

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No:1.1

Develop a C++ Program to Find Factorial of a Given Number Using Recursion.

Aim: To develop a program to find the factorial of a given number using recursion.

Description: Recursion is a process in which a function calls itself to solve smaller sub-problems of a bigger problem.

It is commonly used in tasks that can be divided into similar sub-tasks — like Factorial, Fibonacci, Tree Traversals, Backtracking, etc.

Syntax:

```
return_type function_name(parameters)
{
    if (base_case_condition)
    {
        // base case: stop recursion
        return some_value;
    }
    else
    {
        // recursive case: function calls itself
        return function_name(modified_parameters);
    }
}
```

Program:

```
#include <iostream>
using namespace std;
int fact(int n);
int main()
{
    int n;
    cout << "Roll No: 24B11AI138" << endl;
    cout << "Enter n value: ";
    cin >> n;
    int result = fact(n);
    if (result == -1)
        cout << "Factorial is not defined" << endl;
    else
        cout << "Factorial: " << result << endl;
    return 0;
}
```

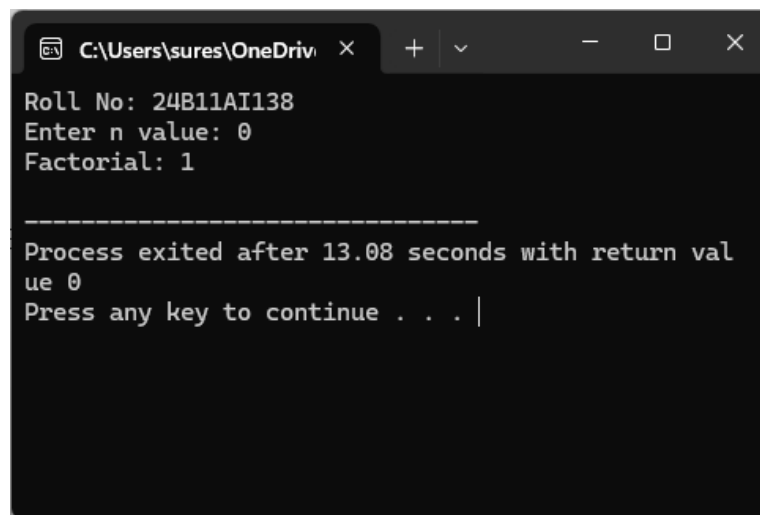
Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
int fact(int n)
{
    if (n < 0)
        return -1;
    else if (n == 0 || n == 1)
        return 1;
    else
        return n * fact(n - 1);
}
```

Output:

A screenshot of a Windows command prompt window. The title bar shows the file path 'C:\Users\sures\OneDrive'. The window contains the following text: 'Roll No: 24B11AI138', 'Enter n value: 0', 'Factorial: 1', a separator line of dashes, 'Process exited after 13.08 seconds with return value 0', and 'Press any key to continue . . . |'.

U N I V E R S I T Y

CO Mapped: CO1

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No :1.2

Develop a C ++ to Demonstrate Call by value and Call by Reference .

Aim: To develop a C ++ to Demonstrate Call by value and Call by Reference .

Description :

Call by Value: A copy of the actual argument is passed to the function. Changes made inside the function don't affect the original variable.

Syntax :

```
void modify(int x)
{
    x = x + 10;
}
```

Call by Reference: The address of the actual argument is passed. Changes inside the function directly affect the original variable.

Syntax:

```
void modify(int &x) {
    x = x + 10;
}
```

Program :

