatherreport {
```

```
private:
```

```
int day - of - month;
double high - temp;
double low - temp;
double amount - rain;
double amount - snow;
```

```
public:
```

```
weatherreport(): day - of - month(0), high - temp(0.0),
low - temp(0.0) amount - rain(0.0) amount - snow(0.0)
{ }
```

Can be written like-

```
* OK  
weather Report () {  
    day - of - month = 0;  
    high - temp = 0.0;  
    low - temp = 0.0;  
    amount - rain = 0.0;  
    amount - snow = 0.0; }*/
```

Void input Report (int day)

```
{  
    day - of - month = day;  
    Cout << "Enter details for day: " << day - of - month << " : \n";  
    Cout << "High temperature: ";  
    Cin >> high - temp;  
    Cout << "Low temperature: ";  
    Cin >> low - temp;  
    Cout << "Amount of Rain (in mm): ";  
    Cin >> amount - rain;  
    Cout << "Amount of Snow (in mm): ";  
    Cin >> amount - snow;  
}
```


// Function to input data.

```
void inputReport (int day) {
```

```
    day - of - month = day;
    cout << "Enter details for day" << day - of - month << ": \n";
    cout << "Enter details for day" << day - of - month << ": \n";
    cout << "High temperature : ";
    cin >> high - temp;
    cout << "Low temperature ";
    cin >> low - temp;
    cout << "Amount of Rain (in mm) : ";
    cin >> amount - rain;
    cout << "Amount of snow (in cm) : ";
    cin >> amount - snow;
```

// Function to display data.

```
void display () const { cout << "Day - of - month" << "High"
                        << high - temp << "C";
    cout << "Low" << low - temp << "C, Rain" << amount - rain;
    cout << "mm, Snow" << amount - snow << "cm" << endl;
```

// Getter function to retrieve data.

```
double get hightemp () const { return high - temp; }
double get low temp () const { return low - temp; }
double get amount Rain () const { return amount - rain; }
double get amount snow () const { return amount snow; }
};
```



```
int main () {
```

```
int N days
```

```
cout << "Enter the number of days in the month ; "
```

```
cin >> N days ; // Assume 31 days of the month
```

```
Weather Report reports [31]; // array of object
```

```
for (int i = 0; i < N days; i++) {
```

```
reports [i] input Report (i+1);
```

```
}  
double total high T = 0, total Low T = 0, Total Rain = 0;  
total snow = 0;
```

NotesSociety

```
for (int i = 0; i < N days
```


Types of Constructor

Parameterized Constructor: It is one of the type of constructor

1] Syntax:

which have parameter with that as like simple function

2] Constructor name is as its class name. Same as default constructor, parameterized constructor are also invoked automatically when object is created. We can pass the values or parameters to the parameterized constructor either implicitly or explicitly

Syntax

```
class      class_name  
{ private :  
    // Data members
```

```
    public :  
        // member functions  
        // parameterized constructor
```

```
    class_name (parameters)
```

```
{ // body of parameterized constructor
```

```
    == .
```

```
}
```

Constructor has no any ~~or~~ written type and there is no need to have written values.

It

Ex

Class A

```
{ Private
  int x;
```

Public:

```
  A (int val)
```

```
{
```

```
  x = val;
```

```
} // Parameterized Constructor.
```

```
int main()
```

```
{ A a(10)
```

// implicitly values passed to Constructor.

```
  cout << a.x;
```

```
  return 0;
```

```
}
```


Example Create class Student with their data members Roll NO, Name, fees used parameterized constructor to initialize the constructor.
Write a C++ program to read the data and display it.

```
#include <iostream.h>
#include <string.h>
```

```
class Student
{
private
    int Roll NO;
    int Fees;
    string Name;

```

```
public:
    Student (int R, int F, string N)
    {
        R.NO = R;
        Fees = F;
        Name = N;
    }
};
```

```
void main
{
    Student s(10, 400, XYZ);
```

```
cout << s.R << endl << s.F << endl << s.N << endl;
}
```

```

cout << "Roll NO : " << S.R << "\n";
cout << "Name : " << S.N << "\n";
cout << "Fees : " << S.F << "\n";

```

Employee

```

#include <iostream.h>
#include <string.h>

```

Class Employee.

