https://shikshamentor.com/object-oriented-programming-using-c-for-msbte-3k-scheme/

312304 - OOP Using C++ (Sem III)

As per MSBTE's K Scheme

CO / CM / IF

Unit 2:Functions and Constructors

Marks - 16

Sr.No	Questions	Marks	Year
1.	**Explain friend function with suitable example**	4	S-24
	Explain the friend function with proper example.(Repeat)	4	W-19

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.
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.

Characteristics:
- Friend function is not in the scope of the class to which it has been declared as friend.
- It is called without any object of class like a normal function.
- It cannot access the member names directly and has to use an object name and dot membership operator with each member name.
- It can be declared either in the public or the private part of a class without affecting its meaning.
- It has the objects as arguments.

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

```
#include<iostream>
using namespace std;
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;
}
int main()
{
A a;
B b;
a.accept();
b.accept();
swap(a,b);
}
```

2.	**Differentiate between constructor and destructor (any 4 points)**	**4**	**S-24**
	Differentiate between constructor and destructor.	**4**	**S-19**
	Differentiate between constructor and destructor in C++(Any four points)	**4**	**W-23**
	State the difference between constructor and destructor. (any six points)	**6**	**S-23**

S.No.	Constructor	Destructor
1	Constructor helps to initialize the object of a class.	Whereas destructor is used to destroy the instances.
2	It is declared as **className(arguments if any){Constructor's Body }**.	Whereas it is declared as ~ className(no arguments){ }.
3	Constructor can either accept arguments or not.	While it can't have any arguments.
4	A constructor is called when an instance or object of a class is created.	It is called while object of the class is freed or deleted.
5	Constructor is used to allocate the memory to an instance or object.	While it is used to deallocate the memory of an object of a class.
6	Constructor can be overloaded.	While it can't be overloaded.
7	The constructor's name is same as the class name.	Here, its name is also same as the class name preceded by the tiled (~) operator.
8	In a class, there can be multiple constructors.	While in a class, there is always a single destructor.
9	There is a concept of copy constructor which is used to initialize an object from another object.	While here, there is no copy destructor concept.
10	They are often called in successive order.	They are often called in reverse order of constructor.

3. **Develop a C++ program using parameterized constructor** 6 **S-24**

```
#include<iostream>
```

	using namespace std;
	class number
	{
	int x;
	public:
	number(int y)
	{
	x=y;
	}
	void display( )
	{
	cout<<"The square of number is:"<<x*x;
	}
	};
	int main( )
	{
	number n(50);
	n.display( );
	}		

4.	**Explain inline member function**	**2**	**W-23**
	An inline function in C++ is a function that the compiler attempts to replace with its actual code whenever the function is called. This can potentially improve the performance of your program by reducing the overhead associated with function calls. Syntax: inline <return type> function_name(parameter list) {		

	` // function body`
	`}`
	Example:
	`inline int square(int x) {`
	` return x * x;`
	`}`
	`int main() {`
	` int y = square(5);`
	` // ...`
	`}`		

5.	**Explain the characteristics of friend function.**	**4**	**W-23**
	1) It is not in the scope of the class to which it has been declared as friend. 2) Since it is not in the scope of a class, it cannot be called using the object of that class. 3) It can be invoked like a normal function without the help of any object. 4) Unlike member functions, it cannot access the member names directly and has to use an object name and dot membership operator with each member name. 5) It can be declared either in public or the private part of a class without affecting its meaning. 6) It has the objects as arguments. 7) The friend function should not be defined inside the class.		
6.	**Write a program to declare a class measure having data members add1,add2 and add3. Initialize the data members using constructor and store their addition in third data member using function and display the addition.**	**4**	**W-23**

```
#include<iostream>
#include<conio.h>
using namespace std;
class measure
{
private:
int add1,add2,add3;
public:
measure()
{
add1=10;
add2=20;
}
void sum()
{
add3=add1+add2;
}
void display()
{
cout<<"\nSum = "<<add3;
}
};
int main()
{
```

```
measure m();
measure m1;
m1.sum();
m1.display();
getch();
return 0;
}
```