```
#include <iostream>
using namespace std;
// Function to swap two values using references
void swapValues(int &first, int &second)
{
    int temp = first;
    first = second;
    second = temp;
}
```

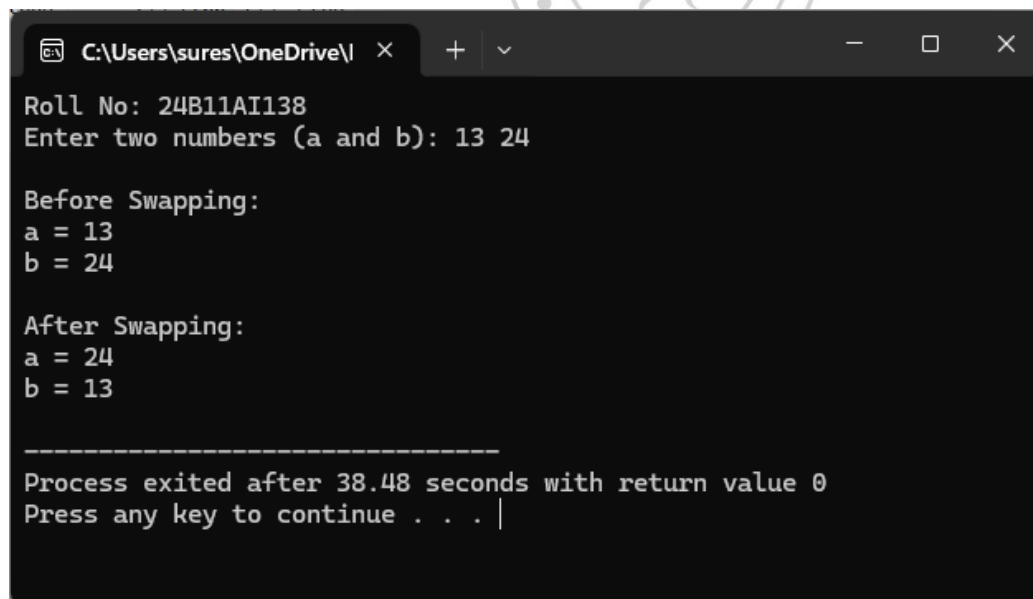
Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
int main()
{
    cout << "Roll No: 24B11AI138" << endl;
    int a, b;
    cout << "Enter two numbers (a and b): ";
    cin >> a >> b;
    cout << "\nBefore Swapping:\n";
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    swapValues(a, b); // Swapping the values
    cout << "\nAfter Swapping:\n";
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    return 0;
}
```

Output_:



The screenshot shows a Windows command prompt window with the title bar "C:\Users\sures\OneDrive\l". The output of the program is as follows:

```
Roll No: 24B11AI138
Enter two numbers (a and b): 13 24

Before Swapping:
a = 13
b = 24

After Swapping:
a = 24
b = 13

-----
Process exited after 38.48 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO1

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No :2.1

Develop a C++ program that demonstrates the use of the scope resolution operator and namespaces

Aim : To develop a C++ program that demonstrates the use of the scope resolution operator and namespaces.

Description :

a)Global and Local Variables :

Variables declared outside all functions are global and accessible throughout the program. Variables declared inside a function or block are local and only accessible within that scope .

Syntax :

```
int globalVar = 10;  // Global

void fun() {

    int localVar = 5;  // Local

}
```



b)Scope Resolution Operator (::) :

The :: operator is used to access global variables when there's a local variable with the same name, or to access members from a specific scope like a namespace.

Syntax :

```
int x = 50;  // Global

void fun() {

    int x = 10;

    cout << ::x;  // Refers to global x

}
```

c)Namespaces :

Namespaces help avoid name conflicts by encapsulating identifiers . Members of a namespace can be accessed using the scope resolution operator.

Syntax:

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
namespace MyNamespace {  
    int value = 100;  
}  
  
cout << MyNamespace::value; // Accessing namespace variable
```

Program :

```
#include <iostream>  
using namespace std;  
// Global variable  
int b = 200;  
// Namespace declarations  
namespace N1  
{  
    int a = 500;  
}  
namespace N2  
{  
    int a = 1000;  
}  
// Function to demonstrate local, global, and namespace scope  
void fun()  
{  
    int a = 100; // Local variable inside fun  
    cout << "Fun A : " << a << " " << ::b << endl; // local a, global b  
}  
  
int main()  
{  
    // Display Roll Number in output  
    cout << "Roll No: 24B11AI138" << endl;  
  
    {  
        int a = 100; // Inner block variable
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
        cout << "Inner A : " << a << endl;
    }

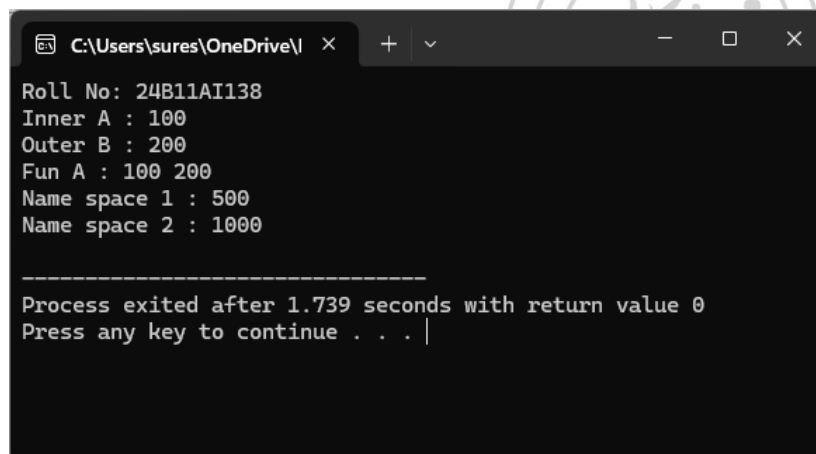
    cout << "Outer B : " << b << endl;

    fun(); // Call to function fun()

    // Using fully qualified names
    cout << "Name space 1 : " << N1::a << endl;
    cout << "Name space 2 : " << N2::a << endl;

    return 0;
}
```