```

{ private.

```

```

    int Employee id;

```

```

    String Employee Name;

```

```

    Float Salary;

```

Public

```

Employee (int id, String Name, Float Salary)

```

```

{ id = S;

```

```

  Name = N;

```

```

  Salary = F;
}

```

```

};

```

void main

```

{ Student S

```

```

  Employee

```


Lab Assignment 4

Create a class called Time that has three integer data member for hours, minutes and second. A constructor to initialize the object to zero. A constructor to initialize the object to some constant value. A member function to add two Time objects. A member function to display time in HH:MM:SS format. Write a main function to create two time objects add them and display the result in HH:MM:SS format.

```
#include <iostream>
#include <iomanip>
```

Class Time

private:

```
int hour;
int minutes;
int second;
```

public:

```
void Time()
{
    Time()
    {
        h = 0;
        m = 0;
        s = 0;
    }
}
```

Time () *

```
{ hours = 0;  
  minutes = 0;  
  second = 0;
```

```
}
```

Time (int h, int m, int s)

```
{ hours = h;  
  minutes = m;  
  second = s;
```

```
}
```

NotesSociety

Copy Constructor

One of the types of constructor.

→ To copy the value of existing object values we require copy constructor.

If we have an object of a class and we want to make a copy in a new declared object of the same class then copy constructor is used.

The compiler provides each class a default copy constructor and users can define it also. It takes a single argument which is an object of the same class.

NotesSociety

Syntax

```
class class name {
```

```
private :
```

```
// private members
```

```
public
```

```
// declaring copy constructor.
```

```
class name [const class name &obj] {
```

```
// constructor body
```

- 1] A copy constructor which takes an object of same class but it is declared constant and passed as fn. parameter it creates a copy for it. So create that copy compiler will again call the copy constructor.
- 2] In the above process this will take memory and Recursion of creating copies.
- 3] To prevent such condition be passed it as a reference.

NotesSociety

Assignment: 4.

Create a class called "Time" that has three integers as member for hours, minutes and seconds. A constructor to initialize the objects to zero. A constructor to initialize the object to same constant value. A member function to add two time objects. A member function to display time in HH:MM:SS format. Write the function to create two time objects add them and display the result. HH:MM:SS.

```
#include <iostream>
#include <iomanip>
```

```
class Time {
```

```
private:
```

```
int hours, minutes, seconds;
```

```
public:
```

```
Time() {
    hours = 0;
    minutes = 0;
    seconds = 0;
}
```

```
Time(int h, int m, int s) {
    hours = h;
    minutes = m;
    seconds = s;
}
```

Time add (Time other)

```
{ int total second = (hours + other.hours) * 3600 + (minutes + other.min) * 60 + (seconds + other.seconds);
```

```
int h = total seconds / 3600;
int m = (total seconds % 3600) / 60;
int s = total seconds % 60;
return Time (h, m, s);
}
```

void display ()

```
{ cout << setw(2) << setfill('0') << hours << ":" <<
  << setw(2) << setfill('0') << minutes << ":" <<
  << setw(2) << setfill('0') << second << endl;
}
```

int main () {

Time time1 (1, 5, 39);

Time time2 (4, 10, 45);

Time result = time1.add (time2);

result.display ();

return 0;

output 06:06:24

28/12

Function overloading.

- 1] Function overloading in C++ allows you to define multiple function with the same name, but with different parameter types or a different number of parameter.
- 2] To define multiple function with same name, making but with different parameter type.
- 2] Constructor overload
- 3]

Constructor overload.

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

```
class Print {
```

```
public
```

```
void show (int i)
```

```
{ cout << "Integer: " << i << endl; }
```

```
void show (float f)
```

```
{ cout << "Float: " << f << endl; }
```

```
void show (string s)
```

```
{ cout << "String: " << s << endl; }
```

```
int main () {
```

```
Print obj;
```

```
obj.show(5);
```

```
obj.show(3.14f);  
obj.show("Hello!");
```

Constructor Overloading

Constructor overloading in C++ refers to the concept of having multiple constructors with the same name but with different parameter lists.

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