OR

//Using Parameterized Constructor

```
#include<iostream>
#include<conio.h>
using namespace std;
class measure
{
private:
int add1, add2, add3;
public:
measure(int a,int b)
{
add1=a;
add2=b;
}
void sum()
{
```

```
add3=add1+add2;
}
void display()
{
cout<<"\nSum = "<<add3;
}
};
int main()
{
measure m(10,20);//if parameterized constructor is used
m.sum();
m.display();
getch();
return 0;
}
```

7.	**Write a program to declare class ‘employee’ containing data members ‘emp-id’ and salary. Accept and display the data for 10 employee.**	**6**	**W-23**

```
#include <iostream>
using namespace std;
class employee {
private:
int emp_id;
double salary;
public:
void setData(int id, double sal) {
```

```
emp_id = id;
salary = sal;
}
void displayData() {
cout << "Employee ID: " << emp_id << ", Salary: " << salary << endl;
}
};
int main()
{
Employee employees[10];
// Accepting data for 10 employees
for (int i = 0; i <10; ++i) {
int empId;
double salary;
cout << "Enter details for Employee " << i + 1 << std::endl;
cout << "Enter Employee ID: ";
cin >> empId;
cout << "Enter Salary: ";
cin >> salary;
employees[i].setData(empId, salary);
}
// Displaying data for 10 employees
cout << "\nDetails of Employees:" <<endl;
for (int i = 0; i <10; ++i) {
std::cout << "Employee " << i + 1 << ": ";
employees[i].displayData();
```

	}
	return 0;
	}		

8.	**State the characteristics of static member function**	**2**	**S-23**
	Write two properties of static member function.(Repeat)	**2**	**W-19**
	Characteristics of Static Member Functions in C++ Static member functions are special functions associated with a class rather than individual objects. They have the following characteristics: 1. Associated with the Class They belong to the class itself, not to specific objects of the class. They can be accessed using the class name and the scope resolution operator (::). 2. No this pointer Static member functions do not have access to the this pointer. This means they cannot access non-static members (data or functions) of the class. 3. Can access static members They can access static data members and other static member functions of the class. 4. Can be called without creating an object They can be invoked directly using the class name, even if no objects of the class exist. 5. Can be declared as const Static member functions can be declared as const to indicate that they do not modify the class's state. 6. Cannot be virtual Static member functions cannot be declared as virtual. Polymorphism		

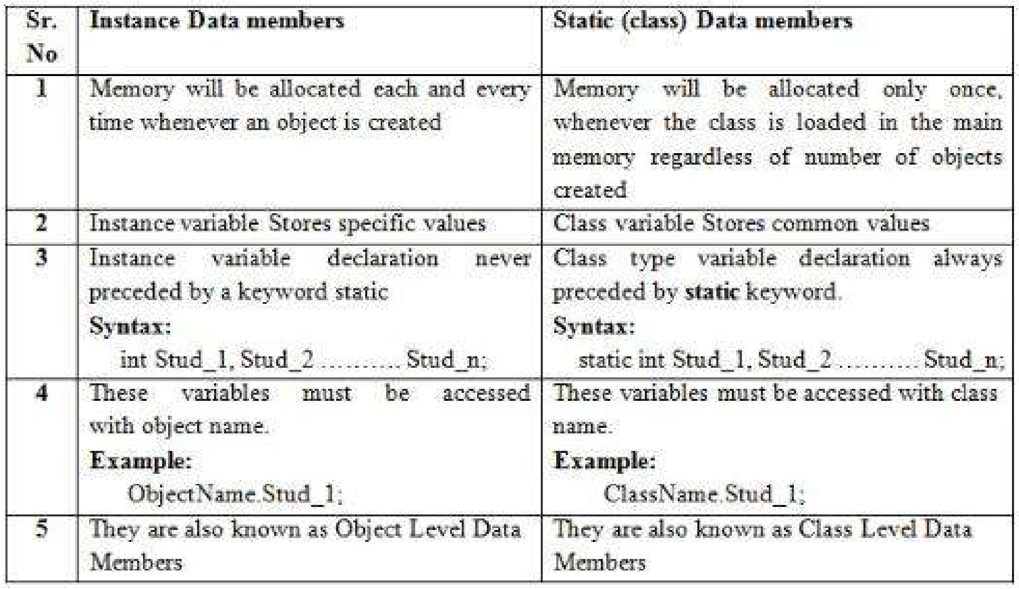
	is not applicable to static members. 7. Used for class-level operations They are often used for operations that are related to the class as a whole, rather than to specific objects.		
9.	**Give the syntax for constructor in derived classes.**	**2**	**S-23**
	```
	#include <iostream>
	// Base class
	class Base {
	public:
	    Base() {
	        std::cout << "Base constructor\n";
	    }
	};
	// Derived class
	class Derived : public Base {
	public:
	    Derived() {
	        std::cout << "Derived constructor\n";
	    }
	};
	int main() {
	    Derived d; // Creating an object of the derived class
	    return 0;
	}
	```
10.	**Explain overloaded constructor with suitable example.**	**4**	**S-23**
	1. Overloaded constructors essentially have the same name (exact		

