OOP MSBTE PAPERS

2 Marks Questions

Winter-19

1 State the difference between OOP and POP.

Sr.	OBJECT ORIENTED PROGRAMMING (OOP)	PROCEDURE ORIENTED PROGRAMMING (POP)
1	Focus is on data rather than procedure.	Focus is on doing things (procedure)
2	Programs are divided into multiple objects	Large programs are divided into multiple functions
3	Data is hidden and cannot be accessed by external functions.	Data move openly around the system from function to function
4	Objects communicate with each other through function	Functions transform data from one form to another by calling each other.
5	Employs bottom-up approach in program design	Employs top-down approach in program design.
6	Object oriented approach is used in C++ language	Procedure oriented approach is used in C language.

2 What is a class? Give its example.

Class is a user defined data type that combines data and functions together. It is a collection of objects of similar type.

Example:

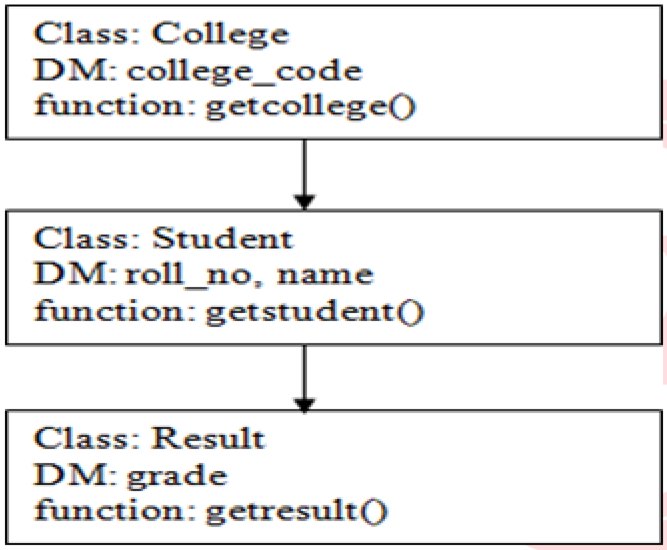
```
class Student
{
int rollno;
```

char name[10];

public:
void getdata( );
void putdata( );
};

3 What is multilevel inheritance? Draw the diagram to show multilevel inheritance. using classes with data members and member functions.

When a class is derived from another derived class then it is called multilevel inheritance.



4 Explain use of scope resolution operator.

It is used to uncover a hidden variable. Scope resolution operator allows access to the global version of a variable. The scope resolution operator is used to refer variable of class anywhere in program. :: Variable_name

OR

Scope resolution operator is also used in classes to identify the class to which a member function belongs. Scope resolution operator is used to define function outside of class.
Return_type class_name:: function_name( )
{
Function body
}

5 Write two properties of static member function.

i) A static member function can have access to only other static data members and functions declared in the same class.
ii) A static member function can be called using the class name with a scope resolution operator instead of object name as follows:
class_name::function_name;

6 Explain virtual base class with suitable example.

A virtual base class (Grandparent class) is a class that avoids duplicate on of inherited data in derived class (child class) derived from parent classes (parent1 and parent2) which in turn derived from base class.
e.g.

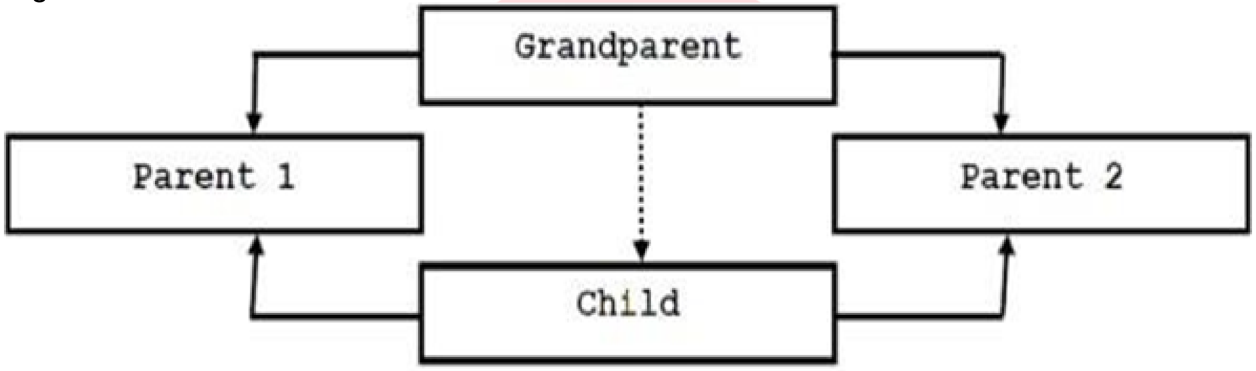


Fig. a: Virtual Base Class

7 Give syntax and use of fclose ( ) function.

Syntax:
int fclose(FILE* stream);

Use: This function is used to close a file stream.The data that is buffered but not written is flushed to the OS and all unread buffered data is discarded.

2 marks Questions
Summer-19

1.State the use of cin and cout.

cin: cin is used to accept input data from user (Keyboard).
cout:cout is used to display output data on screen.

2.Write syntax to define a derived class

Syntax:
class derived_class_name : visibility_mode/access_specifier base_class_name
{
class body
};

3.State use of scope resolution operator.

It is used to uncover a hidden variable. Scope resolution operator allows access to the global version of a variable. The scope resolution operator is used to refer variable of class anywhere in the program. :: Variable_name

OR

Scope resolution operator is also used in classes to identify the class to which a member function belongs. Scope resolution variable is used to define function outside of class.
Return_typeclass_name:: function_name( )
{ }

4.Define class and object.

Class: Class is a user defined data type that combines data and functions together. It is a collection of objects of similar type.
Object: It is a basic run time entity that represents a person, place or any item that the program has to handle.

5.Write the use of ios : : in and ios : : out.
OR
Describe meaning of the following
(i) ios : : in
(ii) ios : : out

ios::in - It is used as file opening mode to specify open file reading only.
ios::out- It is used as file opening mode to specify open file writing only.

6.Describe use of static data member.

Use of static data member: Static data member (variable) is used to maintain values common to the entire class. Only one copy of static member is created for the entire class and is shared by all the objects of that class. Its lifetime is the entire program.

7.Give meaning of following statements:
int *ptr, a = 5;
ptr = & a;
cout<< * ptr;
cout<< (* ptr) + 1;

int *ptr, a = 5;
Declare pointer variable ptr and variable a with initial value 5.
ptr = & a;
initialize pointer variable with address of variable a (store address of variable a in ptr)
cout<< * ptr;
Displays value of a i.e. value at address stored inside ptr. It displays value 5.
cout<< (* ptr) + 1;
Displays value by adding 1 to the value at address stored inside ptr. It displays value 6

Winter- 18
2 Marks Questions

1.State any four object oriented languages.

- **Object oriented programming language:**
- C++
- Smalltalk
- Object pascal
- java Simula
- Ada
- Turbo pascal
- Eiffel
- C#
- Python

2.Describe use of protected access specifier used in the class.

Protected access specifier is use to declare a class member that is accessible by the member functions within its class and any class immediately derived from it.

3.Write any two characteristics of destructor.

Characteristics:
1. It is used to destroy objects created by a constructor.
2. Name of destructor and name of the class is same.
3. Its name is preceded with tilde (~) symbol.
4. It never takes any argument.
5. It does not return any value.
6. It is invoked implicitly by the compiler upon exit from the program (or block or function) i.e when scope of object is over.

4.Give output for following code:

```
class student
{
int roll no;
char name [14];
} s[6];
void main()
{
cout<<sizeof(s);
}
```

Considering roll_no (Single variable) the output is: 96
OR
Considering roll, no (Two variables) the output is: 108
OR
Considering roll no the output is: error – space between roll and no

5.Describe derived class with example. 2M

Derived class: In inheritance a new class is derived from an old class. The new class is referred to as a derived class. The derived class can inherit all or some properties of its base class.

Example:
class base

```
{
};
class derived: public base
{
};
```

Winter-22
2 Marks Questions

1.State any Four application of oop.

Object-Oriented Programming (OOP) is a programming paradigm that is widely used in software development for various applications. Here are four common applications of OOP:

1. Software Development: OOP is extensively used in software development to model real-world entities, structure code, and facilitate code reuse. Object-oriented languages like C++, Java, and Python are used to build a wide range of software applications, including desktop applications, web applications, mobile apps, and more. OOP helps in organizing code into classes and objects, making it more modular and maintainable.

2. Game Development: Game development often involves complex systems with multiple interacting elements. OOP is well-suited for modeling game objects, characters, behaviors, and interactions. Game engines like Unity (C#), Unreal Engine (C++), and Godot (GDScript) use OOP principles to create games with rich and dynamic environments.

3. Simulation and Modeling: OOP is used in simulations and modeling to represent and simulate real-world processes, systems, and phenomena. Whether it's simulating traffic flow, financial markets, ecological systems, or physical simulations in scientific research, OOP allows developers to create modular and easily extendable models.

4. Graphical User Interface (GUI) Development: Creating user interfaces for software applications is a common use case for OOP. OOP allows developers to design and manage GUI components as objects, making it easier to create interactive and user-friendly interfaces. Frameworks like Qt (C++), JavaFX (Java), and Tkinter (Python) use OOP concepts for GUI development.

2.Explain user defined datatype with example.

User-Defined Data Types (UDTs)

In Object-Oriented Programming (OOP), user-defined data types (UDTs) are custom data structures created by programmers to represent specific entities, concepts, or data in a program. UDTs allow you to define your own data structures with associated attributes and behaviors, encapsulating related data and functions into a single unit.

Key Concepts:

1. **Custom Data Structures:** UDTs enable you to define custom data structures that may not be available as built-in data types in the programming language. These structures can represent real-world objects, concepts, or complex data.

2. **Abstraction:** UDTs provide a level of abstraction by hiding the internal details of data and exposing only the essential attributes and behaviors. This abstraction simplifies the use of the data type.

3. **Encapsulation:** UDTs encapsulate data and related functions within a single unit. This encapsulation helps in maintaining data integrity and preventing unauthorized access to internal data.

4. **Code Reusability:** By defining UDTs, you can create reusable components that can be used in different parts of your program. This promotes code reusability and modularity.

5. **Example**: For instance, in a financial application, you might define a `BankAccount` UDT to represent a bank account. It could include attributes like `accountNumber`, `balance`, and methods like `deposit` and `withdraw`.

6. **User-Defined Classes:** In OOP languages like C++, UDTs are often implemented as user-defined classes. These classes have member variables (attributes) and member functions (methods).

7. **Organization**: UDTs help organize complex programs by grouping related data and functions together. This improves code organization and makes it easier to understand and maintain.

8. **Data Abstraction:** UDTs allow you to define abstract data types that may have specific rules and constraints associated with them. For example, a `Date` UDT may enforce rules for valid date values.

9. **Improved Readability**: Well-named UDTs enhance code readability and make the program more self-explanatory.

10. **Examples in Other Languages:** While OOP languages use classes for UDTs, other programming languages may use struct-like constructs, records, or custom-defined types to achieve similar goals.

3.Describe use of scope resolution operator with example.

The scope resolution operator (::) in C++ is used to access members (variables or functions) of a class or namespace. It allows you to specify which class or namespace a particular member belongs to, resolving naming conflicts and providing access to members that might otherwise be hidden. Here's how the scope resolution operator is used, along with an example:

Syntax:

```
namespace_name::member_name; // For accessing a member of a namespace
class_name::member_name;    // For accessing a member of a class
```

Using the scope resolution operator is essential when there are naming conflicts or when you want to access members of a specific namespace or class. It helps disambiguate and specify which member you intend to use.

4.Define constructors and it's type.

Concepts of Constructors
Constructor is a special function used to initialize class data members or
we can say constructor is used to initialize the object of class.

Characteristics of a constructor.
• Constructor name class name must be same.
• Constructor doesn't return value.
• Constructor is invoked automatically, when the object of class is
Created.

Types of Constructor
• Default Constructor
• Parameterize Constructor
• Copy Constructor

Default Constructor:
Construct without parameter is called default constructor**.**

Parameterize Constructor:
Constructor with parameter is called parameterize constructor.
In parameterize constructor, we have to pass values to the constructor through object.

Copy Constructor:
Initialization of an object through another object is called copy constructor. In other words, copying the values of one object into another object is called copy constructor.

5.Explain concept of pointer with example.

A pointer in C++ is a variable that holds the memory address of another variable. Pointers are a powerful feature that allows you to work with memory locations and manipulate data indirectly. They are often used for tasks like dynamic memory allocation, passing parameters to functions by reference, and working with data structures. Let's explain the concept of pointers with an example:

Example: Using Pointers

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

int main() {
    int x = 10; // Declare and initialize an integer variable
    int* ptr;   // Declare a pointer to an integer

    // Assign the memory address of 'x' to 'ptr'
    ptr = &x;

    // Access the value of 'x' indirectly using the pointer
    cout << "Value of x: " << x << endl;
    cout << "Value of x using pointer: " << *ptr << endl;

    // Modify the value of 'x' using the pointer
    *ptr = 20;
    cout << "Modified value of x: " << x << endl;

    return 0;
}
```

pointers allow you to:
- Access the memory address of a variable (`&x` gives the address of `x`).
- Access the value of a variable indirectly through a pointer (`*ptr` gives the value of `x`).
- Modify the value of a variable through a pointer (`*ptr = 20;` changes the value of `x`).

Pointers are especially useful in scenarios like dynamic memory allocation, passing parameters by reference, and working with arrays and data structures where direct memory manipulation is required.

6.Explain file modes used to perform file operations.

In C++, file modes are used to specify the type of file operations you want to perform when opening a file. The file mode is an optional parameter when opening a file using the `fstream`, `ifstream`, or `ofstream` classes. Each mode has a specific purpose and controls how the file can be accessed. There are three primary file modes:

1. **ios::in (Input Mode):**
   - This mode is used for reading data from a file.
   - It allows you to read data from the file but does not permit writing to the file.
   - If the file doesn't exist, attempting to open it in input mode will result in an error.
   - Example: `ifstream inputFile("example.txt", ios::in);`

2. **ios::out (Output Mode):**
   - This mode is used for writing data to a file.
   - It allows you to write data to the file but does not permit reading from the file.
   - If the file already exists, it will be truncated (its contents will be deleted) when opened in output mode.
   - If the file doesn't exist, a new file with the specified name will be created.
   - Example: `ofstream outputFile("output.txt", ios::out);`

3. **ios::app (Append Mode):**
   - This mode is used for appending data to an existing file.
   - It allows you to write data to the end of the file without affecting its existing contents.
   - If the file doesn't exist, a new file with the specified name will be created.
   - Example: `ofstream appendFile("data.log", ios::app);`

Additionally, you can combine these modes using the bitwise OR operator (`|`) to achieve different combinations of read, write, and append capabilities. For example:

- `ios::in | ios::out` allows both reading from and writing to the file.

- `ios::in | ios::app` allows reading from the file and appending to it.

7.List all stream classes used in stream operation.

In C++, there are various stream classes used for input and output operations. Here is a list of commonly used stream classes for stream operations:

Input Stream Classes (`istream`):
1. `istream`: The base class for input streams, representing standard input.
2. `ifstream`: Used for reading data from files.
3. `istringstream`: Used for reading data from strings as if they were input streams.
4. `wistream`: The wide-character version of `istream` for wide character input.
5. `wifstream`: The wide-character version of `ifstream` for wide character file input.
6. `wistringstream`: The wide-character version of `istringstream` for wide character string input.

Output Stream Classes (`ostream`):
1. `ostream`: The base class for output streams, representing standard output.
2. `ofstream`: Used for writing data to files.
3. `ostringstream`: Used for writing data to strings as if they were output streams.
4. `wostream`: The wide-character version of `ostream` for wide character output.
5. `wofstream`: The wide-character version of `ofstream` for wide character file output.
6. `wostringstream`: The wide-character version of `ostringstream` for wide character string output.

Bidirectional Stream Classes (`iostream`):
1. `iostream`: The base class for bidirectional streams, representing standard I/O (both input and output).
2. `fstream`: Used for reading and writing data to files.
3. `strstream`: Used for reading and writing data to character arrays as bidirectional streams (deprecated in C++11 and later).

Wide Character Stream Classes (Wide Character I/O):
1. `wcin`: The wide-character version of `cin`, representing standard wide character input.
2. `wcout`: The wide-character version of `cout`, representing standard wide character output.
3. `wcerr`: The wide-character version of `cerr`, representing standard wide character error output.
4. `wclog`: The wide-character version of `clog`, representing standard wide character log output.

These stream classes are part of the C++ Standard Library and provide a standardized way to perform various input and output operations, whether it's reading/writing to/from files, working with strings, or handling wide character data. Depending on your specific needs, you can choose the appropriate stream class for your I/O operations.

Summer-22
2 Marks Questions

a) List two memory management operators available in C++. Also state its use.

In C++, two memory management operators are:

1. new operator:

Use: The `new` operator is used for dynamic memory allocation. It allocates memory for an object or an array of objects on the heap and returns a pointer to the allocated memory.

Example:
```
int* p = new int; // Allocate memory for an integer
```

2. delete operator:

Use: The `delete` operator is used to deallocate memory that was previously allocated with the `new` operator. It releases the memory on the heap to prevent memory leaks.

Example:
```
int* p = new int;  // Allocate memory
delete p;          // Deallocate memory when it's no longer needed
```

These operators are crucial for managing memory dynamically, especially for creating and managing objects whose lifetimes are not limited to a specific scope. It's important to use them carefully to prevent memory leaks and avoid accessing deallocated memory, which can lead to undefined behavior.

b) List c++ stream classes along with their function. (any two

classes)

In C++, there are several stream classes, but I'll provide information on two commonly used stream classes along with some of their functions:

1. iostream class:

- cin : This is an object of the `istream` class, and it is used to read input from the standard input (usually the keyboard).
- cout : This is an object of the `ostream` class, and it is used to display output to the standard output (usually the console).
- **Functions:**
- cin >> variable`: Reads input from the user and stores it in the specified variable.
- cout << "text"`: Displays text or the value of a variable to the console.

2. fstream class:

This class is used to work with files, including reading from and writing to files.

- ifstream : An object of the `ifstream` class, used for reading from files.
- ofstream : An object of the `ofstream` class, used for writing to files.
- fstream : An object of the `fstream` class, used for both reading from and writing to files.
- **Functions:**
- open("filename") : Opens a file with the specified filename.
- close() : Closes the file.
- >> : Reads data from the file, similar to `cin`.
- << : Writes data to the file, similar to `cout`.
- eof() : Checks for the end of the file.

These are some of the commonly used stream classes and their functions for basic input and output operations in C++.

c) Write any four applications of OOP.

Object-Oriented Programming (OOP) is a programming paradigm that provides a structured and efficient way to design and build software. It offers several advantages and is used in various applications. Here are four common applications of OOP:

1. Software Development: OOP is widely used in software development to create applications and systems. It promotes modularity, code reusability, and a more organized approach to design and development. Software developed using OOP is often easier to understand, maintain, and expand.

2. Game Development: OOP is a popular choice in the game development industry. Game objects and characters can be modeled as objects with properties and behaviors. Game engines and frameworks often use OOP principles for designing game logic and interactions.

3. Graphical User Interface (GUI) Development: Building graphical user interfaces for desktop and web applications benefits from OOP. GUI elements, such as windows, buttons, and menus, can be represented as objects. This makes it easier to design and manage complex user interfaces, enabling event-driven programming and interactive applications.

4. Database Management Systems (DBMS): Many modern database management systems, such as Oracle, MySQL, and MongoDB, incorporate OOP concepts. Data entities are represented as objects with attributes and methods. This approach simplifies data modeling, storage, and retrieval.

These are just a few examples of how OOP is applied in various domains. OOP's versatility and ability to model real-world entities and interactions make it a valuable tool in software development and system design.

d) Define constructor. List types of constructor

A constructor in C++ is a special member function that is automatically called when an object of a class is created. Its primary purpose is to initialize the data members of the class and set up the object for use. Constructors have the same name as the class, and they do not have a return type, not even `void`. Constructors can be called explicitly but are typically invoked automatically when an object is created.

Here are the types of constructors in C++:

1. Default Constructor: A default constructor is a constructor with no parameters. It is automatically called when an object is created with no arguments. If you don't define any constructor for a class, the compiler generates a default constructor for you. It initializes the data members to their default values (e.g., zero for integers).

```
MyClass()
{
    // Default constructor code
}
```

2. Parameterized Constructor: A parameterized constructor takes one or more arguments. It allows you to provide initial values for the data members when the object is created.

```
MyClass(int x, double y)
{
    // Constructor code
}
```

3. Copy Constructor: The copy constructor creates a new object by copying the values from an existing object. It is called when an object is passed by value to a function or when an object is explicitly created as a copy of another object. By default, C++ generates a copy constructor for you, which performs a shallow copy (copying the values of data members). You can override this default behavior with a user-defined copy constructor.

```
MyClass(const MyClass& other)
{
    // Copy constructor code
}
```

4. Constructor Overloading: You can define multiple constructors within a class, each with a different parameter list. This is called constructor overloading. It allows you to create objects in various ways by providing different sets of arguments.

```
MyClass(int x)
{
    // Constructor code
}

MyClass(int x, double y)
{
    // Constructor code
}
```

5. Destructor: While not a type of constructor, it's worth mentioning the destructor. A destructor is a special member function called when an object goes out of scope or is explicitly deleted. It is used to clean up resources (e.g., memory) allocated by the object. Like constructors, the destructor has the same name as the class but is preceded by a tilde (`~`).