```
class Box {
```

Private:

```
    double length;  
    double width;  
    double height;
```


public ;

Box()

```
{ length = 1.0;
  width = 1.0;
  height = 1.0;
```

```
cout << "Default constructor called." << endl;
```

Box(double side)

```
{ length = side;
  width = side;
  height = side;
```

```
cout << "Constructor with one parameter called."
      << endl;
```

```
}
```

Box(double l, double w, double h)

```
{ length = l;
  width = w;
  height = h;
```

~~void display~~

cout << "

void display()

```
int main()
```

```
{
```

```
Box box1;
```

```
Box box2[8.5]
```

```
Box box3[2.0, 3.0, 4.0];
```

```
cout << "\n Box 1 Diameter: ";  
box1.Display();
```

```
cout << "\n Box 2 dimensions: ";  
box2.Display();
```

```
cout << "\n Box 3 dimensions: ";  
box3.Display();
```

```
return 0;
```


Operator Overloading

operator overloading in C++ allows you to define the behaviour of operators.

operator overloading means giving a new meaning or functionality

NotesSociety

5/3/25

Memory Management with Constructors and Destructor

Why is memory management Important.

→ when working with dynamic memory allocation, it is critical to manage memory properly. following points are useful to proper memory management,

1] Constructor }
Distractor }

Q S1 → display()

→ this pointer // symbol of this pointer

sym

This pointer symbol [→ this]

Q Define and explain pointer this pointer.

Inheritance

- 1] Concept of inheritance
 - 2] Type of inheritance
 - 3] Single
 - 4] multiple, multilevel .
 - 5] hierarchical hybrid .
 - 6] Base and derived class
 - 7] Access and control inheritance
 - 8] Virtual function key
- } Types of inheritance .
- ↳
- Virtual keyword

Benefits of inheritance

- 1] Code Reusability → Reduces redundancy by allowing classes to reuse existing properties and methods.
- 2] Hierarchical Organization: Helps in structuring code with parent-child relationships making it more manageable.
- 3] Faster Development - Speeds up implementation by leveraging previously defined functions.
- 4] Easy Maintenance - Changes made in the base class automatically reflect in derived classes reducing efforts.
- 5] Extensibility: New features can be added to derived class without modifying existing code.

Access specifiers

- 1] Public mode
- 2] Protected mode
- 3] Private mode.

Mode of inheritance or Access control in inheritance.

- 1] Mode of inheritance controls the access level of the inheritance members of the base class and in the derived class.
C++ has three modes → public mode
→ private mode
→ protected mode.

Public inheritance mode



- 1] If we derive a subclass from a public base class
- 2] Then the public member of the base class will become public in the derived class and protected members of base class will become protected in the derived class.

class ABC : public XYZ

// public derivation.

{ // members
// member function?;
}

Notes Society

Accessibility in public

Accessibility	Private members	Protected members	Public members
Base	Yes		

Type of Inheritance

- 1] Single inheritance $\rightarrow$ Single Base class and single derived class
- 2] Multiple Inheritance
- 3] Hierarchical inheritance
- 4] Hybrid inheritance

$\rightarrow$ Derived class $\Rightarrow$ A derived class

1] Single Inheritance : is defined as the inheritance in which a derived class is inherited from the only one Base class.

A $\rightarrow$ Base class



B $\rightarrow$ derived class

When one class inherits another class it's known as Single level inheritance

Q. Class A is Base class have the data members and two integer numbers. B is a derived class having data member having 2 float number. Write a C++ code to display the addition of two number. Here base class derived class inherit property of Base class.

A-

#include ~~is~~ <iostream>
using namespace .

Class
{
public
int A
int B

#include <iostream>
using namespace .

Class .
{ private.
int A
float B.

Public :
void calculate - int Add ()
{ z = x + y ;
}
}

Class B : Public A
Private :
float a, b, c

```

class public :
    void calculate int Add()
    {
        c = a + b;

        return Z;
    }
};

```

```

int main()
{
    B obj;
    obj.calculate - int - Add(12, 10);
    obj.calculate - flt - Add(10.5, 20.5);
}

```

- Class Account having data member salary worker is a derived class with bound data member write a c++ code to derive the above class properly publicly and display worker salary and bonus.