Output_:



```
C:\Users\sures\OneDrive\I
Roll No: 24B11AI138
Inner A : 100
Outer B : 200
Fun A : 100 200
Name space 1 : 500
Name space 2 : 1000

-----
Process exited after 1.739 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO1

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No :2.2

C++ program that illustrates the use of inline functions .

Aim : To write C++ program that illustrates the use of inline functions .

Description : An **inline function** is a function where the compiler replaces the function call with the actual code of the function (to reduce function call overhead).

Syntax :

```
inline return_type function_name(parameter_list) {  
    // function body  
  
    return value;  
}
```

Program :

```
#include <iostream>  
  
using namespace std;  
  
inline int square(int x)  
{  
    return x * x;  
}  
  
int main()  
{  
    cout << "Roll No: 24B11AI138" << endl;  
  
    int num = 4;  
  
    cout << "Square of " << num << " is " << square(num) << endl;  
  
    return 0;  
}
```

Output_:

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
C:\Users\sures\OneDrive\ \times + \square \times
Roll No: 24B11AI138
Square of 4 is 16

-----
Process exited after 3.276 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO1

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1



A D I T Y A
UNIVERSITY

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No :3.1

C++ program that models a Bank Account using a class. The class include data member account number, name, balance and member functions deposit, withdraw, display balance.

Aim: To C++ program that models a Bank Account using a class. The class include data member account number, name, balance and member functions deposit, withdraw, display balance.

Description :

a)Classes :

A *class* is a user – defined data type that groups data members and function into a single unit to represent real-world entities.

Syntax :

```
Class className{  
  
    //data members and member functions  
  
};
```

b)Objects :

An *object* is an instance of a class that holds actual data and access to the class's functions

Syntax :

```
ClassName object;
```

c)Data Members && Member Functions :

Data Members store object attributes ; member functions define behaviours that operate on these attributes.

Syntax :

```
class Bank {  
  
private:  
  
    int accNumber;           // data member  
  
public:  
  
    void deposit(double);    // member function
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

};

d)Encapsulation :

Encapsulation hides data by declaring members private and exposes through public functions , ensuring data integrity.

Syntax :

```
Class Bank{  
  
Private:  
  
    double balance;  
  
Public :  
  
    double getBalance();  
  
};
```

Program :

```
#include<iostream>  
#include<string>  
using namespace std;  
class Bank  
{  
    private:  
        int accNumber;  
        string name;  
        double balance;  
  
    public:  
        Bank(int accno,string accname,double bal)  
        {  
            accNumber=accno;  
            name=accname;  
            balance=bal;  
        }  
};
```



ADITYA
UNIVERSITY

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
void withdraw(double amount)
{
    if(amount < 0)
    {
        cout<<"Withdrrow Not Allowed";
    }
    else if(balance < amount)
    {
        cout<<"Insufficient Balance";
    }
    else
    {
        balance -=amount;
        cout<<"Amount withdraw:"<<balance;
    }
}

void deposit(double amount)
{
    if(amount > 0)
    {
        balance += amount;
        cout<< "Amount is Deposited:"<<balance<<endl;
    }
    else
    {
        cout<<"Not allowed to deposite"<<balance<<endl;
    }
}

void displayBalance()
{
    cout<<"Acc num:"<<accNumber<<endl;
    cout<<"Acc Name:"<<name<<endl;
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