```
  ~MyClass()
  {
     // Destructor code
  }
```

These constructors provide flexibility in creating and initializing objects based on different requirements.

e) Explain ios app and ios:: in flags

In C++, ios is a base class that defines the basic properties and behaviors of input and output streams. It's part of the Standard Template Library (STL) and is used for various input and output operations. The ios class provides various flags, functions, and member variables for stream operations.

ios:: :- it is used as a scope resolution operator to access the flags and functions defined within the ios class. It allows you to set or modify the state and formatting of input and output streams. Here are some common flags and their meanings:

ios::in: This flag is used for input operations. When it's set, the stream can be used for reading data from a source (e.g., a file or keyboard input).

ios::app: This flag is used when opening a file for output. It specifies that data should be appended to the end of the file rather than overwriting the existing content. It's commonly used when you want to add data to an existing file without losing the previous content.

f) Write any two characteristics of friend function.

Two characteristics of friend functions in C++ are:

1. Access to Private Members: A friend function can access and modify the private and protected members of the class it is a friend of. This provides a way to break the encapsulation of a class temporarily, allowing external functions to work with the class's private data.

2. Declared Outside the Class: Friend functions are declared outside the class in which they are defined as friends. They are not members of the class, but they are associated with it. This means that they can be used to provide external functions that have privileged access to the class without being part of it.

These characteristics make friend functions useful for scenarios where you need external functions to interact with a class's private members while still maintaining data integrity and access control. However, it's important to use friend functions judiciously, as they can potentially violate the encapsulation principle if overused.

g) State the need of virtual function in C++.

The need for virtual functions in C++ is primarily driven by the following objectives and requirements:

1. Polymorphism: Virtual functions play a crucial role in achieving runtime polymorphism in C++. Polymorphism allows objects of different classes to be treated as objects of a common base class. This is essential for designing flexible and extensible code. With virtual functions, you can invoke methods of derived classes through pointers or references to base class objects, and the appropriate method is called at runtime based on the actual derived class type. This enables dynamic dispatch and ensures the correct method is executed, making it easier to work with diverse objects in a unified manner.

2. Extensibility and Maintainability: Virtual functions are vital for designing and maintaining large software systems. They support the "open-closed" principle of object-oriented design, which encourages the addition of new classes or derived types without altering existing code. By overriding virtual functions in derived classes, you can extend the behavior of a base class while leaving the base class itself unchanged. This leads to more maintainable and extensible code.

3. Base Class Abstraction: Virtual functions help to create abstract base classes, which are classes that cannot be instantiated directly but serve as a blueprint for derived classes. These abstract classes can contain pure virtual functions, making them abstract in nature. Subclasses must implement these pure virtual functions, ensuring that essential behavior is defined for each derived class.

4. Interfaces and Frameworks: Virtual functions enable the creation of interfaces and frameworks in C++. An interface defines a contract of methods that must be implemented by any class that conforms to the interface. In C++, interfaces are often created using abstract base classes with pure virtual functions. Frameworks leverage virtual functions to allow customization of their behavior by derived classes.

5. Method Overriding: Virtual functions provide a mechanism for method overriding. Derived classes can provide their implementations of a virtual function, thereby customizing the behavior inherited from a base class. This is critical for creating specialized subclasses that cater to specific requirements while maintaining the interface provided by the base class.

In summary, virtual functions are fundamental to object-oriented programming in C++ as they facilitate polymorphism, maintainability, extensibility, and the creation of abstract classes, interfaces, and frameworks. They are essential for designing flexible and robust software systems.

Winter-19
4 Marks Questions

1.Describe memory allocation for objects.

The memory space for object is allocated when they are declared and not when the class is specified. The member functions are created and placed in memory space only once when they are defined as a part of a class definition. Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions. When the objects are created only space for member variable is allocated separately for each object. Separate memory locations for the objects are essential because the member variables will hold different data values for different objects.

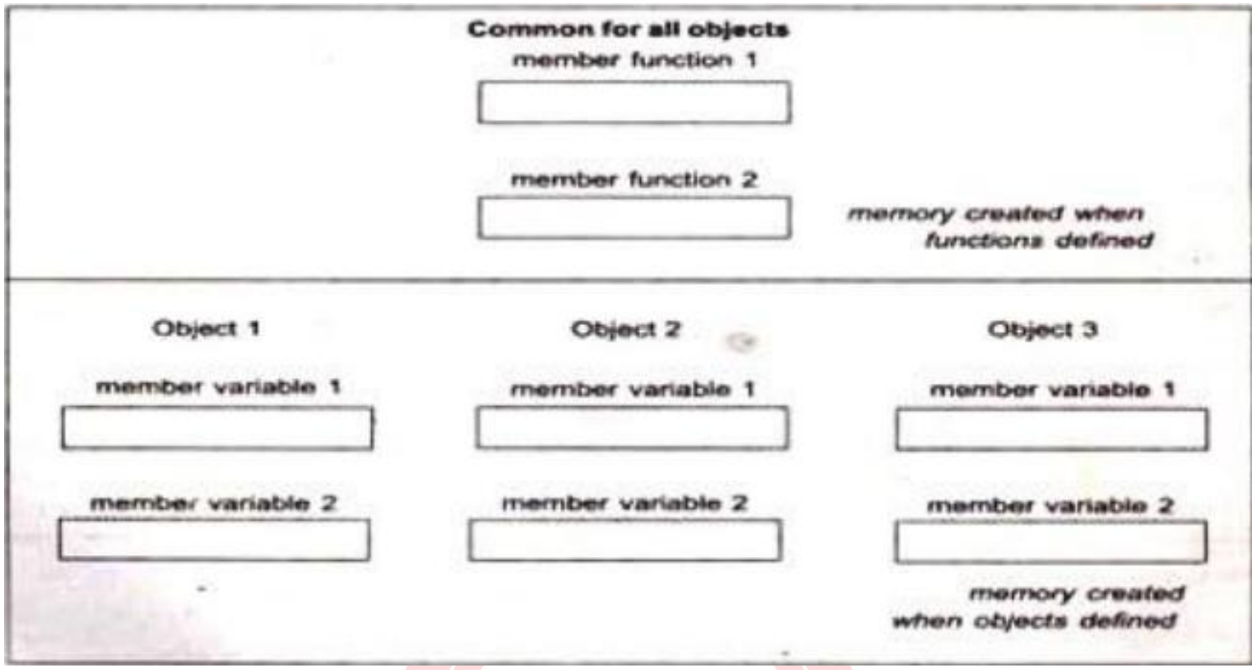


Fig: Memory allocation for objects

2.Write a program to implement single inheritance from the following Refer Figure No.1

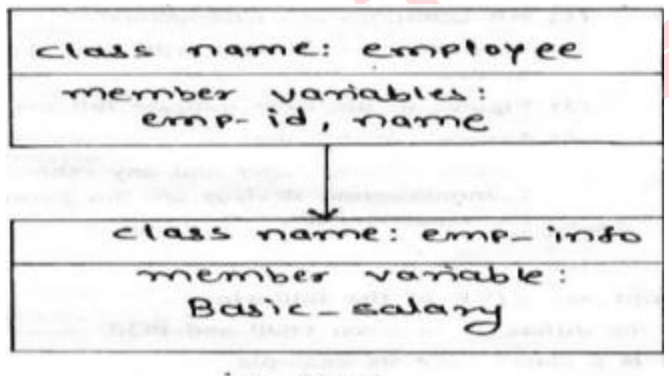


Fig. No. 1

(Note: Any other correct logic shall be considered)

```
#include <iostream.h>
#include <conio.h>
class employee
{
```

```
protected:
int emp_id;
char name[10];
};

class emp_info:public employee
{
int basic_salary;
public:
void getdata()
{
cout<<"Enter emp id";
cin>>emp_id;
cout<<"Enter name";
cin>>name;
cout<<"Enter basic salary";
cin>>basic_salary;
}
void putdata()
{
cout<<"\nEmp_id="<<emp_id;
cout<<"\nName="<<name;
cout<<"\nBasic Salary="<<basic_salary;
}
};
void main()
{
emp_info e;
clrscr();
e.getdata();
e.putdata();
getch();
}
```

```
Enter emp id: 123
Enter name: John
Enter basic salary: 50000

Emp_id=123
Name=John
Basic Salary=50000
```

3.Write any four benefits of OOP. Benefits of OOP:

1. We can eliminate redundant code and extend the use of existing classes.
2. We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch. This leads to saving of development time and higher productivity.
3. The principle of data hiding helps the programmer to build secure programs that cannot be invaded by code in other parts of the program.
4. It is possible to have multiple instances of an object to co-exist without any interference.
5. It is possible to map objects in the problem domain to those in the program.
6. It is easy to partition the work in a project based on objects.
7. The data-centered design approach enables us to capture more details of a model in implementable form.
8. Object-oriented systems can be easily upgraded from small to large systems.
9. Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.
10. Software complexity can be easily managed.

4.Describe 'this' pointer with an example.
'this' pointer:

C++ uses a unique keyword called „this" to represent an object that invokes a member function. This unique pointer is automatically passed to a member function when it is invoked. „this" is a pointer that always point to the object for which the member function was called.
For example, the function call A.max ( ) will set the pointer „this" to the address of the object A. Then suppose we call B.max ( ), the pointer „this" will store address of object B.
Example:

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

class sample {
   int a;
public:
   void setdata(int x) {
      this->a = x;
   }
   void putdata() {
        cout << a;
    }
};

int main() {
    sample s;
    s.setdata(100);
    s.putdata();
    return 0;
}
```

```
100
```

In the above example, this pointer is used to represent object s when setdata ( ) and putdata ( ) functions are called.'

5.Write the applications of object oriented programming.

Applications of object oriented programming are:
1) Real time systems
2) Simulation and modeling
3) Object-oriented databases
4) Hypertext, hypermedia and expertext
5) AI and expert systems
6) Neural networks and parallel programming
7) Decision support and office automation systems
8) CIM/CAM/CAD systems

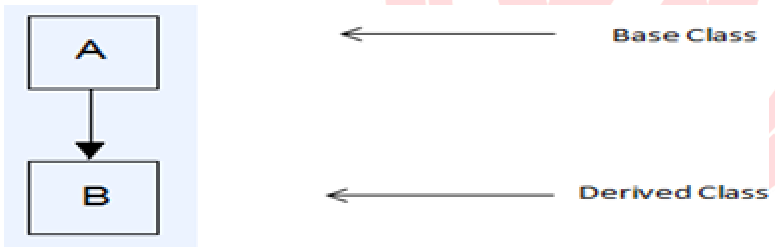
6.State the rules for writing destructor function.

Rules for writing destructor function are:
1) A destructor is a special member function which should destroy the objects that have been created by constructor.
2) Name of destructor and name of the class should be same.
3) Destructor name should be preceded with tilde (~) symbol.
4) Destructor should not accept any parameters.
5) Destructor should not return any value.
6) Destructor should not be classified in any types.
7) A class can have at most one destructor.

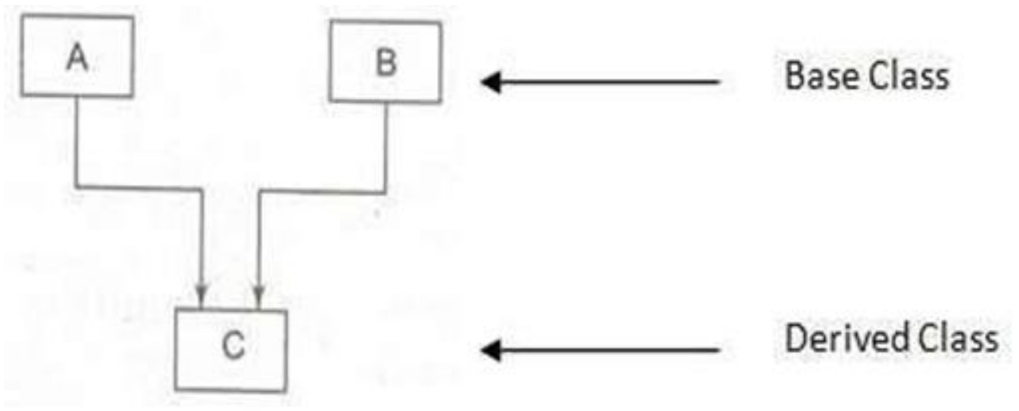
7. What is inheritance? Give different types of inheritance.
4M

Inheritance:
The mechanism of deriving a new class from an old/existing class is called inheritance.
OR
Inheritance is the process by which objects of one class acquired the properties of objects of another class.
Syntax:
```
class derived_class_name : visibility_mode base_class_name
{
//
-----// members of derived class
          //
};
```

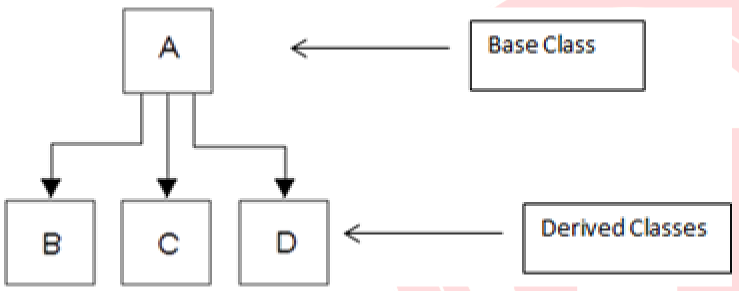
Types of inheritance:
1) **Single inheritance**: In single inheritance, a derived class is derived from only one base class.
Diagram:



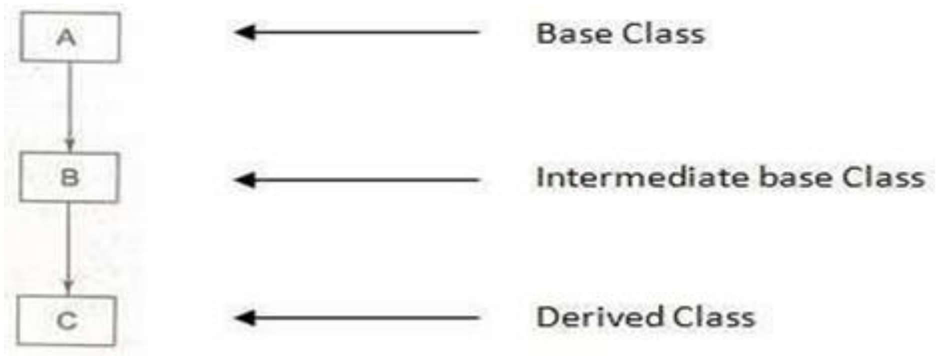
2) **Multiple inheritance:** In multiple inheritance, derived class is derived from more than one base classes.
Diagram:



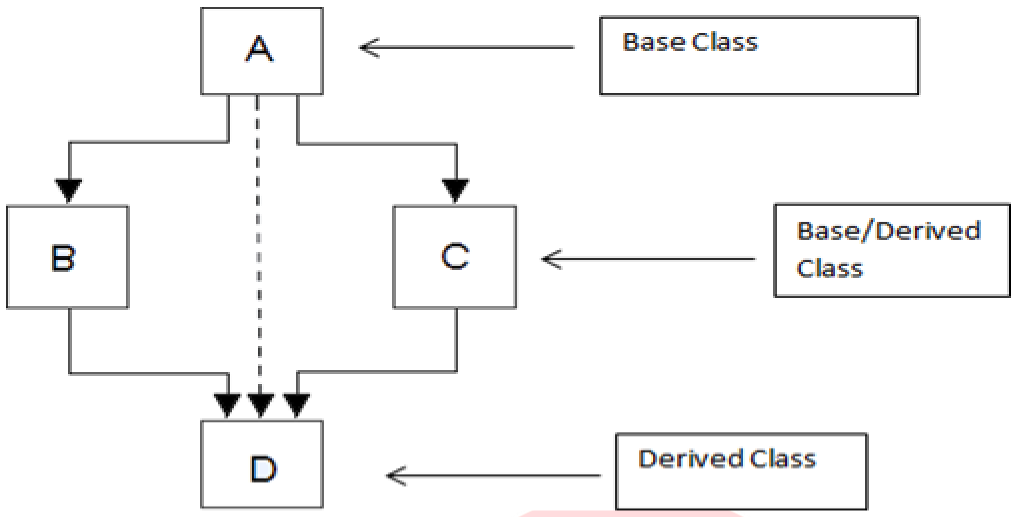
3) **Hierarchical inheritance:** In hierarchical inheritance, more than one derived class is derived from a single class.
Diagram:



4) **Multilevel inheritance**: In multilevel inheritance, a derived class is derived from a derived class (intermediate base class) which in turn is derived from a single base class.
Diagram:



5) **Hybrid inheritance:** Hybrid inheritance is a combination of single, multiple, multilevel and hierarchical inheritance.
Diagram:



8.What are the rules for virtual function? Rules for virtual function: 4M

1. The virtual functions must be members of some class.
2. They cannot be static members.
3. They are accessed by using object pointers.
4. A virtual function can be a friend of another class.
5. A virtual function in a base class must be defined, even though it may not be used.
6. The prototypes of the base class version of a virtual function and all the derived class versions must be identical.
7. We cannot have virtual constructors, but we can have virtual destructors.
8. While a base pointer can point to any type of the derived object, the reverse is not true.
9. When a base pointer points to a derived class, incrementing ordecrementing it will not make it to point to the next object of the derived class.
10. If a virtual function is defined in the base class, it need not be necessarily redefined in the derived class

9.What is parameterized constructor?

A constructor that accepts parameters is called as parameterized constructor.
In some applications, it may be necessary to initialize the various data

members of different objects with different values when they are created. Parameterized constructor is used to achieve this by passing arguments to the constructor function when the objects are created.

Example:

```
class ABC
{
int m;
public:
ABC(int x)
{
m=x;
}
void put()
{
cout<<m;
}
};
void main()
{
ABC obj(10);
obj.put();
}
```

```
10
```

In the above example, constructor ABC (int x) is a parameterized constructor function that accepts one parameter. When ‘obj’ object is created for class ABC, parameterized constructor will invoke and data member m will be initialized with the value 10 which is passed as an argument. Member function put ( ) displays the value of data member ‘m’.

10.Write a program to sort an 1-d array in ascending order. (Note: Any other correct logic shall be considered)

```
#include<iostream.h>
#include<conio.h>
void main()
{
int arr[20];
int i, j, temp,n;
clrscr();
cout<<"\n Enter the array size:";
cin>>n;
cout<<"\n Enter array elements:";
for(i=0;i<n;i++)
{
 cin>>arr[i];
}
for(i=0;i<n;i++)
{
 for(j=i+1;j<n;j++)
{
 if(arr[i]>arr[j])
{
 temp=arr[i];
 arr[i]=arr[j];
 arr[j]=temp;
 }
}
}
cout<<"Sorted Array:";
for(i=0;i<n;i++)
{
 cout<<"\n"<<arr[i];
}
getch();
}
```

```
Enter the array size: [You will enter the size of the array]
Enter array elements: [You will enter the elements of the array]
Sorted Array:
[The sorted array elements will be displayed here]
```

11.Explain the friend function with proper example.

Friend function:

The private members of a class cannot be accessed from outside the class but in some situations two classes may need access of each other‟s private data. So a common function can be declared which can be made friend of more than one class to access the private data of more than one class. The common function is made friendly with all those classes whose private data need to be shared in that function. This common function is called as friend function. Friend function is not in the scope of the class in which it is declared. It is called without any object. The class members are accessed with the object name and dot membership operator inside the friend function. It accepts objects as arguments.