name of the class) and different by number and type of arguments.

2. A constructor is called depending upon the number and type of arguments passed.
3. While creating the object, arguments must be passed to let compiler know, which constructor needs to be called.

```
// C++ program to illustrate
// Constructor overloading
#include <iostream>
using namespace std;
class construct
{
public:
        float area;
        // Constructor with no parameters
        construct()
        {
                area = 0;
        }
        // Constructor with two parameters
        construct(int a, int b)
        {
                area = a * b;
        }
        void disp()
        {
                cout<< area<< endl;
        }
};
int main()
{       // Constructor Overloading
        // with two different constructors
        // of class name
        construct o;
        construct o2( 10, 20);
        o.disp();
        o2.disp();
        return 1;
}
```

11.	**Compare static and non-static data members. (any four points)**	**4**	**S-23**



Sr. No	Instance Data members	Static (class) Data members
1	Memory will be allocated each and every time whenever an object is created	Memory will be allocated only once, whenever the class is loaded in the main memory regardless of number of objects created
2	Instance variable Stores specific values	Class variable Stores common values
3	Instance variable declaration never preceded by a keyword static **Syntax:** int Stud_1, Stud_2 .... Stud_n;	Class type variable declaration always preceded by **static** keyword. **Syntax:** static int Stud_1, Stud_2 .... Stud_n;
4	These variables must be accessed with object name. **Example:** ObjectName.Stud_1;	These variables must be accessed with class name. **Example:** ClassName.Stud_1;
5	They are also known as Object Level Data Members	They are also known as Class Level Data Members

12.	**List any four properties of constructor function.**	**4**	**S-23**

◆ The constructor name is the same as the Class Name. ...

	◆ A constructor must not declare a return type or void. ... ◆ They can be defined inside or outside the class definition. ◆ Automatically calls when an object is created for the class.		
13.	**Write a program to show object as function argument.**	**6**	**S-23**

```
// C++ program to show passing of objects to a function
#include <iostream>
using namespace std;
class Example {
public:
        int a;
        // This function will take an object as an argument
        void add(Example E)
        {
                a = a + E.a;
        }
};
// Driver Code
int main()
{       // Create objects
        Example E1, E2;
        // Values are initialized for both objects
        E1.a = 50;
        E2.a = 100;
        cout << "Initial Values \n";
        cout << "Value of object 1: " << E1.a
```

```
             << "\n& object 2: " << E2.a
             << "\n\n";
        // Passing object as an argument
        // to function add()
        E2.add(E1);
        // Changed values after passing
        // object as argument
        cout << "New values \n";
        cout << "Value of object 1: " << E1.a
             << "\n& object 2: " << E2.a
             << "\n\n";
        return 0;
}
```