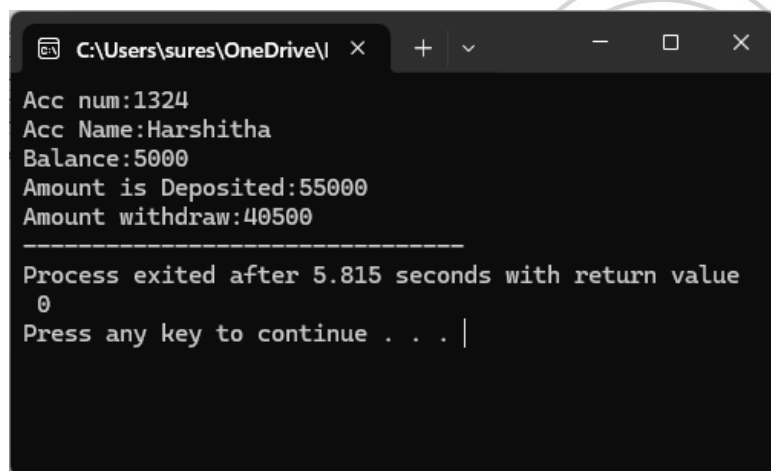
```
        cout<<"Balance:"<<balance<<endl;

    }

};

int main()
{
    Bank bank(1324,"Harshitha",5000.00);
    bank.displayBalance();
    bank.deposit(50000.00);
    bank.withdraw(14500.00);
    return 0;
}
```

Output:



```
Acc num:1324
Acc Name:Harshitha
Balance:5000
Amount is Deposited:55000
Amount withdraw:40500
-----
Process exited after 5.815 seconds with return value
0
Press any key to continue . . . |
```

CO Mapped: CO2

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No:3.2

Program that illustrates the difference between the public and private access specifiers

Aim : To write a program that illustrates the difference between the public and private access specifiers

Description : Access specifiers control visibility of class members.

- private: Accessible only inside the class.
- public: Accessible from outside the class.

They support **data hiding** and **encapsulation**

Syntax:

```
class ClassName {  
private:  
    int a; // private  
public:  
    int b; // public  
};
```

Program :

```
#include <iostream>  
#include <string>  
class Student {  
private:  
    std::string name;  
    int age;  
  
public:  
    void setDetails(const std::string& studentName, int studentAge)  
    {  
        name = studentName;  
        age = studentAge;  
    }  
    void displayDetails() const  
    {  
        std::cout << "Roll No: 24B11AI138" << std::endl;
```



ADITYA
UNIVERSITY

Date:

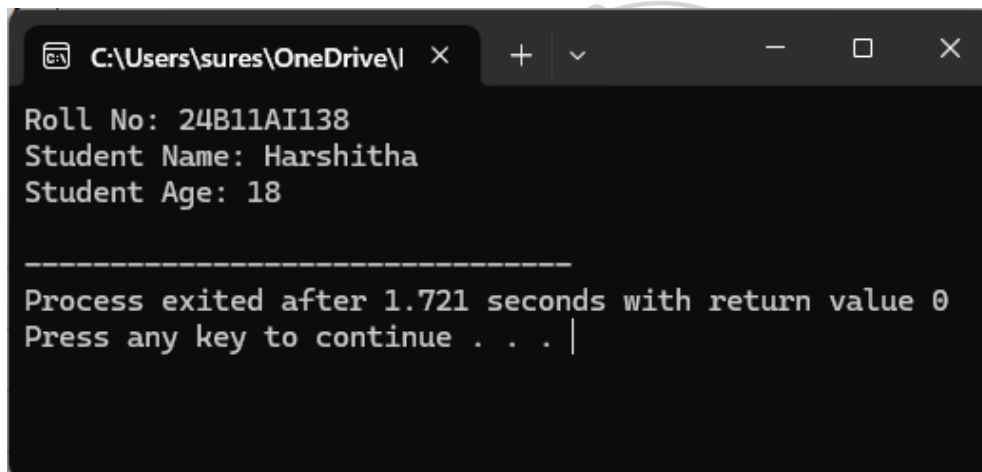
Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
        std::cout << "Student Name: " << name << std::endl;
        std::cout << "Student Age: " << age << std::endl;
    }
};

int main()
{
    Student student;
    student.setDetails("Harshitha", 18);
    student.displayDetails();
    return 0;
}
```

Output_:



```
C:\Users\sures\OneDrive\l
Roll No: 24B11AI138
Student Name: Harshitha
Student Age: 18
-----
Process exited after 1.721 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO2

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No:3.3

Develop a C++ program that uses the this pointer to refer to the current object

Aim : To Develop a C++ program that uses the this pointer to refer to the current object

Description :

This is an implicit pointer that refers to the **current object**. Used to resolve naming conflicts between class members and parameters. Also helps in returning the object itself for **chaining**.