```

#include <iostream>
using namespace .

```

```

class
{

```


#include <iostream>
using namespace std;

Class A

{

Private:

int x = 10, y = 20, z =

Public

void calculate - int - Add()

{ z = x + y;

cout << z << " z << endl;

}
};

Class B : public A

{ private

float a = 10.5, b = 20.5, c;

Public

void calculate - float - Add()

{

c = a + b

cout << "c" << c << endl;

}
};

```

int main()
{
    B obj;
    obj.calculate_int_Add();
    obj.calculate_Flt_Add();
}

```

Accessibility in Public

1] If

Accessibility	Private member	Protected member	Public members
Base class	Yes	Yes	Yes
Derived class	No	Yes	Yes

1] If we derive a subclass from public base class.

2] Then the public member of the base class will become public in the derived class and protected members of the base class will become protected.

Syntax

```

class ABC: public XYZ
{
    // public derivation
    // members
    // member functions
}

```


Concepts of

Accessability in public.

Public Inheritance

→ Syntax

Class

Modes in inheritance

1] Type of inheritance

- 1] Single inheritance.
- 2] Multiple inheritance
- 3] Multi-level inheritance
- 4] Hierarchical Inheritance
- 5] Hybrid inheritance

1] Single inheritance

Q What is polymorphism in C++ , 3) What is the purpose of polymorphism in OP.

- 3) how do you achieve polymorphism in OP: → static
31] What is run time polymorphism
5] What is compile type polymorphism
6] What is function overloading
7] What is function overriding
8] Diff b/w virtual functions and overridden function.
9] What is pure virtual function in C++
10] What is an abstract class in C++

Q.2

→ 1] Polymorphism in object-oriented programming (OOP) is a concept that allows one interface to be used for different data types.

2] here are five key points explaining its purpose.

A] Code Reusability: Polymorphism allows a single function or method to work with different types of object reducing code duplication by promoting reuse.

B] Flexibility And Extensibility: It allows new classes to be added with minimal changes to existing code.

C] Promoting easy system extensions
It provides Improved Maintainability.

Dynamic Method Binding
simplifies the code.

In C++, polymorphism is a core concept of object-oriented programming [oop] allowing you to use a single interface to represent different underlying forms.
It enables one function, operator or method to behave differently based on the object it is operating on.

1] Compile-time polymorphism [static polymorphism]

2] Run-time Polymorphism [Dynamic Polymorphism]

3] how & do you achieve polymorphism in C++ static and dynamic.

Q17 Runtime polymorphism.

- It is also known as late binding (dynamic) polymorphism.

- function in derived class have same name and same behaviour

- As a function in a base class but provide different implementation

- Statement If derived class defines same function as defined in its base class is known as function overriding.

- It is used to achieve runtime polymorphism. It enables specific implementation of a function which is already provided by the base class.

→ The base class function must be declared as "virtual" to achieve overriding.

→ The function in a derived class must have the same name, written types and parameters as the function in the base class.

15 To override the function in derived class by using 'override' keyword.

To override the function ~~is~~ (over time), base class pointer pointing to derived class object.

```
a = &b;
```

By using base class class pointer. This operation [→] implements overriding function.

```
int main()
```

```
{ A *a;  
  B b;
```

```
  a = &b;
```

```
  a = Display();
```

```
  return 0;
```

```
}
```

NotesSociety

implement fun overriding with ~~base~~ have base class Parent has three member fun display the value for data member name & phone no, override display function in the derive class to display the data member derive class child name & phone no.

```
#include <iostream>
```

```
using namespace std;
```

```
class Parent
```

```
{ public
```

```
  string name = "ABC";
```

int phone NO = "123456789";

void Display()

```
{ cout << "name" << Name << endl;
  cout << "phone NO =" << phone NO;
}
```

Class Child : Public Parent

```
{ public :
  int (phone NO = "986718835")
```

void Display Overriding()

```
{ cout << "phone.NO =" << (phone NO);
}
```