14.	**Define constructors and it's type.**	**2**	**W-22**
	Define constructor. List types of constructor	**2**	**S-22**
	Constructors in C++ are the member functions that get invoked when an object of a class is created. There are mainly three types of constructors in C++, 1. Default 2. Parameterized 3. Copy constructors.		
15.	**Develop a c++ program for accept data from user to calculate percentage for 5 subject and display grade according to percentage.**	**4**	**W-22**
	#include <iostream>		

```
using namespace std;
int main() {
    float sub1, sub2, sub3, sub4, sub5, total, percentage;
    // Input marks for 5 subjects
    cout << "Enter marks for 5 subjects: ";
    cin >> sub1 >> sub2 >> sub3 >> sub4 >> sub5;
    // Calculate total marks
    total = sub1 + sub2 + sub3 + sub4 + sub5;
    // Calculate percentage
    percentage = (total / 500) * 100;
    // Determine grade based on percentage
    char grade;
    if (percentage >= 90) {
        grade = 'A+';
    } else if (percentage >= 80) {
        grade = 'A';
    } else if (percentage >= 70) {
        grade = 'B+';
    } else if (percentage >= 60) {
        grade = 'B';
    } else if (percentage >= 50) {
        grade = 'C';
    } else if (percentage >= 40) {
        grade = 'D';
    } else {
        grade = 'F';
    }
    // Display results
    cout << "Total marks: " << total << endl;
    cout << "Percentage: " << percentage << "%" << endl;
    cout << "Grade: " << grade << endl;
    return 0;
}
```

16.	**Explain with suitable example Friend Function.**	**4**	**W-22**

Friend Functions in C++

A friend function is a special function that is not a member of a class but has the privilege to access the private and protected members of that class. It's declared within the class using the friend keyword.

Why use friend functions?

To access private members of multiple classes.

To provide utility functions that operate on class objects without being class members.

For operator overloading (which we'll discuss in another context).

```
#include <iostream>
class Box {
private:
    int length;
    int breadth;
    int height;
public:
    Box(int l, int b, int h) {
        length = l;
        breadth = b;
        height = h;
    }
    friend int volume(Box b); // Friend function declaration
};
int volume(Box b) {
    return b.length * b.breadth * b.height;
}
int main() {
    Box box(3, 4, 5);
    std::cout << "Volume of the box is: " << volume(box) << std::endl;
    return 0;
}
```

17.	**Develop a c++ program for constructor with default argument and use of destructor.**	**6**	**W-22**

```
#include <iostream>
using namespace std;
class MyClass {
public:
    int x;
    // Constructor with default argument
    MyClass(int val = 10) {
        x = val;
        cout << "Constructor called with value: " << x << endl;
    }
    // Destructor
    ~MyClass() {
        cout << "Destructor called for object with value: " << x << endl;
    }
};
int main() {
    MyClass obj1; // Using default argument
    MyClass obj2(20); // Passing a value
    return 0;
}
```

18.	**Write any two characteristics of friend function.**	**2**	**S-22**
	Two Characteristics of Friend Functions in C++ **Access to Private Members:** Friend functions can access private and protected members of a class, unlike ordinary functions which are restricted to public members. This ability to bypass encapsulation is a key feature of friend functions. **Not a Member of the Class:** Despite having access to private members, a friend function is not considered a member of the class. It is declared within the class using the friend keyword but defined outside the class. This means it cannot be called using the dot operator on an object.		
19.	**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**	**4**	**S-22.S-19**

class. Display the data.

```
#include <iostream>

#include <string>

using namespace std;

class Student {

public:

    int roll_no;

    string name;

    // Constructor to initialize data members

    Student(int r, string n) {

        roll_no = r;

        name = n;

    }

    // Function to display student data

    void display() {

        cout << "Roll No: " << roll_no << endl;

        cout << "Name: " << name << endl;

    }

};

int main() {

    int r;

    string n;

    cout << "Enter roll number: ";

    cin >> r;

    cout << "Enter name: ";
    cin >> n;
    Student s(r, n); // Create a student object
    s.display(); // Display student data
    return 0;
}
```