Syntax :

```
class Sample {  
    int x;  
public:  
    void setX(int x) {  
        this->x = x;  
    }  
};
```



A D I T Y A
U N I V E R S I T Y

Program :

```
#include <iostream>  
using namespace std;  
class Student  
{  
private:  
    int rollNo;  
    string name;  
  
public:  
  
    void setData(int rollNo, string name)  
    {  
        this->rollNo = rollNo;
```


Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
        this->name = name;
    }

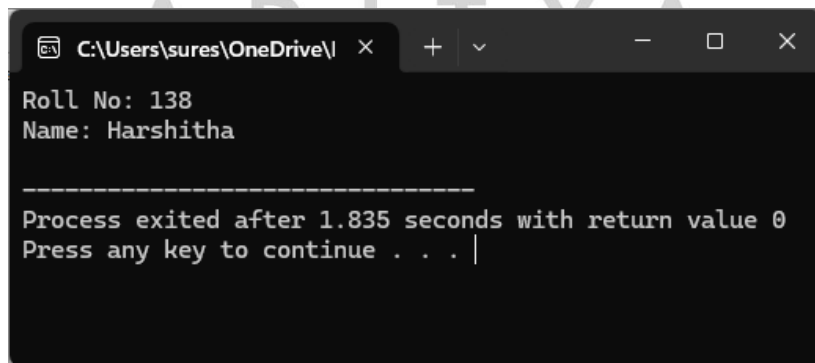
    void display() {
        cout << "Roll No: " << rollNo << endl;
        cout << "Name: " << name << endl;
    }
};

int main()
{
    Student s1;

    s1.setData(200, "Harshitha");
    s1.display();

    return 0;
}
```

Output_:



```
C:\Users\sures\OneDrive\
Roll No: 138
Name: Harshitha
-----
Process exited after 1.835 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO2

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No:4.1

Create a C++ program that demonstrates function overloading by defining multiple functions with the same name but different parameter types or counts

Aim: To create a C++ program that demonstrates function overloading by defining multiple functions with the same name but different parameter types or counts

Description :

Function overloading allows multiple functions with the same name but different parameter types or counts. It improves code readability and reusability by using the same name for similar operations.

Program:

```
#include<iostream>
using namespace std;
class DispDemo
{
    public:
    void display()
    {
        cout<<"Nothing to say"<<endl;
    }
    void display(string msg)
    {
        cout<<"I have something to say : "<<msg<<endl;
    }
    void display(int age)
    {
        cout<<"Age: " <<age<<endl;
    }
    void display(string msg,int age)
    {
        cout<<"Hi " <<msg<<endl;
        cout<<"Age : " <<age<<endl;
    }
    int display(int age, int grace)
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

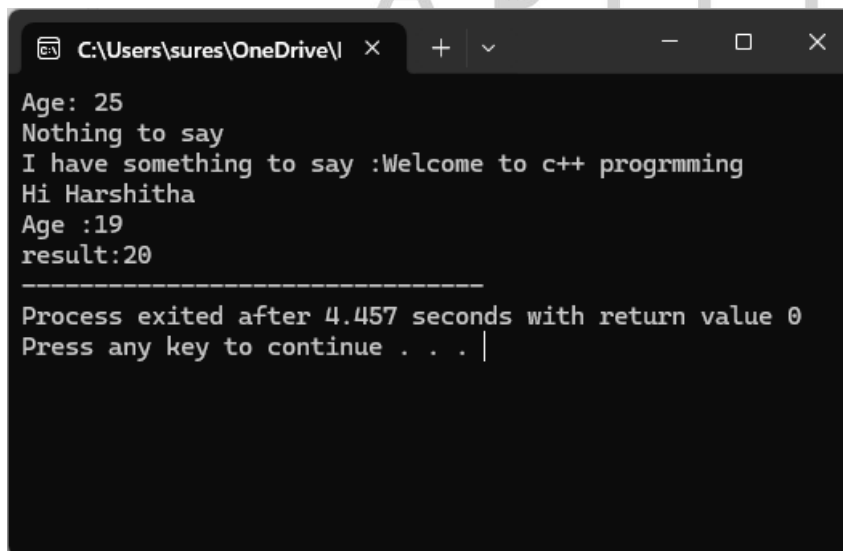
```
{  
    return (age+grace);  
}  
};  
int main()  
{  
    DispDemo dd;  
    dd.display(25);  
    dd.display();  
    dd.display("Welcome to c++ progrmming");  
    dd.display("Harshitha",19);  
    cout<<"result:"<<dd.display(19,1);  
    return 0;  
}
```

Output:

CO Mapped: CO2

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1