Example:
Program to interchange values of two integer numbers using friend function.

```
#include <iostream.h>
#include <conio.h>
class B;
class A
{
int x;
public:
void accept()
{
cout<<"\n Enter the value for x:";
cin>>x;
}
friend void swap(A,B);
};
class B
{
int y;
public:
void accept()
{
cout<<"\n Enter the value for y:";
cin>>y;
}
friend void swap(A,B);
};
void swap(A a,B b)
{
cout<<"\n Before swapping:";
cout<<"\n Value for x="<<a.x;
cout<<"\n Value for y="<<b.y;
int temp;
temp=a.x;
a.x=b.y;
b.y=temp;
cout<<"\n After swapping:";
cout<<"\n Value for x="<<a.x;
cout<<"\n Value for y="<<b.y;
}
void main()
{
A a;
B b;
clrscr();
a.accept();
b.accept();
swap(a,b);
getch();
}
```

```
Enter the value for x: [You will enter a value for x]
Enter the value for y: [You will enter a value for y]
Before swapping:
Value for x = [The value you entered for x]
Value for y = [The value you entered for y]
After swapping:
Value for x = [The value you entered for y]
Value for y = [The value you entered for x]
```

12.Write a program to count the number of lines in file.
(Note: Any other correct logic shall be considered)

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>

void main() {
   ifstream file;
   char ch;
   int n = 0;
   clrscr();
   file.open("abc.txt");

   while (file) {
       file.get(ch);
       if (ch == '\n') {
           n++;
       }
   }

   cout << "\nNumber of lines in the file are: " << n;
   file.close();
   getch();
}
```

```
Number of lines in the file are: X
```

Summer-19
4 Marks Questions

1.Write a ‘C++’ program to find factorial of given number using loop.
(Note: Any other correct logic shall be considered)

```
#include<iostream.h>
#include<conio.h>
void main()
{
int no,fact=1,i;
clrscr();
cout<<"Enter number:";
cin>>no;
for(i=1;i<=no;i++)
{
fact=fact*i;
}
cout<<"Factorial ="<<fact;
getch();
}
```

```
Enter number: 5
Factorial = 120
```

2.Write a C++ program to declare a class COLLEGE with members as college code. Derive a new class as STUDENT with members as studid. Accept and display details of student along with college for one object of student.
(Note: Any other correct logic shall be considered)

```
#include
#include
class COLLEGE
{
protected:
int collegecode;
};

class STUDENT:public COLLEGE
{
int studid;
public:
void accept()
{
cout<<"Enter college code:";
cin>>collegecode;
cout<<"Enter student id";
cin>>studid;
}
void display()
{
cout<<"College code:"<<collegecode<<endl;
cout<<"Student id:"<<studid;
}
};
void main()
{
STUDENT s;
clrscr();
s.accept();
s.display();
getch();
}
```

```
Enter college code: 101
Enter student id: 12345
College code: 101
Student id: 12345
```

3.Write a C++ program to find smallest number from two numbers using friend function. (Hint: use two classes).
(Note: Any other correct logic shall be considered)

```
#include
#include
class class2;
class class1
{
int no1;
public:
void get1()
{
cout<<"Enter number 1:";
cin>>no1;
}
friend void smallest(class1 no1,class2 no2);
};
class class2
{
int no2;
public:
void get2()
{
cout<<"Enter number 2:";
cin>>no2;
}
friend void smallest(class1 no1,class2 no2);
};
void smallest(class1 c1,class2 c2)
```

```
{
if(c1.no1<c2.no2)
cout<<"no1 is smallest";
else
cout<<"no2 is smallest";
}
void main()
{
class1 c1;
class2 c2;
clrscr();
c1.get1();
c2.get2();
smallest(c1,c2);
getch();
}
```

```
Enter number 1: 5
Enter number 2: 3
no2 is smallest
```

4.Differentiate between run time and compile time polymorphism.

	Compile time polymorphism	Runtime polymorphism
	In this polymorphism, an object is bound to its function call at compile time	In this polymorphism, selection of appropriate function is done at run time.
	Functions to be called are known well before	Function to be called is unknown until appropriate selection is made.
	This does not require use of pointers to objects	This requires use of pointers to object
	Function calls execution are faster	Function calls execution are slower

	It is implemented with operator overloading or function overloading	It is implemented with virtual function.

5.Write a C++ program to create a class STUDENT The data members of STUDENT class.
Roll_No
Name
Marks
(Note: Accepting and displaying data functions is optional).

```
#include <iostream.h>
#include <conio.h>
class STUDENT
 {
int Roll_No;
char Name[20];
float Marks;
};
```

OR

```
#include <iostream.h>
#include <conio.h>
class STUDENT
 {
int Roll_No;
char Name[20];
float Marks;
public:
void Accept();
 void Display();
};
void STUDENT::Accept()
{
cout<<"\nEnter data of student:";
cout<<"\nRoll number:";
cin>>Roll_No;
cout<<"\nName:";
cin>>Name;
cout<<"\nMarks:";
cin>>Marks;
}
void STUDENT::Display()
{
cout<<"\nStudents data is:";
cout<<"\nRoll number:"<<Roll_No;
cout<<"\nName:"<<Name;
cout<<"\nMarks:"<<Marks;
}
void main()
{
STUDENT S[5];
int i;
clrscr();
for(i=0;i<5;i++)
{
 S[i].Accept();
}
for(i=0;i<5;i++)
{
 S[i].Display();
}
getch();
}
```

```
Enter data of student:
Roll number: 101
Name: Student1
Marks: 95

Enter data of student:
Roll number: 102
Name: Student2
Marks: 88

Enter data of student:
Roll number: 103
Name: Student3
Marks: 76

Enter data of student:
```

```
Roll number: 104
Name: Student4
Marks: 89

Enter data of student:
Roll number: 105
Name: Student5
Marks: 92
```

6.Accept data for five students and display it. Write a C++ program to disply a sum of array elements of array size n.
(Note: Any other correct logic shall be considered)

```
#include
#include
void main()
{
int arr[20],i,n,sum=0;
clrscr();
cout<<"\nEnter size of an array:";
cin>>n;
cout<<"\nEnter the elements of an array:";
for(i=0;i<n;i++)
{
cin>>arr[i];
}
for(i=0;i<n;i++)
{
 sum=sum+arr[i];
}
cout<<"\nArray elements are:";
for(i=0;i<n;i++)
{
cout<<arr[i]<<" ";
}
cout<<"\nSum of array elements is:"<<sum;
getch();
}
```

```
Enter size of an array: 5
Enter the elements of an array: 10 20 30 40 50
Array elements are: 10 20 30 40 50
Sum of array elements is: 150
```

7.Describe with examples, passing parameters to base class constructor and derived class constructor by creating object of derived class.

When a class is declared, a constructor can be declared inside the class to initialize data members. When a base class contains a constructor with one or more arguments then it is mandatory for the derived class to have a constructor and pass arguments to the base class constructor. When both the derived and base classes contain constructors, the base constructor is executed first and then the constructor in the derived class is executed. The constructor of derived class receives the entire list of values as its arguments and passes them on to the base constructors in the order in which they are declared in the derived class.

General form to declare derived class constructor:
Derived-constructor (arglist1, arglist (D)):Base1(arglist1)
{ Body of derived class constructor }

Derived constructor declaration contains two parts separated with colon (:). First part provides declaration of arguments that are passed to the derived constructor and second part lists the function calls to the base constructors.
Example:

```
#include<iostream.h>
#include<conio.h>
class base
{
int x;
public:
base(int a)
{
x=a;
cout<<"Constructor in base x="<<x<<endl;
}
};
class derived: public base
{
int y;
public:
derived(int a,int b):base(a)
{
y=b;
cout<<"Constructor in derived.y="<<y;
}
};
void main()
{
clrscr();
derived ob(2,3);
getch();
}
```

```
Constructor in base x=2
Constructor in derived. y=3
```

In the above example, base class constructor requires one argument and derived class constructor requires one argument. Derived class constructor accepts two values and passes one value to base class constructor.

8.Describe how memory is allocated to objects of class with suitable diagram.

Description: The memory space for object is allocated when they are declared and not when the class is specified. Actually, the member functions are created and placed in memory space only once when they are defined as a part of a class definition. Since all the objects belonging to that class use the same member functions, no separate space is allocated for member functions. When the objects are created only space for member variable is allocated separately for each object. Separate memory locations for the objects are essential because the member variables will hold different data values for different objects this is shown in fig:

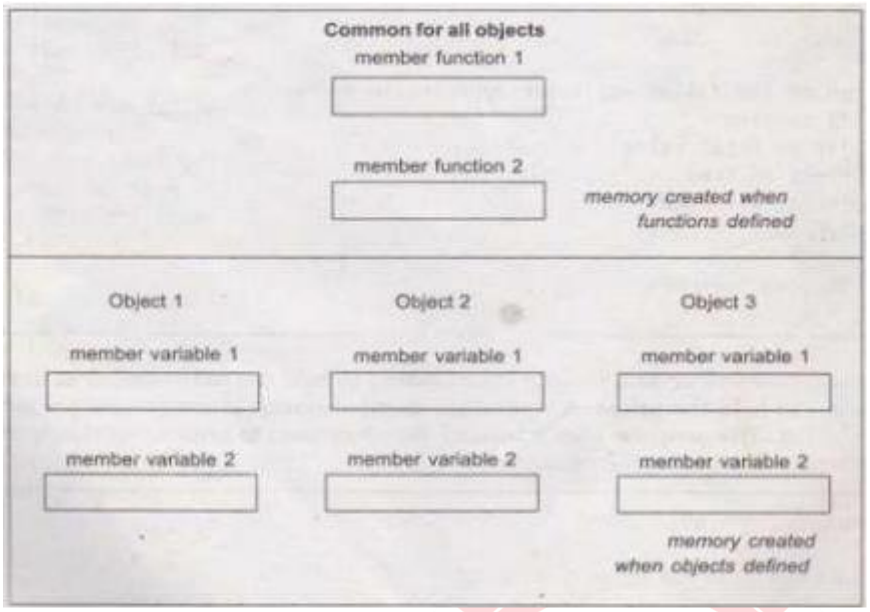


Fig: Memory allocation for objects

9. Write a program to implement multiple inheritance as shown in following Figure :

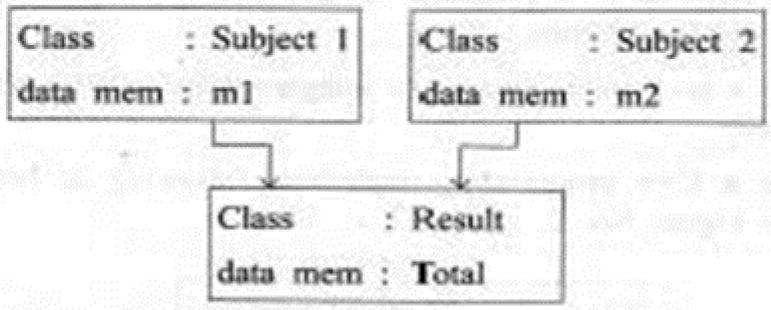


Fig. No. 1

Accept and display data for one object of class result. 4M

```
#include<iostream.h>
#include<conio.h>
class Subject1
{
protected:
float m1;
};

class Subject2
{
protected:
float m2;
};

class Result:public Subject1,public Subject2
{
float Total;
public:
void accept()
{
cout<<"Enter marks of subject1:";
cin>>m1;
cout<<"\nEnter marks of subject2:";
cin>>m2;
}
void calculate()
{
Total=(m1+m2);
}
void display()
{

cout<<"\nSubject 1 marks:"<<m1;
cout<<"\nSubject 2 marks:"<<m2;
cout<<"\nTotal is:"<<Total;
}
};

void main()
{
Result r;
clrscr();
r.accept();
 r.calculate();
r.display();
getch();
}
```

```
Enter marks of subject1: 85
Enter marks of subject2: 90

Subject 1 marks: 85
Subject 2 marks: 90
Total is: 175
```

10.Describe following terms: Inheritance, data abstraction, data encapsulation, dynamic binding.

Inheritance:
1. Inheritance is the process by which objects of one class acquire the properties of objects of another class.
2. It supports the concept of hierarchical classification. It also provides the idea of reusability.

Data abstraction:
1. Data abstraction refers to the act of representing essential features without including the background details or explanations.
2. Classes use the concept of abstraction and are defined as a list of abstract attributes such as size, weight and cost and functions to operate on these attributes.

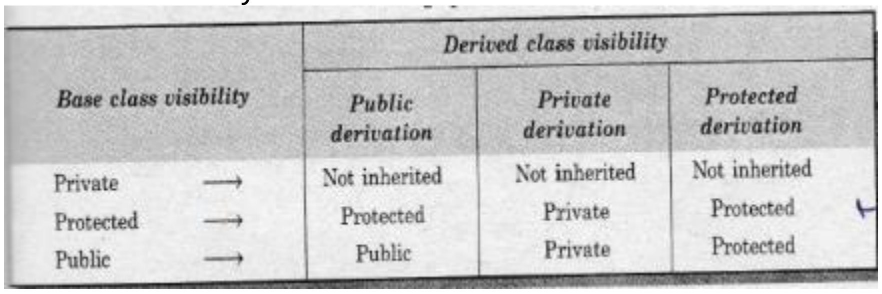
Data encapsulation:
1. The wrapping up of data and functions together into a single unit (called class) is known as encapsulation.
2. By this attribute the data is not accessible to the outside world, and only those functions which are wrapped in the class can access it.

Dynamic Binding:
1. Dynamic binding refers to the linking of a procedure call to be executed in response to the call.
2. It is also known as late binding. It means that the code associated with a given procedure call is not known until the time of the call at run-time.

11.State and describe visibility modes and its effects used in inheritance. (Note: Diagram is optional)

Different visibility modes are:
1. Private
2. Protected
3. Public

Effects of visibility modes in inheritance:



Base class visibility	Derived class visibility		
	Public derivation	Private derivation	Protected derivation
Private →	Not inherited	Not inherited	Not inherited
Protected →	Protected	Private	Protected
Public →	Public	Private	Protected

Private members of base class are not inherited directly in any visibility mode.
1. Private visibility mode In this mode,
protected and public members of base class become private members of derived class.
2. Protected visibility mode In this mode,
protected and public members of base class become protected members of derived class.
3. Public visibility mode In this mode,
protected members of base class become protected members of derived class and public members of base class become public members of derived class

12.Write a C++ program to count number of spaces in text file.
(Note: Any other correct logic shall be considered)

Program:

```
#include <iostream.h>
#include <fstream.h>
#include <conio.h>
void main()
{
ifstream file;
int s=0;
char ch;
clrscr();
file.open("abc.txt");
while(file)
{
file.get(ch);
 if(ch==' ')
 {
 s++;
 }
}
cout<<"\nNumber of spaces in text file are:"<<s;
getch();
}
```

```
Number of spaces in text file are: X
```

13.Differentiate between contractor and destructor.
(Note: Contractor shall be considered as Constructor.)

Constructor	Destructor
A constructor is a special member function whose task is to initialize the objects of its class.	A destructor is a special member function whose task is to destroy the objects that have been created by constructor.
It constructs the values of data members of the class.	It does not construct the values for the data members of the class
It is invoked automatically when the objects are created	It is invoked implicitly by the compiler upon exit of a program/block/function.
Constructors are classified in various types such as : Default constructor Parameterized constructor Copy constructor Overloaded constructor	Destructors are not classified in any types.
A class can have more than one constructor.	A class can have at most one constructor.
Constructor accepts parameter. Also it can have default value for its parameter	Destructor never accepts any parameter.

Syntax:
classname()
{
…
…
}
	Syntax:
	destructor name is preceded with tilde.
	~classname()
	{
	….
	….
	}

Example: ABC() { … }	Example: ~ABC() { … .}

14.
(i) Write any three rules of operator overloading.
(ii) Write a program in C++ to overload unary '_' operator to negate values of data members of class.

(i) Write any three rules of operator overloading.
Rules for overloading operators:
1. Only existing operators can be overloaded. New operators cannot be created.
2. The overloaded operator must have at least one operand that is of user defined data type.
3. We can't change the basic meaning of an operator. That is to say, we can't redefine the plus(+) operator to subtract one value from other.
4. Overloaded operators follow the syntax rules of the original operators. They can't be overridden.
5. There are some operators that can't be overloaded.
6. We can't use friend functions to overload certain operators. However, member function scan be used to overload them.
7. Unary operators overloaded by means of member function take no explicit arguments and return no explicit values, but, those overloaded by means of the friend function, take one reference argument (the object of the relevant class).
8. Binary operators overloaded through a member function, take one explicit argument and those which are overloaded through a friend function take two explicit arguments.
9. When using binary operators overloaded through a member function, the left hand operand must be an object of the relevant class.
10. Binary arithmetic operators such as +,-,* and / must explicitly return a value. They must not attempt to change their own arguments.

(ii) Write a program in C++ to overload unary '_' operator to negate values of data members of class.

(Note: Any other correct logic shall be considered)

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class Number
{
int x, y;
public:
Number (int a,int b)
{
a =x;
b =y;
}
void display()
{
cout<<"value of x=”<<x<<”\nValue of y= ”<<y;
}
void operator - ( )
{
x = - x;
y = - y;
}
};
void main()
{
Number N1(5,6);
clrscr();
N1.display();
-N1;
cout<<"\n After negation:";
N1. display ();
getch();
}
```

```
Value of x=5
Value of y=6
After negation:
Value of x=-5
Value of y=-6
```

15.Write a C++ program to append data from abc.txt to xyz.txt file. (Note: Any other correct logic shall be considered)

Assuming input file as abc.txt with contents "World" and output file named as xyz.txt with contents "Hello" have been already created.

```
#include <iostream.h>
#include<fstream.h>
int main()
{
fstream f;
ifstream fin;
fin.open("abc.txt",ios::in);
ofstream fout;
fout.open("xyz.txt", ios::app);
if (!fin)
 {
 cout<< "file not found";
 }
 else
 {
 fout<<fin.rdbuf();
 }
 char ch;
 f.seekg(0);
 while (f)
 {
 f.get(ch);
 cout<< ch;
 }
 f.close();
 return 0;
}
```

```
Output:
Hello World
```

16.Write a C++ program to declare a class student with members as roll no, name and department. Declare a parameterized constructor with default value for department as ‘CO’ to initialize members of object. Initialize and display data for two students. (Note: Any other relevant logic should be considered)

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class student
{
int roll_no;
char name[20],department[40];
public:
student(int rno,char *n,char *d="CO")
{
roll_no=rno;
strcpy(name,n);
strcpy(department,d);
}
void display()
{
cout<<"\n Roll No:"<<roll_no;
cout<<"\n Name:"<<name;
cout<<"\n Department:"<<department;
}
};
void main()
{
student s1(112," Chitrakshi"),s2(114,"Anjali");
clrscr();
s1.display();
s2.display();
getch();
}
```

```
Roll No: 112
Name: Chitrakshi
Department: CO
Roll No: 114
```

Name: Anjali
Department: CO

Winter-18
4 Marks Questions

1.State any four object oriented languages.

Object oriented programming language:

- C++
- Smalltalk
- Object pascal
- java
- Simula
- Ada
- Turbo pascal
- Eiffel
- C#
- Python

2.Describe use of protected access specifier used in the class.

Protected access specifier is use to declare a class member that is accessible by the member functions within its class and any class immediately derived from it.

3.Differentiate between OOP and POP

Sr. No.	PROCEDURE ORIENTED PROGRAMMING (POP)	OBJECT ORIENTED PROGRAMMING (OOP)
1	Focus is on doing things	Focus is on data rather than

	(procedure).	procedure.
2	Large programs are divided into multiple functions.	Programs are divided into multiple objects.
3	Data move openly around the system from function to function.	Data is hidden and cannot be accessed by external functions.
4	Functions transform data from one form to another by calling each other.	Objects communicate with each other through function.
5	Employs top-down approach in program design.	Employs bottom-up approach in program design
6	Procedure oriented approach is used in C language.	Object oriented approach is used in C++ language.

4.Write any two characteristics of destructor.

Characteristics:
1. It is used to destroy objects created by a constructor.
2. Name of destructor and name of the class is same.
3. Its name is preceded with tilde (~) symbol.
4. It never takes any argument.
5. It does not return any value.
6. It is invoked implicitly by the compiler upon exit from the program (or block or function) i.e when scope of object is over.

5.Describe meaning of the following
(i) ios : : in
(ii) ios : : out

(i) ios : : in : It is a file mode. It is used to open a file in read only mode.
(ii) ios : : out : It is a file mode. It is used to open a file in write only mode.

6.Give output for following code:

```
class student
{
int roll no;
char name [14];
} s[6];
void main()
{
cout<<sizeof(s);
}
```

Considering roll_no(Single variable) the output is: 96
OR
Considering roll, no (Two variables) the output is: 108
OR
Considering roll no the output is: error – space between roll and no

7.Write syntax to define a derived class

Syntax:

```
class derived_class_name: visibility_mode/access_specifier
base_class_name
{
class body
};
```

8.Write a C++ program to accept array of five elements, find and display smallest number from an array.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int a[5],smallest,i;
clrscr();
cout<<" Enter array elements:";
for(i=0;i<5;i++)
cin>>a[i];
smallest=a[0];
for(i=1;i<5;i++)
{
if(a[i]<smallest)
{
smallest=a[i];
}
}
cout<<endl<<"Smallest number="<<smallest;
getch();
}
```

```
Enter array elements: 15 8 24 3 12
Smallest number=3
```

9.Write a C++ program to declare a class 'College' with data members as name and college code. Derive a new class 'student' from the class college with data members as sname and roll no. Accept and display details of one student with college data. 4M

```
#include<iostream.h>
#include<conio.h>
using namespace std;
class college
{
char name[10];
int collegecode;

public:
void getcollege()
{
cout<<"Enter college name:";
cin>>name;
cout<<"Enter college code:";
cin>>collegecode;
}
void putcollege()
{
```

```
cout<<endl<<"College name="<<name;
cout<<endl<<"College code="<<collegecode;
}
};

class student:public college
{
char sname[10];
int rollno;

public:
void getstudent()
{
cout<<"Enter student name";
cin>>sname;
cout<<"Enter roll no:";
cin>>rollno;
}
void putstudent()
{
cout<<endl<<"Student name= "<<sname;
cout<<endl<<"Roll no= "<<rollno;
}
};

void main()
{
student s;
clrscr();
s.getcollege();
s.getstudent();
s.putcollege();
s.putstudent();
getch();
}
```

Output:-

```
Enter college name:johdhale
Enter college code:0034
Enter student name shivani
Enter roll no:30

College name=johdhale
College code=34
Student name= shivani
Roll no= 30
```

10.Write a C++ program to declare a class 'circle' with data members as radius and area. Declare a function getdata to accept radius and putdata to calculate and display area of circle.

```
#include<iostream.h>
#include<conio.h>
class circle
{
float radius,area;
public:
void getdata()
{
cout<<"Enter radius:";
cin>>radius;
}
void putdata()
{
area=3.14*radius*radius;
cout<<"Area of circle="<<area;
}
};
void main()
{
circle c;
clrscr();
c.getdata();
c.putdata();
getch();
}
```

```
Enter radius: 5
Area of circle = 78.5
```

11.With suitable example, describe effect of ++ and - - operators used with pointer in pointer arithmetic.