20.	**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.**	**4**	**S-22**

```
#include <iostream>
#include <string>
using namespace std;
class Mobile {
public:
    int price;
    string model_number;
    Mobile(int p, string m) {
        price = p;
        model_number = m;
    }
    void display() {
        cout << "Price: " << price << endl;
        cout << "Model Number: " << model_number << endl;
    }
};
int main() {
    Mobile mobiles[10];
    int p;
    string m;
    for (int i = 0; i < 10; i++) {
        cout << "Enter price for mobile " << i + 1 << ": ";
        cin >> p;
        cout << "Enter model number for mobile " << i + 1 << ": ";
        cin >> m;
        mobiles[i] = Mobile(p, m);
    }
    cout << "\nMobile Details:\n";
    for (int i = 0; i < 10; i++) {
        cout << "Mobile " << i + 1 << ":" << endl;
        mobiles[i].display();
        cout << endl;
    }
    return 0;
}
```

21.	**Describe constructor with default arguments with an example.(Repeat)**	**4**	**S-22**

Constructors with Default Arguments

A constructor with default arguments allows you to provide default values for parameters. This means you can create objects of a class without specifying values for all parameters, using the provided default values instead.

```
class MyClass {
public:
    MyClass(int a, int b = 10, int c = 20) {
        // Constructor body
    }
};
```

Example:

```
#include <iostream>
class Rectangle {
public:
    int length, breadth;
    Rectangle(int l, int b = 5) {
        length = l;
        breadth = b;
    }
    int area() {
        return length * breadth;
    }
};
int main() {
    Rectangle rect1(4, 6); // Both parameters are provided
    Rectangle rect2(3);     // Only one parameter is provided, b will be 5
    Rectangle rect3(2);     // Only one parameter is provided, b will be 5
    std::cout << rect1.area() << std::endl;
    std::cout << rect2.area() << std::endl;
    std::cout << rect3.area() << std::endl;
    return 0;
}
```

22.	**(Hint : class 1 contains m1 and class 2 contains m2) Write a C++ program to declare two classes with data members as m1 and m2 respectively. Use friend function to calculate average of two (m1, m2) marks and display it.**	**6**	**S-22**

```
#include <iostream>
using namespace std;
class Class2;
class Class1 {
private:
   int m1;
public:
   Class1(int m) : m1(m) {}
   friend float average(Class1 obj1, Class2 obj2);
};
class Class2 {
private:
   int m2;
public:
   Class2(int m) : m2(m) {}
   friend float average(Class1 obj1, Class2 obj2);
};
float average(Class1 obj1, Class2 obj2) {
   return (obj1.m1 + obj2.m2) / 2.0;
}
int main() {
   Class1 obj1(80);
```

```
    Class2 obj2(90);

    float avg = average(obj1, obj2);

    cout << "Average: " << avg << endl;

    return 0;

}
```

23.	**Write any two characteristics of static data member. Write C++ program to count number of objects created with the help of static data member.**	**6**	**S-22**

Two Characteristics of Static Data Member

Shared by all objects: A static data member is shared by all objects of a class. There's only one copy of the static member, regardless of the number of objects created.

Initialized outside the class: A static data member must be defined and initialized outside the class, usually in the global scope. This is because static members belong to the class itself, not to individual objects.

```
#include <iostream>

using namespace std;

class MyClass {

public:

    static int count; // Static data member to count objects

    MyClass() {

        count++; // Increment count when an object is created

    }

    static void displayCount() {

        cout << "Number of objects created: " << count << endl;

    }

};
```

	int MyClass::count = 0; // Initialization of static member
	
	int main() {
	
	MyClass obj1, obj2, obj3;
	
	MyClass::displayCount(); // Display the count of objects
	
	return 0;
	