```
C:\Users\sures\OneDrive\l x + v - □ x  
Age: 25  
Nothing to say  
I have something to say :Welcome to c++ progrmming  
Hi Harshitha  
Age :19  
result:20  
-----  
Process exited after 4.457 seconds with return value 0  
Press any key to continue . . . |
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No :4.2

Develop a C++ program that illustrates the use of default arguments in functions

Aim: To Develop a C++ program that illustrates the use of default arguments in functions.

Description :

Default arguments let you assign default values to parameters in function declarations. If no value is provided during the call, the default is used. Defaults must be assigned from right to left.

Program :

```
#include<iostream>
using namespace std;
class DispDemo
{
public:
    void display()
    {
        cout << "Nothing to display" << endl;
    }
    void display(string msg)
    {
        cout << "I am having something to display:" << msg << endl;
    }
    void display(int age)
    {
        cout << "Age:" <<age << endl;
    }
    void display(string msg,int age)
    {
        cout << "Hello" << msg <<"Age:" <<age << endl;
    }
    int display(int age, int grace)
    {
```

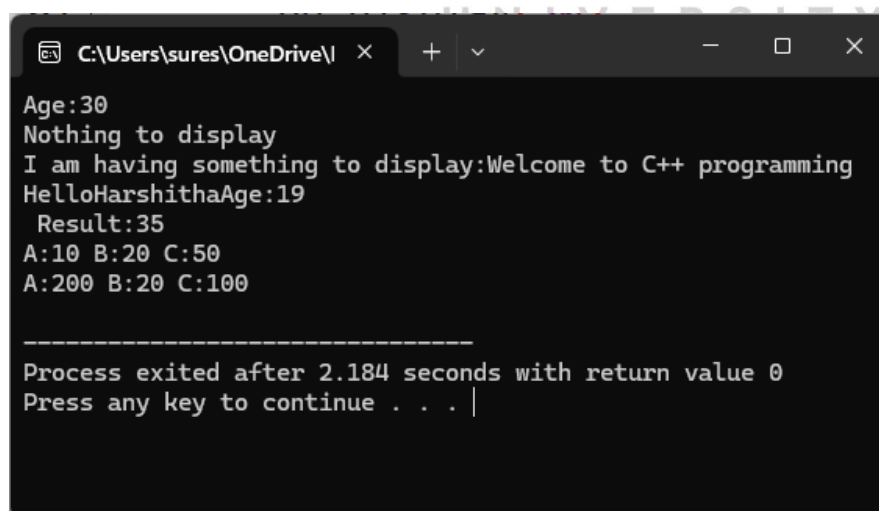
Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
        return (age+grace);
    }
    void show(int c,int a = 10,int b = 20)
    {
        cout << "A:" << a <<" B:" <<b << " C:" << c<< endl;
    }
};
int main()
{
    DispDemo dd;
    dd.display(30);
    dd.display();
    dd.display("Welcome to C++ programming");
    dd.display("Harshitha",19);
    cout << " Result:" << dd.display(30, 5) << endl;
    //dd.show();
    dd.show(50);
    dd.show(100, 200);
    return 0;
}
```

Output:



```
C:\Users\sures\OneDrive\
Age:30
Nothing to display
I am having something to display:Welcome to C++ programming
HelloHarshithaAge:19
Result:35
A:10 B:20 C:50
A:200 B:20 C:100

-----
Process exited after 2.184 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO2

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No:4.3

Create a C++ program that uses a friend function to access the private data of a class

Aim: To Create a C++ program that uses a friend function to access the private data of a class.