++ Operator: - It is referred as increment operator that increments the value of variable. If ++ operator is used with pointer variable, then pointer variable points to next memory address that means pointer increment with respect to size of the data type used to declare pointer variable.

Example:-

```
int a[5]={10,20,30,40,50},*ptr;
ptr=a[0];
for(i=0;i<5;i++)
{
cout<<*ptr;
ptr++;
}
```

In the above example, ptr points to memory location of a[0]. Increment statement ptr++ increments ptr by memory size of int i.e 2 bytes and ptr points to a[1].

- - Operator: - It is referred as decrement operator that decrements the value of variable. If - - operator is used with pointer variable, then pointer variable points to previous memory address that means pointer decrement with respect to size of the data type used to declare pointer variable.

Example:-

```
int a[5]={10,20,30,40,50},*ptr; ptr=a[4];
for(i=0;i<5;i++)
{
cout<<*ptr; ptr- -;
}
```

In the above example, ptr points to memory location of a[4]. Decrement statement ptr- - decrements ptr by memory size of int i.e 2 bytes and ptr points to a[3].

12.Write a C++ program to declare a class addition with data members as x and y. Initialize values of x and y with constructor.
Calculate addition and display it using function 'display'.

```
#include <iostream.h>
#include <conio.h>
class addition
{
int x,y;
public:
addition(int,int);
void display();
};
addition::addition (int x1,int y1)
{
x=x1;
y=y1;
}
void addition::display()
{
cout<<"\nAddition of two numbers is:"<<(x+y);
}
void main()
{
addition a(3,4);
a.display();
getch();
}
```

```
Addition of two numbers is: 7
```

13.With suitable diagram describe structure of C++ program.

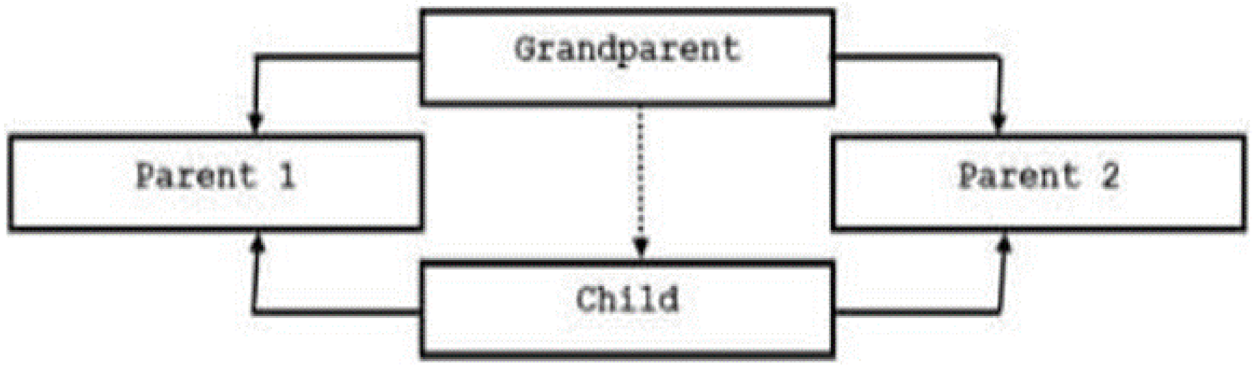
General C++ program has following structure.

INCLUDE HEADER FILES
CLASS DECLARATION
MEMBER FUNCTIONS DEFINITIONS
MAIN FUNCTION PROGRAM

Description:-
1. Include header files
In this section a programmer include all header files which are require to execute given program. The most important file is iostream.h header file. This file defines most of the C++statements like cout and cin. Without this file one cannot load C++ program.
2. Class Declaration
In this section a programmer declares all classes which are necessary for given program. The programmer uses general syntax of creating class.
3. Member Functions Definition
This section allows programmer to design member functions of a class. The programmer can have inside declaration of a function or outside declaration of a function.
4. Main Function Program
In this section programmer creates objects and calls various functions writer within various class.

14.Describe the concept of virtual base class with suitable example.
Note: Program/diagram with syntax shall be considered as an example 4M

Virtual Base Class:
An ancestor class is declared as virtual base class which is used to avoid duplication of inherited members inside child class due to multiple path of inheritance.



Consider a hybrid inheritance as shown in the above diagram. The child class has two direct base classes, Parent1 and Parent2 which themselves have a common base class as Grandparent. The child inherits the members of Grandparent via two separate paths. All the public and protected members of Grandparent are inherited into Child twice, first via Parent1 and again via Parent2. This leads to duplicate sets of the inherited members of Grandparent
inside Child class. The duplication of inherited members can be avoided by making the common base class as virtual base class while declaring the direct or intermediate base classes as shown below.

```
class Grandparent
{
};
class Parent1:virtual public Grandparent
{
};
class Parent2:virtual public Grandparent
{
};
class Child: public Parent1,public Parent2
{
};
```

Example:-

```
#include<iostream.h>
#include<conio.h>
class student
{
int rno;

public:
void getnumber()
{
cout<<"Enter Roll No:";
cin>>rno;
}
void putnumber()
{
cout<<"\n\n\t Roll No:"<<rno<<endl;
}
};

class test: virtual public student
{
public:
int part1,part2;
void getmarks()
{
cout<<"Enter Marks\n";
cout<<"Part1:";
cin>>part1;
cout<<"Part2:";
cin>>part2;
}
void putmarks()
{
cout<<"\t Marks Obtained\n";
cout<<"\n\t Part1:"<<part1;
cout<<"\n\tPart2:"<<part2;
}
};

class sports: public virtual student
{
public:

int score;
void getscore()
{
cout<<"Enter Sports Score:";
cin>>score;
}
void putscore()
{
cout<<"\n\t Sports Score is:"<<score;
}
};

class result: public test, public sports
{
```

```
int total;
public:
void display()
{
total=part1+part2+score;
putnumber();
putmarks();
putscore();
cout<<"\n\t Total Score:"<<total;
}
};

void main()
{
result obj;
clrscr();
obj.getnumber();
obj.getmarks();
obj.getscore();
obj.display();
getch();
}
```

Output:-

```
Enter Roll No:30
Enter Marks
Part1:45
Part2:48
Enter Sports Score:40


        Roll No:30
        Marks Obtained

        Part1:45
        Part2:48
        Sports Score is:40
        Total Score:133
```

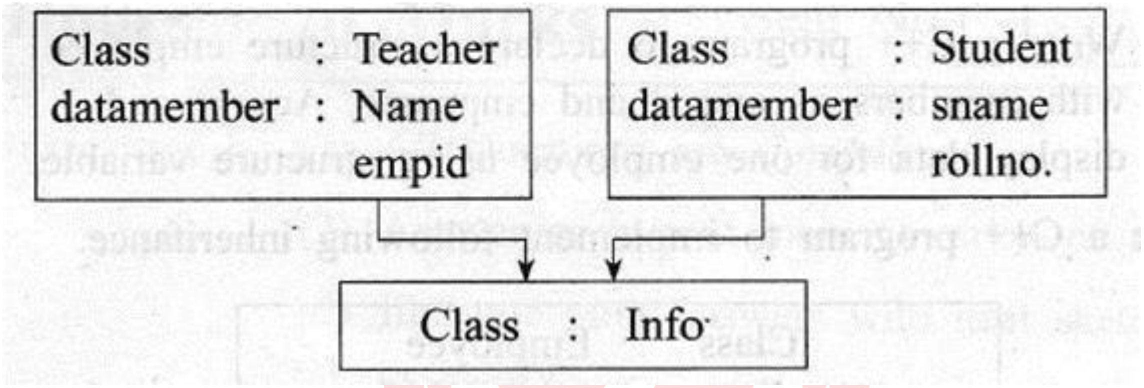
15.Describe use of static data member in C++ with example.

Use of static data member:
1. Static data member is used to maintain values common to the entire class.
2. It is initialized to zero when the first object of its class is created.
3. Only one copy of that member is created for the entire class and is shared by all the objects of that class.

Example:
```
#include <iostream.h>
#include <conio.h>
class test
{
static int count;
int obj_no;
public:
void getdata()
{
obj_no=++count;
cout<<"\n Object number="<<obj_no;
}
static void showcount()
{
cout<<"\n total number of objects="<<count;
}
};
int test::count;
void main()
{
test t1,t2;
clrscr();
t1.getdata();
t2.getdata();
test::showcount();
test t3;
t3.getdata();
test::showcount();
getch();
}
```

```
Object number=1
Object number=2
Total number of objects=2
Object number=3
Total number of objects=3
```

16.Write a C++ program to implement inheritance shown in following figure
4M



Accept and display data of one teacher and one student using object of class 'Info'.

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

class Teacher
{
protected:
char Name[20];
int empid;
};

class Student
{
protected:
char sname[20];
int rollno;
};
```

```
class Info:public Teacher,public Student
{
public:
void acceptT()
{
cout<<"\nEnter data for teacher:";
cout<<"\nName:";
cin>>Name;
cout<<"Employee id:";
cin>>empid;
}
void displayT()
{
cout<<"\nTeacher's data is:";
cout<<"\nName:"<<Name;
cout<<"\nEmployee id:"<<empid;
}
void acceptS()
{
cout<<"\n\nEnter student's data:";
cout<<"\nName:";
cin>>sname;
cout<<"Roll no:";
cin>>rollno;
}
void displayS()
{
cout<<"\nStudent's data is:";
cout<<"\nName:"<<sname;
cout<<"\nRoll no:"<<rollno;
}
};
void main()
{
Info I;
clrscr();
I.acceptT();
I.displayT();
I.acceptS();
I.displayS();
getch();
}
```

```
Enter data for teacher:
Name: [You input the teacher's name]
Employee id: [You input the teacher's employee id]

Teacher's data is:
Name: [Teacher's name]
Employee id: [Teacher's employee id]

Enter student's data:
Name: [You input the student's name]
Roll no: [You input the student's roll number]

Student's data is:
Name: [Student's name]
Roll no: [Student's roll number]
```

17.Write a C++ program to print multiplication table of 7. (example: 7 x 1 ….7 x 10 = 70)

```
#include<iostream.h>
#include<conio.h>
void main()
{
int num;
clrscr();
cout<<"Multiplication table for 7 is:"<<endl;
for(num=1;num<=10;num++)
{
cout<<"7 *"<<num<<"="<<7*num<<endl;
}
getch();
}
```

```
Multiplication table for 7 is:
```

```
7 * 1 = 7
7 * 2 = 14
7 * 3 = 21
7 * 4 = 28
7 * 5 = 35
7 * 6 = 42
7 * 7 = 49
7 * 8 = 56
7 * 9 = 63
7 * 10 = 70
```

18.Write a C++ program to swap two integer numbers and swap two float numbers using function overloading.
(Hint: overload swap function)
Note: Any other relevant logic shall be considered.

```
#include<iostream.h>
#include<conio.h>
void swap(int a,int b)
{
int temp;
temp=a;
 a=b;
 b=temp;
cout<<"\nInteger values after swapping are:"<<a<<" "<<b;
}
void swap(float x,float y)
{
float temp1=x;
 x=y;
 y=temp1;
cout<<"\nFloat values after swapping are:"<<x<<" "<<y;
}
void main()
{
clrscr();
swap(10,20);
swap(10.15f,20.25f);
getch();
}
```

```
Integer values after swapping are: 20 10
Float values after swapping are: 20.25 10.15
```

19.Write a C++ program to count number of spaces present in contents of file.
Note: Any other relevant logic shall be considered

```
#include
#include
#include
void main()
{
ifstream file;
charch;
int s=0;
clrscr();
file.open("abc.txt");
while(file)
{
file.get(ch);
if(ch==' ')
 {
s++;
 }
}
cout<<"\nNumber of spaces present in the content of the given file
are:"<<s;
getch();
}
```

```
Number of spaces present in the content of the given file are: [Number of spaces]
```

20.**Write a C++ program to find greatest number among two numbers from two different classes using friend function.**

```
#include
#include
class second;
class first
{
int x;
public:
void getx()
{
cout<<"\nEnter the value of x:";
cin>>x;
}
friend void max(first,second);
};
class second
{
int y;
public:
void gety()
{
cout<<"\nEnter the value of y:";
cin>>y;
}
friend void max(first,second);
};
void max(first a,second b)
{
if(a.x>b.y)
{
cout<<"\Greater value is:"<<a.x;
}
else
{
cout<<"\nGreater value is:"<<b.y;
}
}
```

```
void main()
{
first a;
second b;
clrscr();
a.getx();
b.gety();
max(a,b);
getch();
}
```

```
Output:
Hello World
```

Winter-22
4 Marks Questions

1.Develop a c++ program to print Fibonacci series.

Certainly! Here's a C++ program that prints the Fibonacci series up to a specified number of terms:

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

int main() {
    int n; // Number of terms in the Fibonacci series

    cout << "Enter the number of terms in the Fibonacci series: ";
    cin >> n;
    int firstTerm = 0, secondTerm = 1, nextTerm;
    cout << "Fibonacci Series up to " << n << " terms: ";
    for (int i = 0; i < n; ++i) {
        if (i <= 1)
            nextTerm = i;
        else {
            nextTerm = firstTerm + secondTerm;
            firstTerm = secondTerm;
            secondTerm = nextTerm;
        }
        cout << nextTerm << " ";
    }
    cout << endl;
    return 0;
}
```

```
Enter the number of terms in the Fibonacci series: 10
Fibonacci Series up to 10 terms: 0 1 1 2 3 5 8 13 21 34
```

The program will display the Fibonacci series up to the specified number of terms as per the user's input.

2.Develop a c++ program for multilevel inheritance 4M

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

// Base class Person
class Person
{
protected:
string name;
int age;

public:
void inputPersonData()
{
 cout << "Enter Name: ";
 cin >> name;
 cout << "Enter Age: ";
 cin >> age;
 }
 void displayPersonData()
{
 cout << "Name: " << name << endl;
 cout << "Age: " << age << endl;
 }
};
// Derived class Student
class Student : public Person
{
protected:
int studentID;

public:
void inputStudentData()
{
 inputPersonData(); // Input Person data
 cout << "Enter Student ID: ";
 cin >> studentID;
}

 void displayStudentData()
{
 displayPersonData(); // Display Person data
 cout << "Student ID: " << studentID << endl;
}
};
// Derived class GraduateStudent (Multilevel inheritance)
class GraduateStudent : public Student
{
private:
string researchTopic;

public:
void inputGraduateStudentData()
{
 inputStudentData(); // Input Student data
 cout << "Enter Research Topic: ";
 cin.ignore();
 getline(cin, researchTopic);
 }
 void displayGraduateStudentData()
{
 displayStudentData(); // Display Student data
 cout << "Research Topic: " << researchTopic << endl;
 }
};

void main()
{
   GraduateStudent gradStudent;

    cout << "Input Graduate Student Data:\n";
    gradStudent.inputGraduateStudentData();

    cout << "\nGraduate Student Data:\n";
    gradStudent.displayGraduateStudentData();
    getch();
}
```

```
Input Graduate Student Data:
Enter Name: John
Enter Age: 25
Enter Student ID: 1234
Enter Research Topic: Artificial Intelligence

Graduate Student Data:
Name: John
Age: 25
Student ID: 1234
Research Topic: Artificial Intelligence
```

3.Develop a c++ program for accept data from user to calculate percentage for 5 subject and display grade according to percentage.

```
#include <iostream>
```

```
using namespace std;

int main() {
    float s1, s2, s3, s4, s5;
    float totalMarks, percentage;

    cout << "Enter marks for Subject 1: ";
    cin >> s1;
    cout << "Enter marks for Subject 2: ";
    cin >> s2;
    cout << "Enter marks for Subject 3: ";
    cin >> s3;
    cout << "Enter marks for Subject 4: ";
    cin >> s4;
    cout << "Enter marks for Subject 5: ";
    cin >> s5;

    // Calculate the total marks
    totalMarks = s1 + s2 + s3 + s4 + s5;

    // Calculate the percentage
    percentage = (totalMarks / 500) * 100;

    // Display the percentage
    cout << "Percentage: " << percentage << "%" << endl;

    // Determine the grade based on the percentage
if (percentage >= 90)
{
        cout << "Grade: A+" << endl;
}
else if (percentage >= 80)
{
     cout << "Grade: A" << endl;
}
else if (percentage >= 70)
{
     cout << "Grade: B" << endl;
}
else if (percentage >= 60)
{
     cout << "Grade: C" << endl;
}
else if (percentage >= 50)
{
    cout << "Grade: D" << endl;
}
else
{
    cout << "Grade: F (Fail)" << endl;
}
    return 0;
}
```

```
Enter marks for Subject 1: 85
Enter marks for Subject 2: 92
Enter marks for Subject 3: 78
Enter marks for Subject 4: 88
Enter marks for Subject 5: 95
Percentage: 87.6%
Grade: A
```

4.Develop c++ program to open and read content of file also write “object oriented” string in file and close it.

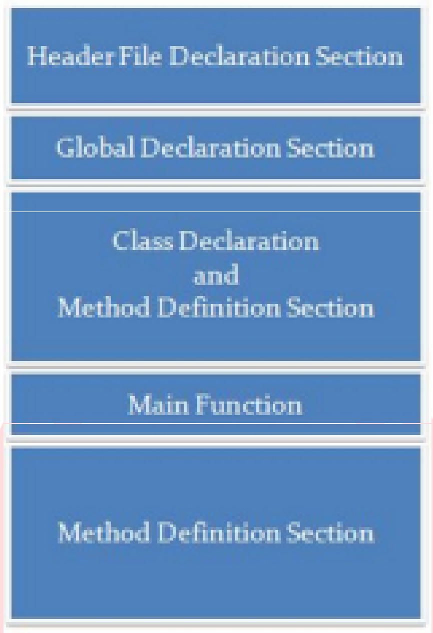
```
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

int main() {
    // Open a file for reading
    ifstream inputFile("sample.txt");

    if (!inputFile) {
        cerr << "Failed to open the file." << endl;
        return 1;
    }
    string content;
    string line;
    // Read the content of the file
    while (getline(inputFile, line)) {
        content += line + "\n";
    }
    // Close the input file
    inputFile.close();

    // Display the content of the file
    cout << "File Content:" << endl;
    cout << content << endl;

    // Open the same file for writing
    ofstream outputFile("sample.txt");

    if (!outputFile) {
        cerr << "Failed to open the file for writing." << endl;
        return 1;
    }
    // Write "object oriented" string to the file
    string textToWrite = "object oriented";
    outputFile << textToWrite << endl;

    // Close the output file
    outputFile.close();

    cout << "The string 'object oriented' has been written to the file." << endl;
    return 0;
}
```

```
File Content:
This is a sample text file.
It contains some text data.
You are reading this from a C++ program.

The string 'object oriented' has been written to the file.
```

5.Describe structure of c++ program.

C++ is first Object oriented programming language.We have summarize structure of C++ Program in the following Picture –

Structure of C++ Program


Header File Declaration Section
Global Declaration Section
Class Declaration and Method Definition Section
Main Function
Method Definition Section

Section 1 : Header File Declaration Section
1. Header files used in the program are listed here.
2. Header File provides Prototype declaration for different library functions.
3. We can also include user define header file.
4. Basically all preprocessor directives are written in this section.

Section 2 : Global Declaration Section
1. Global Variables are declared here.
2. Global Declaration may include –
    o Declaring Structure
    o Declaring Class
    o Declaring Variable

Section 3 : Class Declaration Section
1. Actually this section can be considered as sub section for the global

declaration section.
2. Class declaration and all methods of that class are defined here.

Section 4 : Main Function
1. Each and every C++ program always starts with main function.
2. This is entry point for all the function. Each and every method is called indirectly through main.
3. We can create class objects in the main.
4. Operating system call this function automatically.

Section 5 : Method Definition Section
1. This is optional section . Generally this method was used in C Programming.

6.Explain with suitable example Friend Function.

Friend function:
The private members of a class cannot be accessed from outside the class but in some situations two classes may need access of each other‟s private data. So a common function can be declared which can be made friend of more than one class to access the private data of more than one class. The common function is made friendly with all those classes whose private data need to be shared in that function. This common function is called as friend function. Friend function is not in the scope of the class in which it is declared. It is called without any object. The class members are accessed with the object name and dot membership operator inside the friend function. It accepts objects as arguments.

Example:
Program to interchange values of two integer numbers using friend function.