	}		

24.	**State the rules for writing destructor function**	**4**	**W-19**
	Rules for Writing Destructor Functions in C++ ● Name: The destructor has the same name as the class, preceded by a tilde (~). For example, for a class named MyClass, the destructor would be ~MyClass(). ● Arguments and Return Type: A destructor takes no arguments and returns no value. ● Access Specifier: Destructors are typically declared in the public section of a class, but they can also be declared in private or protected sections. ● Automatic Invocation: The destructor is called automatically when an object goes out of scope or is explicitly deleted using the delete operator. ● No Overloading: A class can have only one destructor. Overloading destructors is not allowed. ● Order of Destruction: Destructors are called in the reverse order of object creation. ● Virtual Destructors: For polymorphic classes (classes with virtual functions), it's recommended to declare the destructor as virtual to ensure correct destruction of derived class objects.		
25.	**What is parameterized constructor?**	**4**	**W-19**
	A parameterized constructor is a special member function of a class that accepts arguments when an object of the class is created. It's used to initialize the object's data members with specific values provided as arguments.		

Key points:

It has the same name as the class.

It doesn't have a return type.

It can accept any number of arguments.

It is used to initialize object members with specific values.

```
#include <iostream>
using namespace std;
class Rectangle {
public:
    int length;
    int breadth;
    // Parameterized constructor
    Rectangle(int l, int b) {
        length = l;
        breadth = b;
    }
    int area() {
        return length * breadth;
    }
};
int main() {
    Rectangle rect(5, 10); // Creating an object with values
    cout << "Area of rectangle: " << rect.area() << endl;
    return 0;
}
```

26.	**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.**	**6**	**W-19**

```
#include <iostream>
#include <string>
using namespace std;
class Student {
public:
    string stud_name;
    int roll_no;
    Student(string name, int roll) {
        stud_name = name;
        roll_no = roll;
    }
    void display() {
        cout << "Student Name: " << stud_name << endl;
        cout << "Roll No: " << roll_no << endl;
    }
};
int main() {
    Student students[5];
    string name;
    int roll;
    for (int i = 0; i < 5; i++) {
        cout << "Enter student " << i + 1 << " name: ";
        cin >> name;
        cout << "Enter student " << i + 1 << " roll number: ";
        cin >> roll;
        students[i] = Student(name, roll);
    }
    cout << "\nStudent Details:\n";
    for (int i = 0; i < 5; i++) {
        cout << "Student " << i + 1 << ":" << endl;
        students[i].display();
        cout << endl;
    }
    return 0;
}
```

27.	**Describe use of static data member.**	**2**	**S-19**
	Static Data Member A static data member is a class member that is shared by all objects of a class, rather than being unique to each object. There's only one copy of the static data member for the entire class, regardless of how many objects are created. Characteristics: Shared by all objects: All instances of the class share the same copy of the static data member. Initialized outside the class: Unlike regular data members, static data members must be defined and initialized outside the class definition. Accessed using class name: You can access a static data member using the class name and the scope resolution operator (::). Lifetime: The static data member's lifetime is the entire program's execution.		
28.	**Write a C++ program to find smallest number from two numbers using friend function. (Hint : use two classes).**	**4**	**S-19**

```
#include <iostream>
using namespace std;
class ClassB;
class ClassA {
private:
   int num1;
public:
   ClassA(int n) : num1(n) {}
   friend int findSmallest(ClassA obj1, ClassB obj2);
};
class ClassB {
private:
   int num2;
public:
   ClassB(int n) : num2(n) {}
   friend int findSmallest(ClassA obj1, ClassB obj2);
};
int findSmallest(ClassA obj1, ClassB obj2) {
   return (obj1.num1 < obj2.num2) ? obj1.num1 : obj2.num2;
}
int main() {
   int num1, num2;
   cout << "Enter two numbers: ";
   cin >> num1 >> num2;
   ClassA obj1(num1);
```

```
    ClassB obj2(num2);

    int smallest = findSmallest(obj1, obj2);

    cout << "Smallest number: " << smallest << endl;

    return 0;

}
```