```
#include <iostream.h>
#include <conio.h>
class B;
class A
{
int x;
public:
void accept()
{
cout<<"\n Enter the value for x:";
cin>>x;
}
friend void swap(A,B);
};
class B
{
int y;
public:
void accept()
{
cout<<"\n Enter the value for y:";
cin>>y;
}
friend void swap(A,B);
};
void swap(A a,B b)
{
cout<<"\n Before swapping:";
cout<<"\n Value for x="<<a.x;
cout<<"\n Value for y="<<b.y;
int temp;
temp=a.x;
a.x=b.y;
b.y=temp;
cout<<"\n After swapping:";
cout<<"\n Value for x="<<a.x;
cout<<"\n Value for y="<<b.y;
}
void main()
{
A a;
B b;
clrscr();
a.accept();
b.accept();
swap(a,b);
getch();
}
```

```
Enter the value for x: 5
Enter the value for y: 10
Before swapping:
Value for x=5
Value for y=10
After swapping:
Value for x=10
Value for y=5
```

7.Describe all visibility modes and effects with example.

Base class visibility	**Derived class visibility**		
	Private	Protected	Public
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Private	Protected	Protected
Public	Private	Protected	Public

Private:
o When a base class is privately inherited by a derived class, „public members‟ and „protected members‟ of the base class become „private members‟ of the derived class.
o Therefore, the public and protected members of the base class can only be accessed by the member functions of derived class but, cannot be accessed by the objects of the derived class.
Syntax:

```
class derived: private base
{
//Members of derived class;
};
```

Public:

o When a base class is publicly inherited by a derived class then „protected members‟ of base class becomes „protected members‟ and ‟public members‟ of the base class become „public members‟ of the derived class.
o Therefore the public members of the base class can be accessed by both the member functions of derived class as well
as the objects of the derived class.
Syntax:
class derived:
public base
{
//Members of derived class;
};

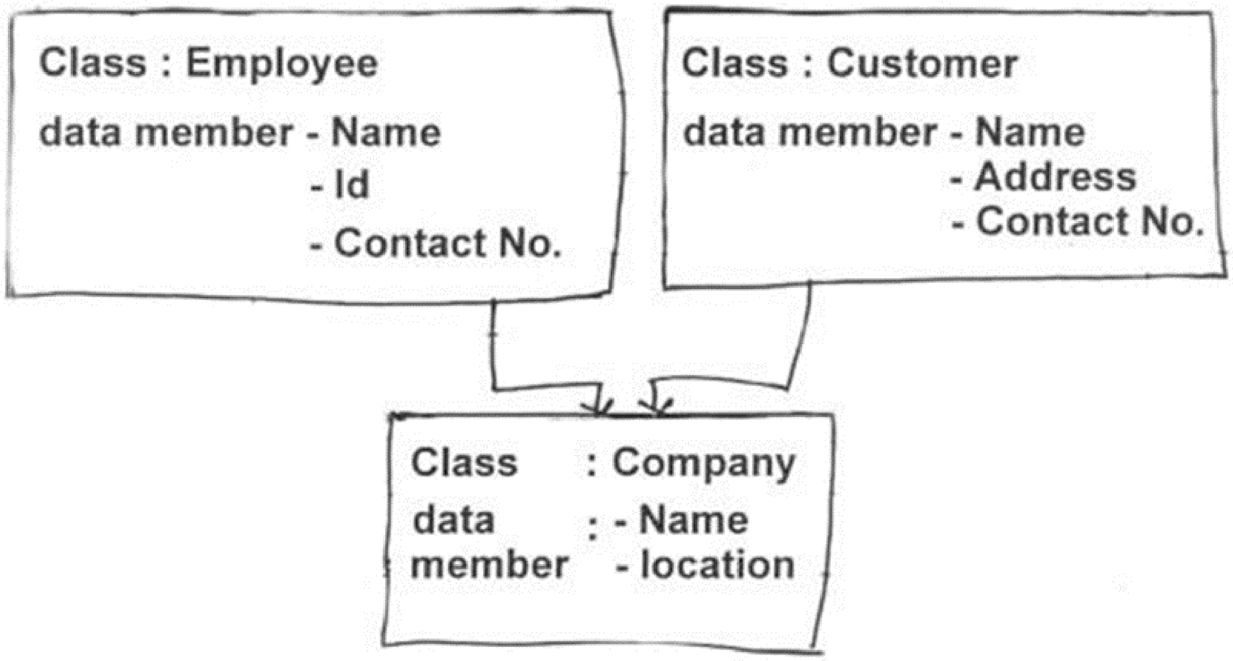
Protected:
o When a base class is protectedly inherited by a derived class, „public and protected members‟ of the base class become „protected members‟ of the derived class.
o Therefore the public and protected members of the base class can be accessed by the member functions of derived class as well as the member functions of immediate derived class of it but they cannot be accessed by the objects of derived class
Syntax:
class derived: protected base
{
//Members of derived class;
};

8.Differentiate between compile time polymorphism and Runtime polymorphism.

Aspect	Compile time polymorphism	Run time polymorphisim
Also Known As	Static or Early Binding Polymorphism	Dynamic or Late Binding Polymorphism
Resolution of Function Calls	Resolved by the Compiler	Not Resolved by the Compiler
Flexibility	Less Flexible	More Flexible
Implementation Mechanisms	Function Overloading, Operator Overloading	Virtual Functions, Function Overriding

Application Examples	Method Overloading	Method Overriding
Execution Speed	Faster Execution (Compile Time)	Slower Execution (Run Time)
Use Cases	Less Preferred for Complex Problems	Suitable for Handling Complex Problems
Knowledge of Methods and Details	Known During Compilation	Revealed During Runtime

9.Develop a c++ program to implement inheritance shown in following fig.
4M



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

// Base class Employee
class Employee
{
protected:
    string name;
    int id;
public:
    void inputEmployeeData()
{
        cout << "Enter Employee Name: ";
        cin >> name;
        cout << "Enter Employee ID: ";
        cin >> id;
    }

    void displayEmployeeData()
{
        cout << "Employee Name: " << name << endl;
        cout << "Employee ID: " << id << endl;
    }
};
// Base class Customer
class Customer
{
protected:
    string name;
    string contactNo;
    string address;

public:
    void inputCustomerData()
{
        cout << "Enter Customer Name: ";
        cin >> name;
        cout << "Enter Customer Contact No.: ";
        cin >> contactNo;
        cout << "Enter Customer Address: ";
        cin >> address;
    }
    void displayCustomerData()
{
        cout << "Customer Name: " << name << endl;
        cout << "Customer Contact No.: " << contactNo << endl;
        cout << "Customer Address: " << address << endl;
    }
};
```

```
// Derived class Company (Multiple Inheritance)
class Company : public Employee, public Customer {
private:
    string location;

public:
    void inputCompanyData() {
        inputEmployeeData(); // Input Employee data
        inputCustomerData(); // Input Customer data
        cout << "Enter Company Location: ";
        cin >> location;
    }

    void displayCompanyData() {
        displayEmployeeData(); // Display Employee data
        displayCustomerData(); // Display Customer data
        cout << "Company Location: " << location << endl;
    }
};
int main() {
    // Create an object of Company
    Company company;

    // Input and display data for Company
    cout << "Enter data for Company:\n";
    company.inputCompanyData();

    cout << "\nCompany Data:\n";
    company.displayCompanyData();
    return 0;
}
```

```
Enter data for Company:
Enter Employee Name: John
Enter Employee ID: 123
Enter Customer Name: Mary
Enter Customer Contact No.: 555-123-4567
Enter Customer Address: 123 Main St.
Enter Company Location: New York
```

```
Company Data:
Employee Name: John
Employee ID: 123
Customer Name: Mary
Customer Contact No.: 555-123-4567
Customer Address: 123 Main St.
Company Location: New York
```

10.Develop a c++ program to print sum of 10 no. using array.

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

int main()
{
    const int numCount = 10; // Number of elements in the array
    int numbers[numCount];   // Declare an array to store the numbers
    int sum = 0;             // Initialize a variable to store the sum

    // Input: Ask the user to enter 10 numbers
    cout << "Enter 10 numbers, one at a time:" << endl;
    for (int i = 0; i < numCount; ++i)
{
        cout << "Enter number " << (i + 1) << ": ";
        cin >> numbers[i];
    }

    // Calculate the sum of the numbers
    for (int i = 0; i < numCount; ++i)
{
        sum += numbers[i];
    }

    // Output: Print the sum
    cout << "Sum of the 10 numbers is: " << sum << endl;

    Return
```

```
 0;
}
```

```
Enter 10 numbers, one at a time:
Enter number 1: 5
Enter number 2: 12
Enter number 3: 7
Enter number 4: 23
Enter number 5: 1
Enter number 6: 9
Enter number 7: 8
Enter number 8: 15
Enter number 9: 3
Enter number 10: 10
Sum of the 10 numbers is: 93
```

When you run this program, it will prompt you to enter 10 numbers, and then it will display the sum of those numbers.

11.Develop a c++ program to perform arithmetic operation using Pointer.

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

int main() {
    int num1, num2;
    int *ptr1, *ptr2;

    // Input: Ask the user to enter two numbers
    cout << "Enter the first number: ";
    cin >> num1;
    cout << "Enter the second number: ";
    cin >> num2;

    // Assign the addresses of the variables to pointers
    ptr1 = &num1;
    ptr2 = &num2;
    // Perform arithmetic operations using pointers
    int sum = *ptr1 + *ptr2;
    int difference = *ptr1 - *ptr2;
    int product = *ptr1 * *ptr2;
    float quotient = static_cast<float>(*ptr1) / *ptr2; // Ensure floating-point division

    // Output: Display the results
    cout << "Sum: " << *ptr1 << " + " << *ptr2 << " = " << sum << endl;
    cout << "Difference: " << *ptr1 << " - " << *ptr2 << " = " << difference << endl;
    cout << "Product: " << *ptr1 << " * " << *ptr2 << " = " << product << endl;
    cout << "Quotient: " << *ptr1 << " / " << *ptr2 << " = " << quotient << endl;

    return 0;
}
```

```
Enter the first number: 15
Enter the second number: 7
Sum: 15 + 7 = 22
Difference: 15 - 7 = 8
Product: 15 * 7 = 105
Quotient: 15 / 7 = 2.14286
```

This program demonstrates how to perform arithmetic operations using pointers to access the values of variables.

12.Develop a c++ program to create structure student with data member Name, Roll No., percentage accept and display data for 5 student using structure.

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

// Define the structure Student
struct Student {
    string Name;
    int RollNo;
    double Percentage;
};

int main() {
    const int numStudents = 5;
    Student students[numStudents]; // Declare an array of Student structures

    // Input: Accept data for 5 students
    for (int i = 0; i < numStudents; ++i) {
        cout << "Enter data for Student " << (i + 1) << ":" << endl;
        cout << "Name: ";
        cin.ignore(); // Ignore any previous newline character
        getline(cin, students[i].Name);
        cout << "Roll No: ";
        cin >> students[i].RollNo;
        cout << "Percentage: ";
        cin >> students[i].Percentage;
    }

    // Output: Display data for all 5 students
    cout << "\nStudent Data:\n";
    for (int i = 0; i < numStudents; ++i) {
        cout << "Student " << (i + 1) << ":\n";
        cout << "Name: " << students[i].Name << endl;
        cout << "Roll No: " << students[i].RollNo << endl;
        cout << "Percentage: " << students[i].Percentage << "%" << endl << endl;
    }
    return 0;
}
```

```
Enter data for Student 1:
Name: Alice Johnson
Roll No: 101
Percentage: 87.5

Enter data for Student 2:
Name: Bob Smith
Roll No: 102
Percentage: 78.3
```

```
Enter data for Student 3:
Name: Carol Davis
Roll No: 103
Percentage: 92.1

Enter data for Student 4:
Name: David Lee
Roll No: 104
Percentage: 68.9

Enter data for Student 5:
Name: Eve Wilson
Roll No: 105
Percentage: 95.2
```

This program allows you to input and display data for multiple students using the defined structure.

13.Develop c++ program to check Detection of end of file.

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

int main() {
    // Open a file for reading
    ifstream inputFile("example.txt");

    // Check if the file is open
    if (!inputFile.is_open()) {
        cerr << "Error opening the file." << endl;
        return 1;
    }

    // Read and display the contents of the file
    char character;
    while (!inputFile.eof()) {
        inputFile.get(character);
        // Check for the end of file
        if (inputFile.eof()) {
            cout << "End of file reached." << endl;
            break; // Exit the loop when the end of file is detected
        }
        cout << character;
    }

    // Close the file
    inputFile.close();
    return 0;
}
```

```
This is a sample text file.
It contains multiple lines.
Here is another line.
End of file reached.
```

This program reads characters from the file until the end of the file is detected and then prints a message indicating the end of the file.

Summer-22
4 Marks Questions

1. State and describe characteristics of object oriented programming. (any 4 points can be considered)

Object-Oriented Programming (OOP) is a programming paradigm that is based on the concept of objects. Objects are instances of classes, which serve as templates for creating objects. OOP has several characteristics that make it a powerful and widely used programming approach. Here are the key characteristics of OOP:

1. **Objects:** Objects are the fundamental building blocks of OOP. An object is an instance of a class and encapsulates both data (attributes) and behaviors (methods). For example, in a car simulation program, a car object would have attributes like color, make, and model, as well as behaviors like starting, stopping, and accelerating.
Class declaration syntax:

```
class class_name
{
private:
```

```
variable declarations;
function declarations;
public:
variable declarations;
function declarations;
};
int main()
{
class_name object_name;
}
```

2. **Classes**: Classes are blueprints or templates for creating objects. They define the structure and behavior of objects. A class specifies what attributes an object will have and what methods it can perform. Classes provide a way to create multiple objects with similar characteristics and behaviors.

Class declaration syntax:

```
class class_name
{
private:
variable declarations;
function declarations;
public:
variable declarations;
function declarations;
};
```

Abstraction: Abstraction refers to showing only the essential features of the application and hiding the details. In C++, classes can provide methods to the outside world to access & use the data variables, keeping the variables hidden from direct access, or classes can even declare everything accessible to everyone, or maybe just to the classes inheriting it. This can be done using access specifiers.
For example, When you send an sms you just type the message, select the contact and click send, the phone shows you that the message has been sent, what actually happens in background when you click send is hidden from you as it is not relevant to you.

Encapsulation: It can also be said data binding. Encapsulation is all about combining the data variables and functions together in class. This is to avoid the access of private data members from outside the class.

Inheritance: Inheritance is a way to reuse once written code again and again. As the name suggests Inheritance is the process of forming a new class from an existing class, the existing class is called as base class, new class is called as derived class.They are also called parent and child class. So when, a derived class inherits a base class, the derived class can use all the functions which are defined in base class, hence making code reusable.

Polymorphism: It is a feature, which lets us create functions with same name but different arguments, which will perform different actions. That means, functions with same name, but functioning in different ways. Or, it also allows us to redefine a function to provide it with a completely new definition. Poly refers to many. That is a single function or an operator functioning in many ways different upon the usage is called polymorphism.

7. **Message Passing:** In OOP, objects communicate with each other by sending and receiving messages. Objects interact by invoking methods on each other. This interaction is essential for achieving the desired behavior of the system.

8. **Modularity:** OOP promotes modularity, which means breaking down a complex system into smaller, manageable, and reusable modules or classes. Each module is responsible for a specific aspect of the system's functionality, making the code more organized and easier to maintain.

9. **Reusability:** OOP encourages code reuse through inheritance and the creation of reusable classes and objects. Reusable code reduces development time and enhances the reliability of the software.

10. **Flexibility and Extensibility:** OOP allows you to easily modify and extend existing code by adding new classes or changing the behavior of existing ones. It supports software evolution and adaptation to changing requirements.

11. **Security:** OOP supports access control mechanisms, such as public, private, and protected access specifiers, which help in implementing security and preventing unauthorized access to data.

12. **Real-World Modeling:** OOP is well-suited for modeling real-world entities and systems. It allows developers to map real-world concepts and relationships directly into software design, making it easier to understand and maintain.

OOP provides a structured and organized way of designing and developing software by emphasizing the use of objects and their interactions. These characteristics make OOP a powerful and widely used programming paradigm for building complex and maintainable software systems.

2.**Write a C++ program to declare a class college with name and code. Derive a new class as student with members as name. Accept and display details of one student along with college data.**

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

// Base class College
class College {
protected:
    string collegeName;
    int collegeCode;

public:
    void inputCollegeData() {
        cout << "Enter College Name: ";
        cin >> collegeName;
        cout << "Enter College Code: ";
        cin >> collegeCode;
    }

    void displayCollegeData() {
        cout << "College Name: " << collegeName << endl;
        cout << "College Code: " << collegeCode << endl;
    }
};

// Derived class Student
class Student : public College {
protected:
    string studentName;

public:
    void inputStudentData() {
        cout << "Enter Student Name: ";
        cin >> studentName;
    }
    void displayStudentData() {
        cout << "Student Name: " << studentName << endl;
    }
};

int main() {
    Student student;

    cout << "Input College Data:\n";
    student.inputCollegeData();

    cout << "\nInput Student Data:\n";
    student.inputStudentData();

    cout << "\nCollege and Student Data:\n";
    student.displayCollegeData();
    student.displayStudentData();

    return 0;
}
```

```
Input College Data:
Enter College Name: Sample College
Enter College Code: 123

Input Student Data:
Enter Student Name: John Doe

College and Student Data:
College Name: Sample College
College Code: 123
Student Name: John Doe
```

3.Write a C++ program to declare a class student with data members as roll no and name. Declare a constructor to initialize data members of class. Display the data.

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

class Student {
private:
    int rollNo;
    string name;

public:
    // Constructor to initialize data members
    Student(int roll, const string& studentName) : rollNo(roll), name(studentName) {}

    // Member function to display data
    void displayData() {
        cout << "Roll No: " << rollNo << endl;
        cout << "Name: " << name << endl;
    }
};

int main() {
    // Create a Student object and initialize its data members
    Student student1(101, "John Doe");

    // Display the data using the member function
    student1.displayData();

    return 0;
}
```

```
Roll No: 101
Name: John Doe
```

4. Describe the concept of type casting using suitable example.

Type casting, also known as type conversion, is a fundamental concept in programming that allows you to convert a value from one data type to another. This is essential when you need to work with different data types in your code.

There are two types of type casting in C++:

1. **Implicit Type Casting (Automatic Type Conversion):**
   - Implicit type casting occurs automatically by the compiler when you assign a value of one data type to a variable of another data type.
   - This type of casting is safe when there is no loss of data.
   - For example, assigning an `int` to a `float` is an implicit type casting. The `int` is automatically converted to a `float` without any data loss.

2**. Explicit Type Casting (Manual Type Conversion):**
   - Explicit type casting is performed manually by the programmer using casting operators.
   - It is used when there may be a risk of data loss or when you want to enforce a specific type conversion.
   - C++ provides casting operators like `static_cast`, `dynamic_cast`, `const_cast`, and `reinterpret_cast`.
   - For example, using `static_cast` allows you to explicitly convert a `float` to an `int`, but you must be aware that it may truncate the fractional part of the number.

Example:-

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

int main() {
    // Implicit type casting (int to float)
    int num = 10;
    float floatValue = num; // Implicitly converts int to float

    cout << "Implicit type casting (int to float): " << floatValue << endl;

    // Explicit type casting (float to int)
    float anotherFloatValue = 15.75;
    int intNum = static_cast<int>(anotherFloatValue); // Explicitly converts float to int

    cout << "Explicit type casting (float to int): " << intNum << endl;

    return 0;
}
```

```
Implicit type casting (int to float): 10
```

```
Explicit type casting (float to int): 15
```

Type casting is a way to ensure that data of one data type can be used in operations or assignments involving another data type. Implicit type casting happens automatically when it's safe, while explicit type casting is done manually when you need control or when data loss may occur. It's important to use type casting carefully to avoid unexpected results and data loss in your programs.

5. Write a C++ program to declare a class mobile having data members as price and model number. Accept and display the data for Ten objects.

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

class Mobile {
private:
    float price;
    string modelNumber;

public:
    // Member function to accept data
    void acceptData() {
        cout << "Enter Mobile Price: ";
        cin >> price;
        cin.ignore(); // Ignore any previous newline character
        cout << "Enter Model Number: ";
        getline(cin, modelNumber);
    }

    // Member function to display data
    void displayData() {
        cout << "Mobile Price: " << price << endl;
        cout << "Model Number: " << modelNumber << endl;
    }
};

int main() {
    Mobile mobiles[10]; // Create an array of 10 Mobile objects

    for (int i = 0; i < 10; ++i) {
        cout << "Enter data for Mobile " << i + 1 << ":" << endl;
        mobiles[i].acceptData();
    }

    cout << "\nDisplaying Mobile Data:\n";
    for (int i = 0; i < 10; ++i) {
        cout << "Mobile " << i + 1 << ":\n";
        mobiles[i].displayData();
        cout << endl;
    }
    return 0;
}
```

```
Enter data for Mobile 1:
Enter Mobile Price: 499.99
Enter Model Number: XYZ-123

Enter data for Mobile 2:
Enter Mobile Price: 699.99
Enter Model Number: ABC-456

... (similar input for Mobiles 3 to 10)

Displaying Mobile Data:
Mobile 1:
Mobile Price: 499.99
Model Number: XYZ-123

Mobile 2:
Mobile Price: 699.99
Model Number: ABC-456

... (similar output for Mobiles 3 to 10)
```

6. Write a C++ program to copy the contents of a source file student 1.txt to a destination file student 2.txt using file operations.