29.	**Write a C++ program to declare a class student with members as roll no, name and department. Declare a parameterised constructor with default value for department as ‘CO’ to initialize members of object. Initialize and display data for two students.**	**6**	**S-19**

```
#include <iostream>

#include <string>

using namespace std;

class Student {

public:

    int roll_no;

    string name;

    string department;

    // Parameterized constructor with default value for department

    Student(int r, string n, string d = "CO") {

        roll_no = r;

        name = n;

        department = d;

    }

    void display() {

        cout << "Roll No: " << roll_no << endl;

        cout << "Name: " << name << endl;
        cout << "Department: " << department << endl;
    }
};
int main() {
    int roll;
    string name, dept;

    // Student 1 with default department
    cout << "Enter roll number for student 1: ";
    cin >> roll;
    cout << "Enter name for student 1: ";
    cin >> name;
    Student s1(roll, name);
    // Student 2 with specified department
    cout << "\nEnter roll number for student 2: ";
    cin >> roll;
    cout << "Enter name for student 2: ";
    cin >> name;
    cout << "Enter department for student 2: ";
    cin >> dept;
    Student s2(roll, name, dept);
    cout << "\nStudent 1 Details:\n";
    s1.display();
    cout << "\nStudent 2 Details:\n";
    s2.display();
```

	return 0;		
	}		
30.	**Write any two characteristics of destructor.**	**2**	**W-18**
	Two characteristics of a destructor:		
	Automatically invoked: A destructor is called automatically when an object goes out of scope or is explicitly deleted using the delete operator. It is not called manually.		
	No return type or parameters: A destructor has no return type and cannot accept any parameters. Its sole purpose is to perform cleanup operations before an object is destroyed.		
31.	**Give output for following code:**	**2**	**W-18**
	class student		
	{		
	int roll no;		
	char name [14];		
	} s[6];		
	void main( )		
	{ cout < <sixeof(s);		
	}		
	108		
32.	**Describe use of static data member in C++ with example.**	**4**	**W-18**
	Static Data Member in C++		
	A static data member is a class member that is shared by all objects of a class. There's only one copy of the static member, regardless of the number of objects created. It's declared within the class using the static keyword but is initialized outside the class.		
	Characteristics:		
	● Shared by all objects of a class.		

	● Only one copy exists for the entire class. ● Initialized outside the class. ● Can be accessed using the class name or an object of the class. ● Lifetime is the entire program.		
33.	**Write a C++ program to find greatest number among two numbers from two different classes using friend function.**	**4**	**W-18**

```
#include <iostream>
using namespace std;
class ClassB;
class ClassA {
private:
   int num1;
public:
   ClassA(int n) : num1(n) {}
   friend int findGreatest(ClassA obj1, ClassB obj2);
};
class ClassB {
private:
   int num2;
public:
   ClassB(int n) : num2(n) {}
   friend int findGreatest(ClassA obj1, ClassB obj2);
};
int findGreatest(ClassA obj1, ClassB obj2) {
   return (obj1.num1 > obj2.num2) ? obj1.num1 : obj2.num2;
}
int main() {
   int num1, num2;
   cout << "Enter two numbers: ";
   cin >> num1 >> num2;
   ClassA obj1(num1);
   ClassB obj2(num2);
   int greatest = findGreatest(obj1, obj2);
   cout << "Greatest number: " << greatest << endl;
   return 0;
}
```

34.	**Write a C++ program to accept array of five elements, find and display smallest number from an array.**	**4**	**W-18**

```
#include <iostream>
using namespace std;
int main() {
    int arr[5];
    cout << "Enter 5 elements: ";
    for (int i = 0; i < 5; i++) {
        cin >> arr[i];
    }
    int smallest = arr[0];
    for (int i = 1; i < 5; i++) {
        if (arr[i] < smallest) {
            smallest = arr[i];
        }
    }
    cout << "Smallest number: " << smallest << endl;
    return 0;
}
```

Thank You

https://shikshamentor.com/object-oriented-programming-using-c-for-msbte-3k-scheme/

Visit

https://shikshamentor.com/

SHIKSHA MENTOR
THE PERFECT MENTOR FOR YOUR BRIGHT CAREER