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

int main() {
   // Open the source file for reading
   ifstream sourceFile("student1.txt");
   if (!sourceFile) {
      cerr << "Error: Unable to open source file." << endl;
      return 1;
   }

   // Open the destination file for writing
   ofstream destinationFile("student2.txt");
   if (!destinationFile) {
      cerr << "Error: Unable to open destination file." << endl;
      sourceFile.close(); // Close the source file
      return 1;
   }

   // Copy the contents from the source file to the destination file
   string line;
   while (getline(sourceFile, line)) {
      destinationFile << line << endl;
   }

   // Close both files
   sourceFile.close();
   destinationFile.close();

   cout << "Contents copied from student1.txt to student2.txt." << endl;

   return 0;
}
```

```
Contents copied from student1.txt to student2.txt.
```

7. Describe visibility modes and their effects used in inheritance.

Base class visibility	**Derived class visibility**		
	Private	Protected	Public
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Private	Protected	Protected
Public	Private	Protected	Public

Private:
o When a base class is privately inherited by a derived class, „public members‟ and „protected members‟ of the base class become „private members‟ of the derived class.
o Therefore, the public and protected members of the base class can only be accessed by the member functions of derived class but, cannot be accessed by the objects of the derived class.
Syntax:

```
class derived: private base
{
//Members of derived class;
};
```

Public:
o When a base class is publicly inherited by a derived class then „protected members‟ of base class becomes „protected members‟ and ‟public members‟ of the base class become „public members‟ of the derived class.
o Therefore the public members of the base class can be accessed by both the member functions of derived class as well
as the objects of the derived class.
Syntax:

class derived:
public base
{
//Members of derived class;
};

Protected:
o When a base class is protectedly inherited by a derived class, „public and protected members‟ of the base class become „protected members‟ of the derived class.
o Therefore the public and protected members of the base class can be accessed by the member functions of derived class as well as the member functions of immediate derived class of it but they cannot be accessed by the objects of derived class
Syntax:
class derived: protected base
{
//Members of derived class;
};

8. Differentiate between compile time and run time polymorphism.

Aspect	**Compile time polymorphism**	**Run time polymorphisim**
Also Known As	Static or Early Binding Polymorphism	Dynamic or Late Binding Polymorphism
Resolution of Function Calls	Resolved by the Compiler	Not Resolved by the Compiler
Flexibility	Less Flexible	More Flexible
Implementation Mechanisms	Function Overloading, Operator Overloading	Virtual Functions, Function Overriding
Application Examples	Method Overloading	Method Overriding

Execution Speed	Faster Execution (Compile Time)	Slower Execution (Run Time)
Use Cases	Less Preferred for Complex Problems	Suitable for Handling Complex Problems
Knowledge of Methods and Details	Known During Compilation	Revealed During Runtime

9.Describe the concept of virtual base class with example.

10. Write a C++ program to overload add function to add two integer numbers and two float numbers.

```
#include <iostream>

// Function to add two integers
int add(int a, int b) {
   return a + b;
}

// Function to add two floats
float add(float a, float b) {
   return a + b;
}

int main() {
   int int1 = 5, int2 = 7;
   float float1 = 3.5, float2 = 2.5;

   // Call the add function for integers
   int sum_int = add(int1, int2);
   std::cout << "Sum of integers: " << sum_int << std::endl;

   // Call the add function for floats
   float sum_float = add(float1, float2);
```

```
    std::cout << "Sum of floats: " << sum_float << std::endl;

    return 0;
}
```

```
Sum of integers: 12
Sum of floats: 6
```

11.Write a C++ program to find and display the smallest number of an array.

```
#include <iostream>

int main() {
    int arr[] = {12, 45, 1, 67, 8, 34, 56, 89, 3, 23};
    int n = sizeof(arr) / sizeof(arr[0]);

    // Initialize the minimum with the first element of the array
    int min = arr[0];

    // Loop through the array to find the minimum
    for (int i = 1; i < n; i++) {
        if (arr[i] < min) {
            min = arr[i];
        }
    }

    // Display the minimum number
    std::cout << "The smallest number in the array is: " << min << std::endl;

    return 0;
}
```

```
The smallest number in the array is: 1
```

12.Write a C++ program to declare a structure book with members as book id and name. Accept and display data of five books using array of structure.

```
#include <iostream>
#include <string>

// Define the structure for a book
struct Book {
    int bookID;
    std::string name;
};

int main() {
    const int numBooks = 5;
    Book books[numBooks];  // Array of structures to store book data

    // Accept data for five books
    for (int i = 0; i < numBooks; i++) {
        std::cout << "Enter data for Book " << i + 1 << ":" << std::endl;
        std::cout << "Book ID: ";
        std::cin >> books[i].bookID;
        std::cin.ignore();  // Clear the newline character from the input buffer
        std::cout << "Book Name: ";
        std::getline(std::cin, books[i].name);
    }

    // Display data for the five books
    std::cout << "\nBook Data:" << std::endl;
    for (int i = 0; i < numBooks; i++) {
        std::cout << "Book " << i + 1 << " - ID: " << books[i].bookID << ", Name: " <<
books[i].name << std::endl;
    }

    return 0;
}
```

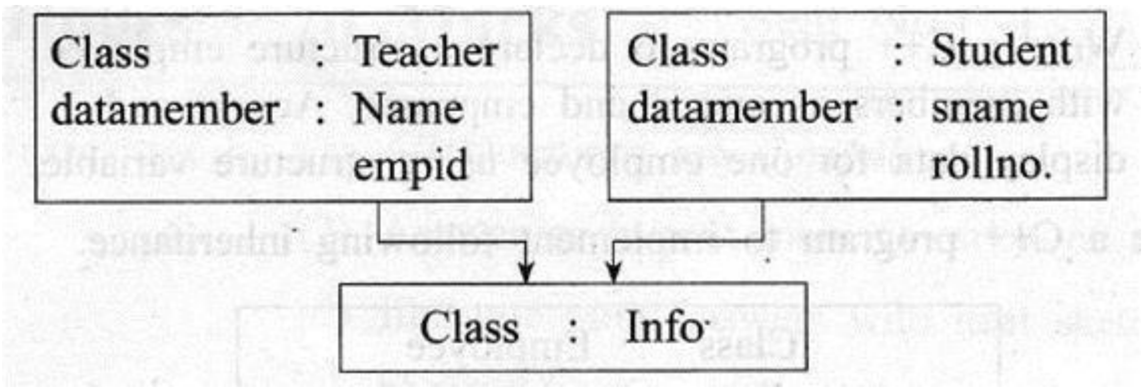
```
Enter data for Book 1:
Book ID: 101
Book Name: Book 1 Title

Enter data for Book 2:
Book ID: 102
Book Name: Book 2 Title

Enter data for Book 3:
Book ID: 103
Book Name: Book 3 Title

Enter data for Book 4:
Book ID: 104
Book Name: Book 4 Title

Enter data for Book 5:
Book ID: 105
Book Name: Book 5 Title

Book Data:
Book 1 - ID: 101, Name: Book 1 Title
Book 2 - ID: 102, Name: Book 2 Title
Book 3 - ID: 103, Name: Book 3 Title
Book 4 - ID: 104, Name: Book 4 Title
Book 5 - ID: 105, Name: Book 5 Title
```

13.Describe constructor with default arguments with an example.

```
#include <iostream>

class Rectangle {
private:
    double length;
    double width;

public:
    // Constructor with default arguments
    Rectangle(double len = 1.0, double wid = 1.0) {
        length = len;
        width = wid;
    }

    double calculateArea() {
        return length * width;
    }
};

int main() {
    // Creating objects of the Rectangle class with default arguments
    Rectangle rect1;  // Uses default values (length = 1.0, width = 1.0)
    Rectangle rect2(5.0);  // Sets length to 5.0, width remains default (1.0)
    Rectangle rect3(3.0, 4.0);  // Sets both length and width

    std::cout << "Area of rect1: " << rect1.calculateArea() << std::endl;
    std::cout << "Area of rect2: " << rect2.calculateArea() << std::endl;
    std::cout << "Area of rect3: " << rect3.calculateArea() << std::endl;

    return 0;
}
```

```
Area of rect1: 1
Area of rect2: 5
Area of rect3: 12
```

14.Write a C++ program to implement inheritance shown in following figure
4M



Accept and display data of one teacher and one student using object of class 'Info'.

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

class Teacher
{
protected:
char Name[20];
int empid;
};

class Student
{
protected:
char sname[20];
int rollno;
};

class Info:public Teacher,public Student
{
public:
void acceptT()
{
cout<<"\nEnter data for teacher:";
cout<<"\nName:";
cin>>Name;
cout<<"Employee id:";
cin>>empid;
}
void displayT()
{
cout<<"\nTeacher's data is:";
cout<<"\nName:"<<Name;
cout<<"\nEmployee id:"<<empid;
}
void acceptS()
{
cout<<"\n\nEnter student's data:";
cout<<"\nName:";
cin>>sname;
cout<<"Roll no:";
cin>>rollno;
}
void displayS()
{
cout<<"\nStudent's data is:";
cout<<"\nName:"<<sname;
cout<<"\nRoll no:"<<rollno;
}
};
void main()
{
Info I;
clrscr();
I.acceptT();
I.displayT();
I.acceptS();
I.displayS();
getch();
}
```

```
Enter data for teacher:
Name: John
Employee id: 12345

Teacher's data is:
Name: John
Employee id: 12345
```

```
Enter student's data:
Name: Alice
Roll no: 101

Student's data is:
Name: Alice
Roll no: 101
```

Summer-19
6 Marks Questions

1.
(i) Write any three rules of operator overloading.
(ii) Write a program in C++ to overload unary '_' operator to negate values of data members of class.

(i) Write any three rules of operator overloading.

Rules for overloading operators:
1. Only existing operators can be overloaded. New operators cannot be created.
2. The overloaded operator must have at least one operand that is of user defined data type.
3. We can't change the basic meaning of an operator. That is to say, we can't redefine the plus(+) operator to subtract one value from other.
4. Overloaded operators follow the syntax rules of the original operators. They can't be overridden.
5. There are some operators that can't be overloaded.
6. We can't use friend functions to overload certain operators. However, member function scan be used to overload them.
7. Unary operators overloaded by means of member function take no explicit arguments and return no explicit values, but, those overloaded by means of the friend function, take one reference argument (the object of the releva**nt class).**
8. Binary operators overloaded through a member function, take oneexplicit argument and those which are overloaded through a friend
function take two explicit arguments.
9. When using binary operators overloaded through a member
 function, the left hand operand must be an object of the relevant

class.
10. Binary arithmetic operators such as +,-,* and / must explicitly returna value. They must not attempt to change their own arguments.

(ii) Write a program in C++ to overload unary '_' operator to negate values of data members of class.

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class Number
{
int x, y;
public:
Number (int a,int b)
{
a =x;
b =y;
}
void display()
{
cout<<"value of x=”<<x<<”\nValue of y= ”<<y;
}
void operator - ( )
{
x = - x;
y = - y;
}
};
void main()
{
Number N1(5,6);
clrscr();
N1.display();
-N1;
cout<<"\n After negation:";
N1. display ();
getch();
}
```

```
Value of x=5
Value of y=6
After negation:
Value of x=-5
Value of y=-6
```

2.Write a C++ program to append data from abc.txt to xyz.txt file. (Note: Any other correct logic shall be considered)

Assuming input file as abc.txt with contents "World" and output file named as xyz.txt with contents "Hello" have been already created.

```
#include <iostream.h>
#include<fstream.h>
int main()
{
fstream f;
ifstream fin;
fin.open("abc.txt",ios::in);
ofstream fout;
fout.open("xyz.txt", ios::app);
if (!fin)
 {
 cout<< "file not found";
 }
 else
 {
 fout<<fin.rdbuf();
 }
 char ch;
 f.seekg(0);
 while (f)
 {
 f.get(ch);
 cout<< ch;
 }
 f.close();
 return 0;
}
```

Output:
```
Hello World
```

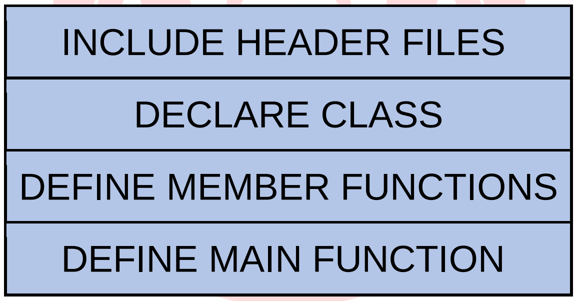
3.Write a C++ program to declare a class student with members as roll no, name and department. Declare a parameterized constructor with default value for department as 'CO' to initialize members of object. Initialize and display data for two students. (Note: Any other relevant logic should be considered).

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class student
{
int roll_no;
char name[20],department[40];
public:
student(int rno,char *n,char *d="CO")
{
roll_no=rno;
strcpy(name,n);
strcpy(department,d);
}
void display()
{
cout<<"\n Roll No:"<<roll_no;
cout<<"\n Name:"<<name;
cout<<"\n Department:"<<department;
}
};
void main()
{
student s1(112," Chitrakshi"),s2(114,"Anjali");
clrscr();
s1.display();
s2.display();
getch();
}
```

```
Roll No:112
Name: Chitrakshi
Department:CO

Roll No:114
Name:Anjali
Department:CO
```

4.
(i) Describe structure of C++ program with diagram.
(ii) Write a C++ program to add two 3 x 3 matrices and display addition.

(i) Describe structure of C++ program with diagram.



Description:-
1. Include header files In this section a programmer include all header files which are require to execute given program. The most important file is iostream.h header file. This file defines most of the C++statements like cout and cin. Without this file one cannot load C++ program.
2. Declare Class In this section a programmer declares all classes which are necessary for given program. The programmer uses general syntax of creating class.
3. Define Member Functions This section allows programmer to design member functions of a class. The programmer can have inside declaration of a function or outside declaration of a function.
4. Define Main Functions This section the programmer creates object and call various functions writer within various class.

(ii) Write a C++ program to add two 3 x 3 matrices and display addition. (Note: Any other relevant logic should be considered).

```
#include
#include
void main()
{
clrscr();
int mat1[3][3], mat2[3][3], i, j, mat3[3][3];
cout<<"Enter matrix 1 elements :";
for(i=0; i<3; i++)
{
for(j=0; j<3; j++)
{
cin>>mat1[i][j];
}
}
cout<<"Enter matrix 2 elements :";
for(i=0; i<3; i++)
{
for(j=0; j<3; j++)
{
cin>>mat2[i][j];
}
}
cout<<"Adding the two matrix to form the third matrix\n";
for(i=0; i<3; i++)
{
for(j=0; j<3; j++)
{
mat3[i][j]=mat1[i][j]+mat2[i][j];
}
}
cout<<"The two matrix added successfully...!!";
cout<<"The new matrix will be :\n";
for(i=0; i<3; i++)
{
for(j=0; j<3; j++)
{
cout<<mat3[i][j]<<" ";
}
cout<<"\n";
}
getch();
}
```

```
Enter matrix 1 elements:
1 2 3
4 5 6
7 8 9
Enter matrix 2 elements:
9 8 7
6 5 4
3 2 1
Adding the two matrices to form the third matrix...
The two matrices added successfully...!!
The new matrix will be:
10 10 10
10 10 10
10 10 10
```

5.Write a program to swap two integers using call by reference method. (Note: Any other relevant logic should be considered).

```
#include<iostream.h>
#include<conio.h>
void swap(int*p, int*q)
{
int t;
t=*p;
*p=*q;
*q=t;
}
```

```
void main()
{
int a,b;
float x,y;
clrscr();
cout<<"Enter values of a and b\n";
cin>>a>>b;
cout<<"Before swapping\n";
cout<<"a="<<a<<"\tb="<<b<<endl;
swap(&a, &b);
cout<<"After swapping\n";
cout<<"a="<<a<<"\tb="<<b<<endl;
getch();
}
```

```
Enter values of a and b
10 20
Before swapping
a=10    b=20
After swapping
a=20    b=10
```

6.Write a C++ program to implement following in heritance. Refer Figure No.2:

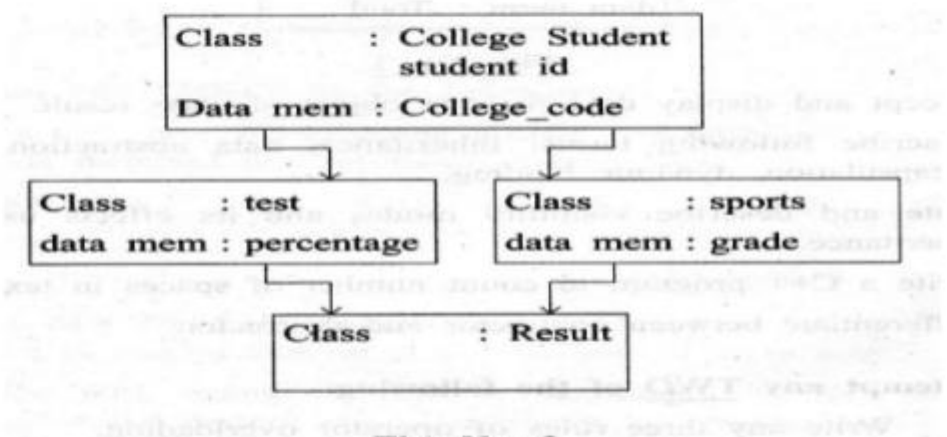


Fig. No. 2

Accept and display data for one object of class result (Hint: use virtual base class).

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

class College_Student {
protected:
    int student_id;
    char College_code[5];

public:
    void read_collegeStud_Data() {
        cout << "Enter college code and student id: ";
        cin >> College_code >> student_id;
    }

    void display_collegeStud_Data() {
        cout << "\ncollege code\tstudent id\n";
        cout << College_code << "\t" << student_id << endl;
    }
};

class test : virtual public College_Student {
protected:
    float percentage;

public:
    void read_test() {
        cout << "Enter test percentage: ";
        cin >> percentage;
    }

    void display_test() {
        cout << "\nTest percentage: " << percentage;
    }
};

class sports : virtual public College_Student {
protected:
    char grade[5];

public:
    void read_sportsData() {
        cout << "Enter sport grade: ";
        cin >> grade;
    }

    void display_sportsData() {
        cout << "\nSport grade: " << grade;
    }
};

class result : public test, public sports {
public:
    void read_result() {
        read_collegeStud_Data();
        read_test();
        read_sportsData();
    }

    void display_result() {
        display_collegeStud_Data();
        display_test();
        display_sportsData();
    }
};

void main() {
    result r;
    clrscr();
    r.read_result();
    r.display_result();
    getch();
}
```

```
Enter college code and student id: ABC12 12345
Enter test percentage: 85.5
Enter sport grade: A

college code    student id
ABC12           12345
```

```
Test percentage: 85.5

Sport grade: A
```

Winter-18
6 Marks Questions

1.Write a C++ program to overload binary operator '+' to concatenate two strings.

```
#include <iostream.h>
#include <conio.h>
#include class opov
{
char str1[10]; public: void getdata()
{
cout<<"\nEnter a strings";
cin>>str1;
}
void operator +(opov o)
{
cout<<strcat(str1,o.str1);
}
 };
void main()
{
opov o1,o2;
clrscr();
o1.getdata();
o2.getdata();
 o1+o2;
getch();

}
```

```
Enter a string: Hello
Enter a string: World
```

```
HelloWorld
```

2.Write a C++ program to write 'Welcome to poly' in a file. Then read the data from file and display it on screen. Note: Any other relevant logic shall be considered

```
#include<iostream.h>
#include<conio.h>
#include<fstream.h>
void main()
 {
char str[25] = "Welcome to poly",ch;
clrscr();
ofstream fout;
fout.open("output.txt");
fout<<str;
fout.close();
ifstream fin;
fin.open("output.txt");
while (!fin.eof())
 {
 fin.getline(str, 25);
 cout<<str<<endl;
 }
fin.close(); getch();
}
```

```
Welcome to poly
```

3.Write a C++ program to declare a class 'Account' with data members as accno, name and bal. Accept data for eight accounts and display details of accounts having balance less than 10,000.

```
#include<iostream.h>
#include<conio.h>
class Account
 {
```

```
long int accno, bal;
char name[10];
public:
void getdata()
{
cout<<"\nEnter account number, balance and name ";
cin>>accno>>bal>>name;
}
void putdata()
{
if(bal>10000)
 {
cout<<"\nThe Account Number is "<<accno;
cout<<"\nThe Balance is "<<bal;
cout<<"\nThe Name is "<<name;
 }
}
 };
void main()
 {
 Account a[8];
int i;
clrscr();
for(i=0;i<8;i++)
 {
a[i].getdata();
 }
for(i=0;i<8;i++)
 {
a[i].putdata();
} getch();
}
```

```
Enter account number, balance and name 1 15000 John
Enter account number, balance and name 2 8000 Alice
Enter account number, balance and name 3 12000 Bob
Enter account number, balance and name 4 9500 Carol
Enter account number, balance and name 5 11000 David
Enter account number, balance and name 6 7500 Emma
Enter account number, balance and name 7 13000 Frank
Enter account number, balance and name 8 9000 Grace

The Account Number is 1
The Balance is 15000
The Name is John

The Account Number is 3
The Balance is 12000
The Name is Bob

The Account Number is 5
The Balance is 11000
The Name is David

The Account Number is 7
The Balance is 13000
The Name is Frank
```

4.Write a C++ program to find whether the entered number is even or odd.

```
#include<iostream.h>
#include<conio.h>
void main()
{
int num; clrscr(); cout<<"\nEnter a Number "; cin>>num; if(num%2==0)
{
cout<<"\nEntered number is even";
}
else
{
cout<<"\nEntered number is odd";
}
getch();
}
```

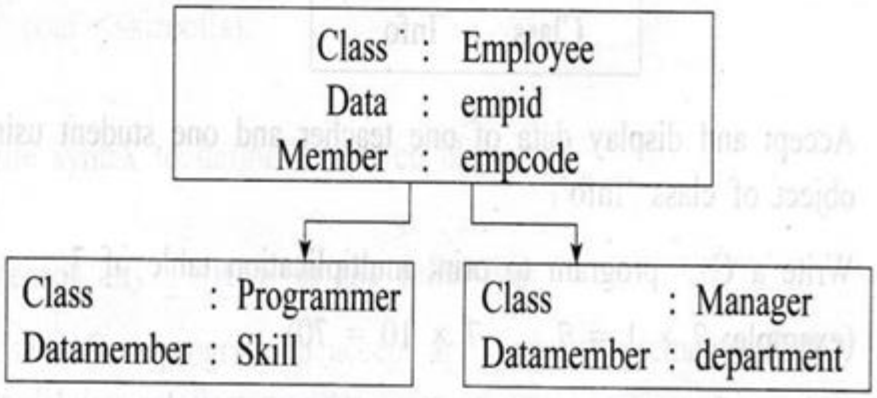
```
Enter a Number 7
Entered number is odd
```

5.Write a C++ program to declare a structure employee with members as empid and empname. Accept and display data for one employee using structure variable.

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

struct employee
 {
int empid;
char empname[10];
 };
void main()
 {
employee e;
clrscr();
cout<<"\nEnter employee id and Employee Name ";
cin>>e.empid>>e.empname;
cout<<"\mThe Employee Id is "<<e.empid;
cout<<"\nThe Employee Name is "<<e.empname;
getch();
 }
```

```
Enter employee id and Employee Name 101 John
The Employee Id is 101
The Employee Name is John
```

6.Write a C++ program to implement the following inheritance. 6M



Accept and display data for one programmer and one manager.Make display function virtual.

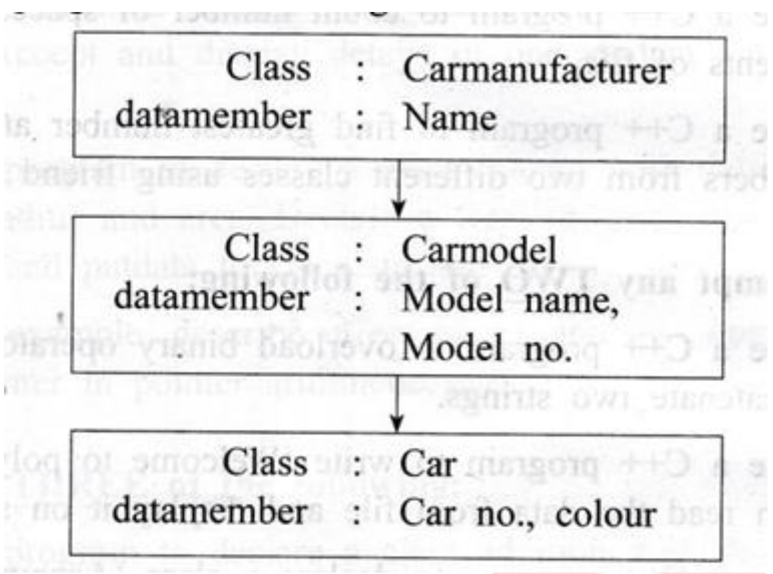
```
#include<iostream.h>
#include<conio.h>
class Employee
{
int empid,empcode;

public:
void emp()
{
cout<<"\nEnter an employee id ";
cin>>empid;
cout<<"\nEnter an employee code ";
cin>>empcode;
}
void virtual display()
{
cout<<"\nEmployee id "<<empid;
cout<<"\nEmployee code"<<empcode;
}
};

class Programmer : public Employee
{
```

```
char Skill[10];

public:
void getskill()
{
cout<<"\nEnter a Skill for Programmer ";
cin>>Skill;
}
void display()
{
cout<<"\nThe Programmer Skill is "<<Skill;
}
};

class Manager : public Employee
{
char department[10];

public:
void getdept()
{
cout<<"\nEnter a Department for Manager ";
cin>>department;
}
void display()
{
cout<<"\nThe Department of Manager is "<<department;
}
};

void main()
{
Employee e, *eptr;
Programmer p;
Manager m;
clrscr();
cout<<"\nFor Programmer Class ";
eptr = &e;
eptr->emp();
p.getskill();
eptr->display();
```

```
eptr= &p;
eptr->display();
cout<<"\n\nFor Manager Class ";
eptr = &e;
eptr->emp();
m.getdept();
eptr->display();
eptr= &m;
eptr->display();

getch();
}
```

```
For Programmer Class
Enter an employee id 101
Enter an employee code 001
Enter a Skill for Programmer C++
Employee id 101
Employee code 1
The Programmer Skill is C++

For Manager Class
Enter an employee id 102
Enter an employee code 002
Enter a Department for Manager HR
Employee id 102
Employee code 2
The Department of Manager is HR
```

7.Write a C++ program for following multilevel inheritance
. 6M



Accept and display data for one car with all details.

```
#include<iostream.h>
#include<conio.h>
class Carmanufacturer
{
char Name[10];

public:
void getcarm()
{
cout<<"\nEnter Car Name ";
cin>>Name;
}
void putcarm()
{
cout<<"\nThe Car Name is "<<Name;
}
};

class Carmodel : public Carmanufacturer
{
char Modelname[10];
int Modelno;
public:
void getcarmodel()
{
cout<<"\nEnter Car Model Name and Model No. ";
cin>>Modelname>>Modelno;
}
void putcarmodel()
{
cout<<"\nEnter Car Model Name and Model No.\n\t"<<Modelname<<"\t\t"<<Modelno;
}
};

class Car: public Carmodel
{
char colour[10],
Carno[10];

public:
void getcar()
{
cout<<"\nEnter Car colour and car number";
cin>>colour>>Carno;
}
void putcar()
{
cout<<"\nEnter Car colour and car number\n\t"<<colour<<"\t\t"<<Carno;
}
};
void main()
{
Car c;
clrscr();
c.getcarm();
c.getcarmodel();
c.getcar();
c.putcarm();
c.putcarmodel();
c.putcar();
getch();
}
```

```
Enter Car Name Maruti
Enter Car Model Name and Model No. Swift 2021
Enter Car colour and car number White MH-01-1234

The Car Name is Maruti
Enter Car Model Name and Model No.
   Swift    2021
Enter Car colour and car number
   White    MH-01-1234
```

Winter-19
6 Marks Questions

1.Write a program to declare a class ‘student’ having data members as ‘stud_name’ and ‘roll_no’. Accept and display this data for 5 students. (Note: Any other correct logic shall be considered)

```
#include<iostream.h>
#include<conio.h>
class student
{
int roll_no;
char stud_name[20];
public:
void Accept();
void Display();
};
void student::Accept()
{
cout<<"\n Enter student‟s name and roll no\n";
cin>>stud_name>>roll_no;
}
void student::Display()
{
cout<<stud_name<<"\t"<<roll_no<<"\n";
}
void main()
{
student S[5];
inti;
clrscr();
for(i=0;i<5;i++)
{
S[i].Accept();
}
cout<<"Student details \n Student"s Name \t Roll No\n";

for(i=0;i<5;i++)
{
S[i].Display();
}
getch();
}
```

```
Enter student's name and roll no
John 101
Enter student's name and roll no
Alice 102
Enter student's name and roll no
Bob 103
Enter student's name and roll no
Eve 104
Enter student's name and roll no
Charlie 105
Student details
 Student's Name    Roll No
John    101
Alice   102
Bob     103
Eve     104
Charlie 105
```

2.State and explain the visibility modes used in inheritance.

Base class visibility	Derived class visibility		
	Private	Protected	Public
Private	Not Inherited	Not Inherited	Not Inherited
Protected	Private	Protected	Protected
Public	Private	Protected	Public

Private:
o When a base class is privately inherited by a derived class, „public members‟ and „protected members‟ of the base class become „private members‟ of the derived class.
o Therefore, the public and protected members of the base class can only be accessed by the member functions of derived class but, cannot be accessed by the objects of the derived class.
Syntax:
```
class derived: private base
{
//Members of derived class;
};
```

Public: o When a base class is publicly inherited by a derived class then „protected members‟ of base class becomes „protected members‟ and ‟public members‟ of the base class become „public members‟ of the derived class.

o Therefore the public members of the base class can be accessed by both the member functions of derived class as well
as the objects of the derived class.
Syntax:
```
class derived:
public base
{
//Members of derived class;
};
```

Protected:
o When a base class is protectedly inherited by a derived class, „public and protected members‟ of the base class become „protected members‟ of the derived class.
o Therefore the public and protected members of the base class can be accessed by the member functions of derived class as well as the member functions of immediate derived class of it but they cannot be accessed by the objects of derived class
Syntax:

```
class derived: protected base
{
//Members of derived class;
};
```

3.Write a program to declare a class 'book' containing data members as 'title', 'author-name', 'publication', 'price'. Accept and display the information for one object using pointer to that object. (Note: Any other correct logic shall be considered)

```
#include<iostream.h>
#include<conio.h>
class book
{
char author_name[20];
char title[20];
char publication[20];
float price; public:
void Accept();
void Display();
};
 void book::Accept()
{
cout<<"\n Enter book‟s title, author_name, publication and price \n:";
cin>> title >>author_name>> publication >> price;
}
void student::Display()
{
cout<<title <<”\t”<<author_name<<”\t”<<publication <<”\t”<<
price<<”\n”<<;
}
void main()
{
book b, *p;
clrscr();
p=&b;
p->Accept();
cout<<"title \t author_name \t publication \t price\n";
p-> Display();
getch();
}
```

```
Enter book's title, author name, publication, and price: The Catcher in the Rye J.D.
Salinger Little, Brown and Company 12.99
title   author_name     publication     price
The Catcher in the Rye   J.D. Salinger    Little, Brown and Company    12.99
```

4.Write a program that copies contents of one file into another file. (Note: Any other correct logic shall be considered)

Assuming input file to be copied file
1.txt contents are “Hello Friends…” and file where the contents need to copy is file
2.txt already created

```
#include<iostream.h>
#include<conio.h>
#include <string.h>
#include<fstream>
#include
void main()
{
clrscr();
ifstream fs;
ofstream ft;
char ch, fname1[20], fname2[20];
cout<<"Enter source file name with extension (like files.txt) : ";
gets(fname1);
fs.open(fname1);
if(!fs)
{
cout<<"Error in opening source file..!!";
getch();
exit(1);
}
cout<<"Enter target file name with extension (like filet.txt) : ";
gets(fname2);
ft.open(fname2);
if(!ft)
{
cout<<"Error in opening target file..!!";
fs.close();
getch();
exit(2);
}
while(fs.eof()==0)
{
fs>>ch;
ft<<ch;
}
cout<<"File copied successfully..!!";
fs.close();
ft.close();
getch();
}
```

```
Enter source file name with extension (like files.txt) : files.txt
Enter target file name with extension (like filet.txt) : filet.txt
File copied successfully..!!
```

5.Write a program to implement the following hierarchy using suitable member functions. Refer Figure No.2. (Note: Any other correct logic shall be considered)

```
#include <iostream.h>
#include<conio.h>
class Student
{
int roll_no;
char name[10];
public:
void read_studentData()
{
cout<<"Enter student‟s roll no and name \n";
cin>>roll_no>> name;
}
void display_studentData ()
{
cout<<"\n roll_no\t name\n";
cout<<roll_no<<"\t"<<name<<"\n";
}
};
class test: public Student
{
protected:
int marks1,marks2;
public:
void read_test()
{
cout<<"\n Enter test marks\n";
cin>>marks1>>marks2;
}
void display_test()
{
cout<<"\n test Marks \n Marks1 \t Marks2 \n";
cout<<marks1<<"\t"<<marks2;
}
};
class sports
{
int score;

public:
void read_sportsData()
{
cout<<"\n Enter sport score\n";
cin>> score;
}
void display_sportsData()
{
cout<<"\n sport score:"<<score;
}
};
```

```
class result: public test, public sports
{
int total;
public:
void read_result()
{
read_ studentData () ;
read_test();
read_sportsData();
total=marks1+marks2;
}
void display_result()
{
display_ studentData () ;
display_test();
display_sportsData();
cout<<"\n Total="<<total;
}
};
void main()
{
result r;
clrscr();
r.read_result();
r.display_result();
getch();
}
```

```
Enter student's roll no and name
101 John

Enter test marks
85 92

Enter sport score
78

Student's Data:
Roll No  Name
```

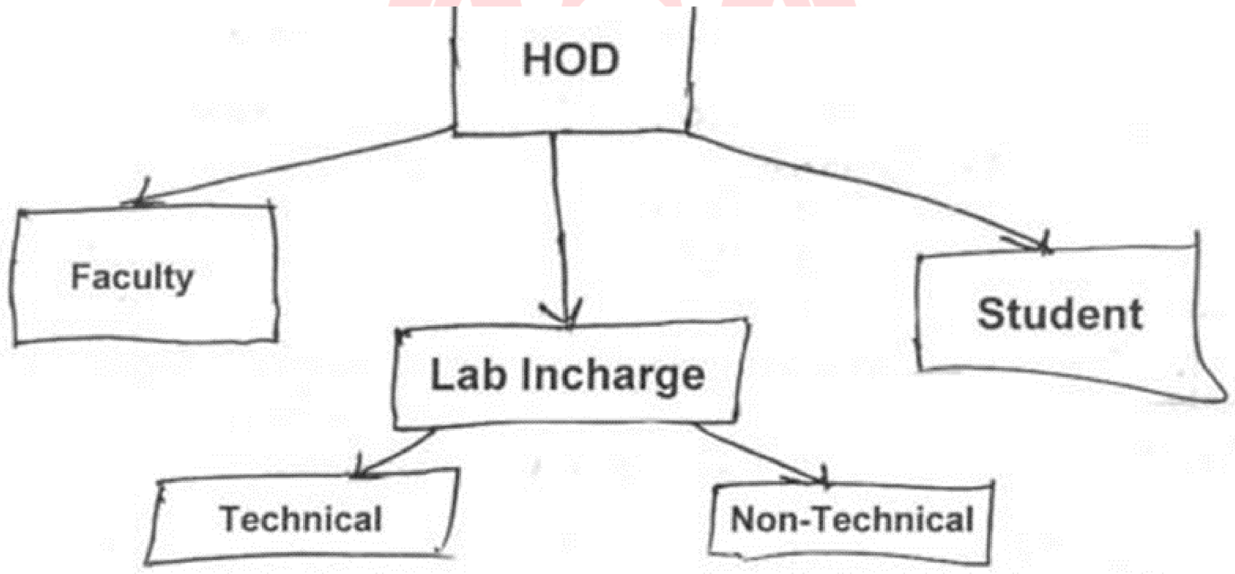
```
101     John

Test Marks:
Marks1  Marks2
85      92

Sports Score: 78
Total: 177
```

6.Write a program to overload the '—' unary operator to negate the values. (Note: Any other correct logic shall be considered)

```
#include<iostream.h>
#include<conio.h>
#include<string.h>
class Number
{
int x,y;
public:
Number (int a, int b)
{
a =x;
b =y;
}
void display()
{
cout<<"value of x=”<<x<<”\n Value of y= ”<<y;
}
void operator - ( )
{
x = - x;
y = - y;
}
};
void main ()
{
Number N1(5,6);
clrscr ();
N1. display ();
-N1;
cout<<"\n After negation:";
N1. display ();
getch ();
}
```

```
Value of x=0
Value of y=0
After negation:
Value of x=0
Value of y=0
```

7.Develop c++ program to implement inheritance shown in fig 6M



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

using namespace std;
```

```
// Base class HOD (Head of Department)
class HOD {
protected:
string department;

public:
void inputHODData()
{
cout << "Enter Department: ";
cin >> department;
}

void displayHODData()
{
cout << "Department: " << department << endl;
}
};

// Derived class Faculty
class Faculty : public HOD
 {
protected:
string facultyName;

public:
void inputFacultyData()
 {
   inputHODData(); // Input HOD data
   cout << "Enter Faculty Name: ";
   cin >> facultyName;
  }

  void displayFacultyData()
 {
    displayHODData(); // Display HOD data
    cout << "Faculty Name: " << facultyName << endl;
  }
};

// Base class LabIncharge
class LabIncharge
```

```
{
protected:
string labName;

public:
 void inputLabInchargeData()
{
  cout << "Enter Lab Name: ";
  cin >> labName;
 }

  void displayLabInchargeData()
{
        cout << "Lab Name: " << labName << endl;
}
};

// Derived class Technical
class Technical : public LabIncharge
{
protected:
string technicianName;

public:
void inputTechnicalData()
{
 inputLabInchargeData(); // Input LabIncharge data
 cout << "Enter Technician Name: ";
 cin >> technicianName;
 }

 void displayTechnicalData()
{
 displayLabInchargeData(); // Display LabIncharge data
 cout << "Technician Name: " << technicianName << endl;
}
};

// Derived class NonTechnical
class NonTechnical : public LabIncharge
{
```

```
protected:
string staffName;

public:
 void inputNonTechnicalData()
 {
 inputLabInchargeData(); // Input LabIncharge data
 cout << "Enter Staff Name: ";
  cin >> staffName
}

void displayNonTechnicalData()
{
 displayLabInchargeData(); // Display LabIncharge data
 cout << "Staff Name: " << staffName << endl;
}
};

// Derived class Student
class Student : public HOD
{
protected:
string studentName;

public:
void inputStudentData()
{
inputHODData(); // Input HOD data
cout << "Enter Student Name: ";
cin >> studentName;
}

void displayStudentData()
{
displayHODData(); // Display HOD data
cout << "Student Name: " << studentName << endl;
}
};

void main() {
    Faculty faculty;
    Technical technicalLab;
    NonTechnical nonTechnicalLab;
    Student student;

    cout << "Input Faculty Data:\n";
    faculty.inputFacultyData();

    cout << "\nInput Technical Lab Data:\n";
    technicalLab.inputTechnicalData();

    cout << "\nInput Non-Technical Lab Data:\n";
    nonTechnicalLab.inputNonTechnicalData();

    cout << "\nInput Student Data:\n";
    student.inputStudentData();

    cout << "\nFaculty Data:\n";
    faculty.displayFacultyData();

    cout << "\nTechnical Lab Data:\n";
    technicalLab.displayTechnicalData();

    cout << "\nNon-Technical Lab Data:\n";
    nonTechnicalLab.displayNonTechnicalData();

    cout << "\nStudent Data:\n";
    student.displayStudentData();
    getch();;
}
```

```
Input Faculty Data:
Enter Department: Computer Science
Enter Faculty Name: John Doe

Input Technical Lab Data:
Enter Lab Name: Physics Lab
Enter Technician Name: Jane Smith

Input Non-Technical Lab Data:
Enter Lab Name: Chemistry Lab
Enter Staff Name: David Johnson

Input Student Data:
Enter Department: Mathematics
Enter Student Name: Alice Brown

Faculty Data:
Department: Computer Science
Faculty Name: John Doe

Technical Lab Data:
Lab Name: Physics Lab
Technician Name: Jane Smith

Non-Technical Lab Data:
Lab Name: Chemistry Lab
Staff Name: David Johnson

Student Data:
Department: Mathematics
Student Name: Alice Brown
```

Winter-22
6 Marks Questions

1.Define class book with following Data member and member function for 10 book.
Data Member Member Function
1. B - name
→ getdata ( )
2. B - author
→ put data ( ) 3. B - price

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

class Book {
private:
    string name;
    string author;
    double price;

public:
    // Member function to get data for a book
    void getData() {
        cout << "Enter book name: ";
        cin.ignore(); // Ignore any previous newline character
        getline(cin, name);

        cout << "Enter author name: ";
        getline(cin, author);

        cout << "Enter book price: ";
        cin >> price;
    }

    // Member function to display data for a book
    void putData() {
        cout << "Book Name: " << name << endl;
        cout << "Author: " << author << endl;
        cout << "Price: $" << price << endl;
    }
};

int main() {
    const int numBooks = 10;
    Book books[numBooks]; // Declare an array of Book objects

    // Input: Get data for 10 books
    for (int i = 0; i < numBooks; ++i)
{
        cout << "Enter data for Book " << (i + 1) << ":" << endl;
        books[i].getData();
    }

    // Output: Display data for all 10 books
    cout << "\nBook Information:\n";
    for (int i = 0; i < numBooks; ++i)
{
        cout << "Book " << (i + 1) << ":\n";
        books[i].putData();
        cout << endl;
    }
    return 0;
}
```

```
Enter data for Book 1:
Enter book name: Book 1
Enter author name: Author 1
Enter book price: 15.99

Enter data for Book 2:
Enter book name: Book 2
Enter author name: Author 2
Enter book price: 12.99

Enter data for Book 3:
Enter book name: Book 3
Enter author name: Author 3
Enter book price: 9.99

Enter data for Book 4:
Enter book name: Book 4
Enter author name: Author 4
Enter book price: 22.50

Enter data for Book 5:
Enter book name: Book 5
Enter author name: Author 5
Enter book price: 17.95

Enter data for Book 6:
Enter book name: Book 6
Enter author name: Author 6
Enter book price: 30.00

Enter data for Book 7:
Enter book name: Book 7
Enter author name: Author 7
Enter book price: 8.99

Enter data for Book 8:
Enter book name: Book 8
Enter author name: Author 8
Enter book price: 14.75

Enter data for Book 9:
Enter book name: Book 9
Enter author name: Author 9
Enter book price: 19.99

Enter data for Book 10:
Enter book name: Book 10
Enter author name: Author 10
Enter book price: 25.00

Book Information:
Book 1:
Book Name: Book 1
Author: Author 1
Price: $15.99

Book 2:
Book Name: Book 2
Author: Author 2
Price: $12.99

Book 3:
Book Name: Book 3
Author: Author 3
Price: $9.99

Book 4:
Book Name: Book 4
Author: Author 4
Price: $22.50

Book 5:
```

```
Book Name: Book 5
Author: Author 5
Price: $17.95

Book 6:
Book Name: Book 6
Author: Author 6
Price: $30.00

Book 7:
Book Name: Book 7
Author: Author 7
Price: $8.99

Book 8:
Book Name: Book 8
Author: Author 8
Price: $14.75

Book 9:
Book Name: Book 9
Author: Author 9
Price: $19.99

Book 10:
Book Name: Book 10
Author: Author 10
Price: $25.00
```

2.Develop a c++ program to implement virtual Base class.

Ex1:-

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

// Base class Animal
class Animal {
public:
    void speak() {
        cout << "Animal speaks!" << endl;
    }
};

// Virtual Base class Pet (Virtual inheritance)
class Pet : virtual public Animal {
public:
    void play() {
        cout << "Pet plays!" << endl;
    }
};

// Derived class Dog
class Dog : virtual public Animal {
public:
    void bark() {
        cout << "Dog barks!" << endl;
    }
};

// Derived class PetDog (Hybrid virtual inheritance)
class PetDog : public Pet, public Dog {
public:
    void perform() {
        cout << "PetDog performs tricks!" << endl;
    }
};

int main() {
    PetDog myPet;

    cout << "My PetDog:\n";
    myPet.speak();   // Calls Animal's speak via Pet
    myPet.bark();    // Calls Dog's bark
    myPet.play();    // Calls Pet's play
    myPet.perform(); // Calls PetDog's perform

    return 0;
}
```

```
My PetDog:
Animal speaks!
Dog barks!
Pet plays!
PetDog performs tricks!
```

ex2:-

```
#include <iostream>
using namespace std;
class Publisher
{
string pname;
string place;
public:
void getdata()
{
cout<<"Enter name and place of publisher:"<<endl;
cin>>pname>>place;
}
void show ()
{
cout<<"Publisher Name:"<<pname<<endl;
cout<<"Place:"<<place<<endl;
}
};
class Author:virtual public Publisher
{
string aname;
public:
void getdata()
{
cout<<"Enter Author name:"<<endl; cin>>aname;
}
void show ()
{
cout<<"Author Name:"<<aname<<endl;
}
};
```

```
class Distributor:virtual public Publisher
{
string dname;
public:
void getdata()
{
cout<<"Enter Distributor name:"<<endl;
cin>>dname;
}
void show ()
{
cout<<"Distributor Name:"<<dname<<endl;
}
};
class Book:public Author, public Distributor
{
string title;
float price;
int pages;
public:
void getdata()
{
Publisher::getdata();
Author::getdata();
Distributor::getdata();
cout<<"Enter Book Title, Price and No. of pages"<<endl;
cin>>title>>price>>pages;
}
void show()
{
Publisher:: show (); Author::
show (); Distributor:: show ();
cout<<"Title:"<<title<<endl;
cout<<"Price:"<<price<<endl;
cout<<"No. of Pages:"<<pages<<endl;
}
};
int main() {
Book b;
b.getdata();
b.show();
return 0;
}
```

```
Enter name and place of publisher:
XYZ Publications
New York
Enter Author name:
John Doe
Enter Distributor name:
Distributor X
Enter Book Title, Price and No. of pages
Sample Book
29.99
300
Publisher Name: XYZ Publications
Place: New York
Author Name: John Doe
Distributor Name: Distributor X
Title: Sample Book
Price: 29.99
No. of Pages: 300
```

3.Explain rules of operator overloading and overload '+' operator to concatenate two string.

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

class MyString {
private:
    string str;

public:
    MyString(const string& s) : str(s) {}

    // Overload the + operator for string concatenation
    friend MyString operator+(const MyString& s1, const MyString& s2)
{
        return MyString(s1.str + s2.str);
    }
    void display() {
        cout << str << endl;
    }
};

int main() {
    MyString str1("Hello, ");
    MyString str2("world!");

    MyString result = str1 + str2; // Using overloaded operator+
    result.display(); // Output: Hello, world!
    return 0;
}
```

```
Hello, world!
```

4.Develop a c++ program for constructor with default argument and use of destructor.

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

class MyClass
{
private:
    int value;

public:
    // Constructor with default argument
    MyClass(int x = 0) {
        value = x;
        cout << "Constructor called with value: " << value << endl;
    }
    // Destructor
    ~MyClass() {
        cout << "Destructor called for value: " << value << endl;
    }

    void display() {
        cout << "Value: " << value << endl;
    }
};

int main() {
    // Create objects with and without arguments
    MyClass obj1;           // Default constructor (value = 0)
    MyClass obj2(42);       // Constructor with argument (value = 42)

    // Display values
    obj1.display();
    obj2.display();

    // Destructor will be called automatically when objects go out of scope
    return 0;
}
```

```
Constructor called with value: 0
Constructor called with value: 42
Value: 0
Value: 42
Destructor called for value: 0
Destructor called for value: 42
```

5.Describe Function overloading with suitable program.

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

// Function to add two integers
```

```
int add(int a, int b) {
    return a + b;
}

// Function to add two double numbers
double add(double a, double b) {
    return a + b;
}

// Function to concatenate two strings
string add(const string& str1, const string& str2) {
    return str1 + str2;
}

int main() {
    int result1 = add(5, 10);
    double result2 = add(3.5, 2.5);
    string result3 = add("Hello, ", "world!");

    cout << "Result 1: " << result1 << endl; // Output: Result 1: 15
    cout << "Result 2: " << result2 << endl; // Output: Result 2: 6
    cout << "Result 3: " << result3 << endl; // Output: Result 3: Hello, world!

    return 0;
}
```

```
Result 1: 15
Result 2: 6
Result 3: Hello, world!
```

Summer-22
6 Marks Questions

1.Define pointer operator and address operator with example.

In C++, the pointer operator and address operator are essential for working with pointers.

1. **Pointer Operator (`*`):**
   - The `*` operator is used to declare a pointer and to access the value stored at the address pointed to by the pointer.
   - It is also used in pointer arithmetic when you want to increment or decrement the pointer to access elements in an array or to move to the next memory location.

```
int main() {
    int num = 42;
    int* ptr = &num; // Declare a pointer and assign the address of 'num'

    // Access the value using the pointer operator
    cout << "Value of num: " << *ptr << endl; // Output: Value of num: 42

    return 0;
}
```

2. **Address Operator (`&`):**
   - The `&` operator is used to get the memory address (or pointer) of a variable.
   - It allows you to obtain the address of a variable, which you can then store in a pointer or use for other purposes.

```
int main() {
    int num = 42;

    // Use the address operator to get the memory address of 'num'
    int* ptr = &num;

    // Display the memory address of 'num'
    cout << "Memory address of num: " << ptr << endl;

    return 0;
}
```

Output:
```
Memory address of num:
0x7ffe65db36b8 (The address may vary)
```

`&` operator is used to obtain the memory address of the `num` variable, and that address is then stored in the `ptr` pointer.

The pointer operator (`*`) is used for dereferencing pointers and accessing the value stored at a specific memory address. The address operator (`&`) is used to get the memory address of a variable. These operators are fundamental when working with pointers in C++.

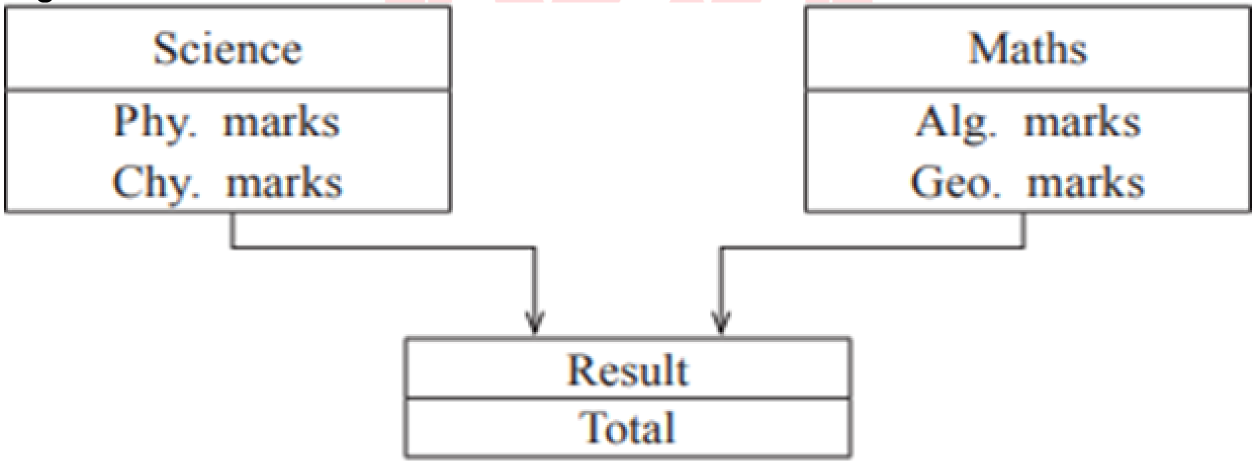
2.Write a C++ program to declare a class train with members as train no and name. Accept and display data for one object of train. Use pointer to object to call functions of class.

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

class Train
{
private:
    int trainNo;
    string trainName;

public:
    // Member function to accept data
    void acceptData() {
        cout << "Enter Train Number: ";
        cin >> trainNo;
        cin.ignore(); // Ignore any previous newline character
        cout << "Enter Train Name: ";
        getline(cin, trainName);
    }

    // Member function to display data
    void displayData() {
        cout << "Train Number: " << trainNo << endl;
        cout << "Train Name: " << trainName << endl;
    }
};

int main() {
    Train* ptrTrain = new Train(); // Create a pointer to a Train object

    // Call member functions using the pointer
    ptrTrain->acceptData();
    ptrTrain->displayData();

    // Deallocate memory for the object
    delete ptrTrain;
    return 0;
}
```

```
Enter Train Number: 123
Enter Train Name: Sample Train
Train Number: 123
Train Name: Sample Train
```

This program demonstrates how to use a pointer to an object of a class to call its member functions and access its data members.

3.**Write a C++ program to implement following inheritance: Refer Fig.** 6M



Accept and display total of one object of result.

```
#include <iostream.h>
#include<conio.h>
using namespace std;

// Base class for Physics and Chemistry marks
class Science {
```

```
protected:
    int phyMarks;
    int chyMarks;
public:
    void inputMarks() {
        cout << "Enter Physics Marks: ";
        cin >> phyMarks;
        cout << "Enter Chemistry Marks: ";
        cin >> chyMarks;
    }

    void displayMarksPC() {
        cout << "Physics Marks: " << phyMarks << endl;
        cout << "Chemistry Marks: " << chyMarks << endl;
    }
};

// Base class for Algorithm and Geometry marks
class Maths {
protected:
    int algMarks;
    int geoMarks;

public:
   void inputMarks()
 {
    cout << "Enter Algebra Marks: ";
    cin >> algMarks;
    cout << "Enter Geometry Marks: ";
    cin >> geoMarks;
 }

   void displayMarksAG()
 {
     cout << "Algebra Marks: " << algMarks << endl;
     cout << "Geometry Marks: " << geoMarks << endl;
 }
};

// Derived class that inherits from both Science and Maths
class Result : public Science, public Maths {
```

```
public:
    void inputTotalMarks()
 {
     Science::inputMarks(); // Input Physics and Chemistry marks
     Maths::inputMarks(); // Input algebra and Geometry marks
 }

    void displayTotalMarks()
 {
     displayMarksPC(); // Inherited function to display Physics and Chemistry marks
     displayMarksAG(); // Inherited function to display Algebra and Geometry marks
     int total = phyMarks + chyMarks + algMarks + geoMarks;
     cout << "Total Marks: " << total << endl;
 }
};

void main() {
  // Create a Result object
 Result studentResult;

  // Input marks for Physics, Chemistry, Algebra, and Geometry
  studentResult.inputTotalMarks();

  // Display the total marks
   studentResult.displayTotalMarks();
    getch();
}
```

```
Enter Physics Marks: 85
Enter Chemistry Marks: 78
Enter Algebra Marks: 92
Enter Geometry Marks: 88
Physics Marks: 85
Chemistry Marks: 78
Algebra Marks: 92
Geometry Marks: 88
Total Marks: 343
```

4.(Hint: class 1 contains ml and class 2 contains m2)
Write a C++ program to declare two classes with data members as ml and m2 respectively. Use friend function to calculate average of two (m1, m2) marks and display it.

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

// Forward declaration of Class2 to make it a friend of Class1
class Class2;

// Class1 declaration
class Class1 {
private:
    int m1;

public:
    Class1(int m) : m1(m) {}

    // Declare Class2 as a friend class to access its data members
    friend double calculateAverage(const Class1&, const Class2&);
};

// Class2 declaration
class Class2 {
private:
    int m2;

public:
    Class2(int m) : m2(m) {}

    // Declare Class1 as a friend class to access its data members
    friend double calculateAverage(const Class1&, const Class2&);
};

// Friend function to calculate the average of m1 and m2
double calculateAverage(const Class1& c1, const Class2& c2) {
    return (static_cast<double>(c1.m1) + c2.m2) / 2.0;
}

int main() {
    Class1 obj1(80); // Create an object of Class1 with m1 = 80
    Class2 obj2(90); // Create an object of Class2 with m2 = 90

    // Calculate and display the average using the friend function
    double average = calculateAverage(obj1, obj2);
    cout << "Average of m1 and m2: " << average << endl;

    return 0;
}
```

```
Average of m1 and m2: 85
```

This program demonstrates the use of friend functions to access and calculate values involving data members of two different classes.

5.Write any two characteristics of …static data member. Write C++ program to count number of objects created with the help of static data member.

Two characteristics of a static data member in C++ are:

1. Shared Among All Objects*: A static data member is shared among all objects of the class. It is not associated with any specific instance of the class but belongs to the class itself.

2. Single Storage Location: Unlike non-static (instance) data members, which have separate storage for each object, a static data member has a single storage location that is shared by all objects of the class. Changes made to a static data member affect all instances of the class.

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

class MyClass
{
private:
    static int objectCount; // Static data member to count objects
public:
   MyClass()
{
      objectCount++; // Increment object count each time an object is created
   }

   // Static member function to get the object count
   static int getObjectCount()
{
      return objectCount;
   }
};
// Initialize the static data member outside the class
int MyClass::objectCount = 0;

int main()
{
   MyClass obj1;
   MyClass obj2;
   MyClass obj3;

   cout << "Number of objects created: " << MyClass::getObjectCount() << endl;
   return 0;
}
```

```
Number of objects created: 3
```

When you run this program, it will display the number of objects created, which is incremented using the static data member `objectCount`.

6.Write a C++ program to overload "+" operator so that it will perform concatenation of two strings. (Use class get data function to accept two strings)

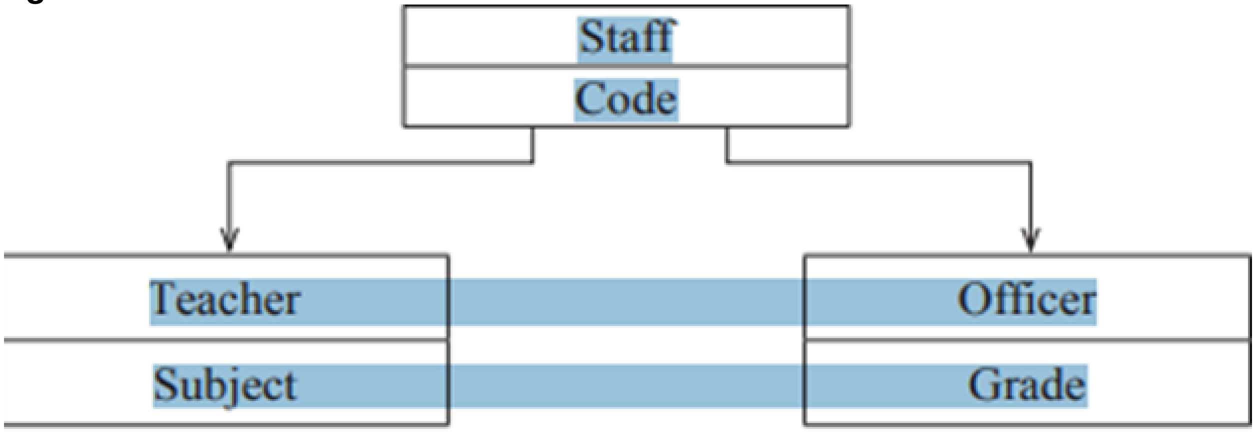
```
#include <iostream>
#include <string>
using namespace std;

class StringConcatenator
```

```
{
private:
    string str1;
    string str2;

public:
    // Member function to get data
    void getData() {
        cout << "Enter the first string: ";
        cin.ignore(); // Ignore any previous newline character
        getline(cin, str1);

        cout << "Enter the second string: ";
        getline(cin, str2);
    }

    // Overload the + operator for string concatenation
    StringConcatenator operator+(const StringConcatenator& other) {
        StringConcatenator result;
        result.str1 = this->str1 + other.str1;
        result.str2 = this->str2 + other.str2;
        return result;
    }

    // Member function to display data
    void displayData()
{
        cout << "Concatenated String 1: " << str1 << endl;
        cout << "Concatenated String 2: " << str2 << endl;
    }
};

int main()
{
    StringConcatenator obj1, obj2, result;

    // Get data for two objects
    cout << "Enter data for Object 1:" << endl;
    obj1.getData();

    cout << "Enter data for Object 2:" << endl;
    obj2.getData();

    // Perform string concatenation using the overloaded + operator
    result = obj1 + obj2;

    // Display the concatenated strings
    result.displayData();
    return 0;
}
```

```
Enter data for Object 1:
Enter the first string: Hello,
Enter the second string: World!

Enter data for Object 2:
Enter the first string: How are
Enter the second string: you?

Concatenated String 1: Hello, How are
Concatenated String 2: World! you?
```

This program demonstrates how to overload the `+` operator to concatenate two strings using a class with a `getData()` member function for input.

7.Write a program to implement inheritance as shown in figure 6M



Accept and display data of one Teacher and one Officer

Without using constructor:-

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

using namespace std;

class Staff
{
protected:
    int code;

public:
    void setCode(int c)
{
        code = c;
    }

    void displayStaff()
{
        cout << "Staff Code: " << code << endl;
    }
};

class Teacher : public Staff
{
private:
    string subject;

public:
    void setSubject(const string &sub)
{
        subject = sub;
    }

    void displayTeacher()
{
        displayStaff(); // Inherited function to display Staff code
        cout << "Subject: " << subject << endl;
    }
};

class Officer : public Staff
{
private:
    char grade;

public:
    void setGrade(char g)
{
        grade = g;
    }

    void displayOfficer()
{
        displayStaff(); // Inherited function to display Staff code
        cout << "Grade: " << grade << endl;
    }
};

void main()
{
    // Create objects for Teacher and Officer
    Teacher teacher;
    Officer officer;

    // Input data for Teacher
    int teacherCode;
    string subject;

    cout << "Enter Teacher's Code: ";
    cin >> teacherCode;
    cin.ignore(); // Clear newline character from the buffer
    cout << "Enter Teacher's Subject: ";
    getline(cin, subject);

    teacher.setCode(teacherCode);
    teacher.setSubject(subject);

    // Input data for Officer
    int officerCode;
    char grade;

    cout << "Enter Officer's Code: ";
    cin >> officerCode;
    cout << "Enter Officer's Grade: ";
    cin >> grade;

    officer.setCode(officerCode);
    officer.setGrade(grade);

    // Display data for Teacher and Officer
    cout << "\nTeacher's Data:\n";
    teacher.displayTeacher();

    cout << "\nOfficer's Data:\n";
    officer.displayOfficer();

    getch();
}
```

```
Enter Teacher's Code: 101
Enter Teacher's Subject: Mathematics
Enter Officer's Code: 201
Enter Officer's Grade: A

Teacher's Data:
Staff Code: 101
Subject: Mathematics

Officer's Data:
Staff Code: 201
Grade: A
```

Using constructor:-

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

```
using namespace std;

class Staff {
protected:
    int code;

public:
    Staff(int c) : code(c) {}

    void displayStaff()
{
        cout << "Staff Code: " << code << endl;
    }
};

class Teacher : public Staff
{
private:
    string subject;

public:
    Teacher(int c, const string &sub) : Staff(c), subject(sub) {}

    void displayTeacher()
{
        displayStaff(); // Inherited function to display Staff code
        cout << "Subject: " << subject << endl;
    }
};

class Officer : public Staff
{
private:
    char grade;

public:
    Officer(int c, char g) : Staff(c), grade(g) {}

    void displayOfficer()
{
        displayStaff(); // Inherited function to display Staff code
        cout << "Grade: " << grade << endl;
    }
};

void main() {
    // Input data for Teacher and Officer
    int teacherCode, officerCode;
    string subject;
    char grade;

    cout << "Enter Teacher's Code: ";
    cin >> teacherCode;
    cout << "Enter Teacher's Subject: ";
    cin.ignore();
    getline(cin, subject);

    cout << "Enter Officer's Code: ";
    cin >> officerCode;
    cout << "Enter Officer's Grade: ";
    cin >> grade;

    // Create objects for Teacher and Officer
    Teacher teacher(teacherCode, subject);
    Officer officer(officerCode, grade);

    // Display data for Teacher and Officer
    cout << "\nTeacher's Data:\n";
    teacher.displayTeacher();

    cout << "\nOfficer's Data:\n";
    officer.displayOfficer();

    getch();
}
```

```
Enter Teacher's Code: 101
Enter Teacher's Subject: Mathematics
Enter Officer's Code: 201
Enter Officer's Grade: A
```

```
Teacher's Data:
Staff Code: 101
Subject: Mathematics

Officer's Data:
Staff Code: 201
Grade: A
```