Description :

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

Program :

```
#include<iostream>
using namespace std;
class B;
class A
{
    private:
        int a;
        friend int sum(A,B);
        /*A(int a)
        {
            this.a = a;
        }*/
    public:
        A() : a(10) { }
};
class B
{
    private:
        int b;
        friend int sum(A,B);
        /*A(int a)
        {
            this.a = a;
        }*/
```



A D I T Y A
UNIVERSITY

Date:

Roll No.:

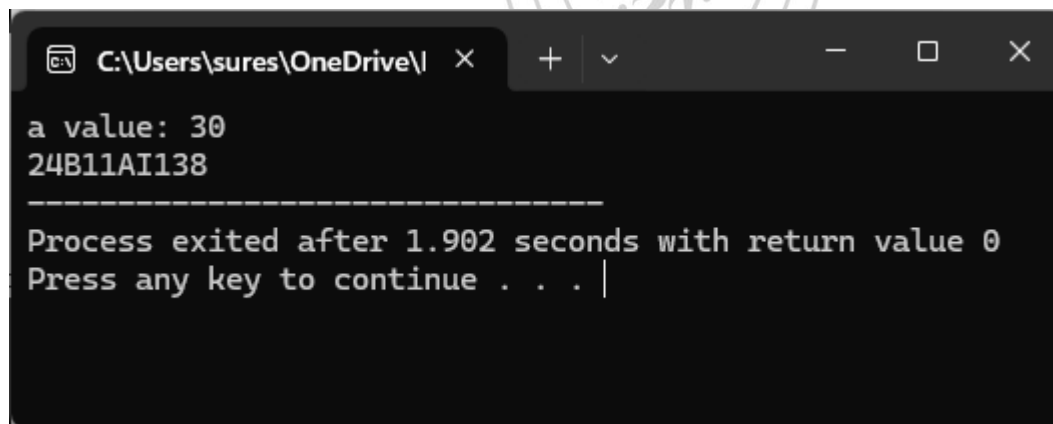
2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
public:
    B() : b(20) { }
};

int sum(A aobj, B bobj)
{
    return(aobj.a + bobj.b);
}

int main()
{
    A aobj;
    B bobj;
    cout << "a value: " << sum(aobj, bobj) << endl;
    cout << "24B11AI138";
    return 0;
}
```

Output:



```
C:\Users\sures\OneDrive\ x + v - □ ×

a value: 30
24B11AI138
-----
Process exited after 1.902 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO2

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No : 5.1

Create a C++ program that demonstrates the use of constructors and destructors in a class.

Aim : To create a C++ program that demonstrates the use of constructors and destructors in a class.

Description :

Constructors and destructors in C++ are special member functions that manage the lifecycle of objects. A constructor is automatically called when an object is created to initialize its data, while a destructor is invoked when the object is destroyed to release resources and perform cleanup

Program :

```
#include <iostream>
using namespace std;
class Student {
private:
    string name;
    int age;
public:
    Student(string n, int a) {
        name = n;
        age = a;
        cout << "Constructor called for " << name << endl;
    }
    ~Student() {
        cout << "Destructor called for " << name << endl;
    }
    void display() {
        cout << "Name: " << name << ", Age: " << age << endl;
    }
};

int main() {
    string n;
    int a;
    cout << "Enter student name: ";
    getline(cin, n);
```

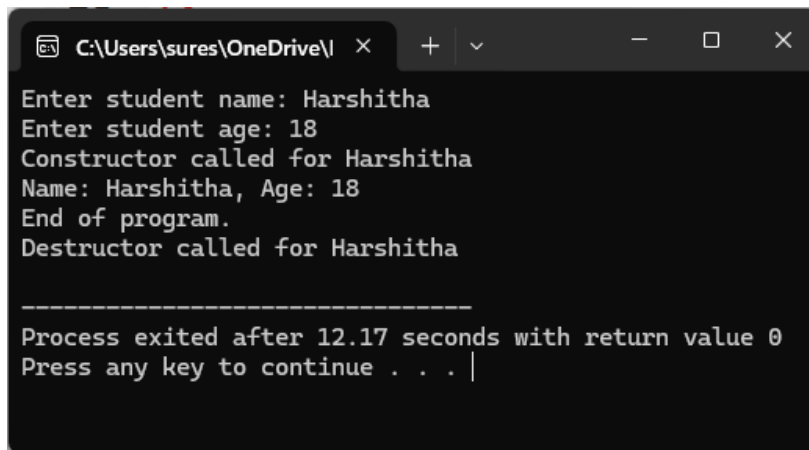

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
cout << "Enter student age: ";  
cin >> a;  
Student s1(n, a);  
s1.display();  
cout << "End of program.\n";  
return 0;  
}
```

Output :



The screenshot shows a Windows command prompt window with the title bar 'C:\Users\sures\OneDrive\I'. The output of the program is as follows:

```
Enter student name: Harshitha  
Enter student age: 18  
Constructor called for Harshitha  
Name: Harshitha, Age: 18  
End of program.  
Destructor called for Harshitha  
  
-----  
Process exited after 12.17 seconds with return value 0  
Press any key to continue . . . |
```

CO Mapped: CO2

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

ADITYA
UNIVERSITY

Program No : 5.2

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Develop a C++ program that illustrates constructor overloading.

Aim : To develop a C++ program that illustrates constructor overloading.

Description :

Constructor overloading in C++ allows a class to have multiple constructors with different parameter lists. This enables objects to be initialized in various ways, applying the same principle as function overloading but specifically for constructors.

Program :

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


class Rectangle {
private:

int length;
    int width;

public:
    Rectangle() {
        length = 0;
        width = 0;
        cout << "Default constructor called!" << endl;
    }

    Rectangle(int side) {
        length = side;
        width = side;
        cout << "Square constructor called!" << endl;
    }

    Rectangle(int l, int w) {
        length = l;
        width = w;
    }
}
```



Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
        cout << "Rectangle constructor called!" << endl;
    }

    void display() {
        cout << "Length: " << length << ", Width: " << width << endl;
    }
};

int main() {
    int choice;
    cout << "Choose how to create the rectangle:\n";
    cout << "1. Default (0x0)\n";
    cout << "2. Square (one side)\n";
    cout << "3. Rectangle (length & width)\n";
    cout << "Enter choice: ";
    cin >> choice;

    if (choice == 1) {
        Rectangle r1;
        r1.display();
    }
    else if (choice == 2){
        int side;
        cout << "Enter side length: ";
        cin >> side;
        Rectangle r2(side);
        r2.display();
    }
    else if (choice == 3) {
        int l, w;
        cout << "Enter length: ";
        cin >> l;
        cout << "Enter width: ";
        cin >> w;
```

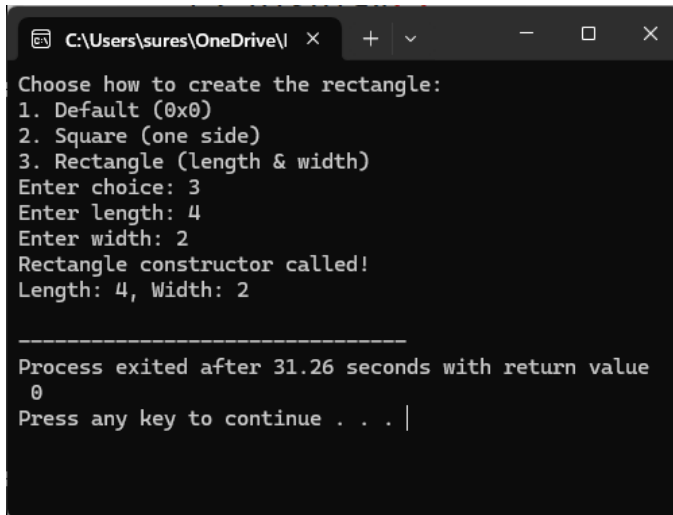
Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
        Rectangle r3(l, w);  
        r3.display();  
    }  
    else {  
        cout << "Invalid choice!\n";  
    }  
  
    return 0;  
}
```

Output:



```
C:\Users\sures\OneDrive\l x + - □ x  
Choose how to create the rectangle:  
1. Default (0x0)  
2. Square (one side)  
3. Rectangle (length & width)  
Enter choice: 3  
Enter length: 4  
Enter width: 2  
Rectangle constructor called!  
Length: 4, Width: 2  
  
-----  
Process exited after 31.26 seconds with return value  
0  
Press any key to continue . . . |
```

CO Mapped: CO2

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

Program No : 5.3

Write a C++ program that illustrates the use of a copy constructor

Aim : Write a C++ program that illustrates the use of a copy constructor .

Description :

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

A copy constructor in C++ is a special constructor used to create a new object as a copy of an existing one. It is invoked when objects are passed by value, returned from functions, or explicitly initialized with another object of the same class.

Program :

```
#include <iostream>
using namespace std;
class Rectangle {
private:

int length;
    int width;
public:
    Rectangle(int l, int w) {
        length = l;
        width = w;
    }
    Rectangle(const Rectangle &r) {
        length = r.length;
        width = r.width;
        cout << "Copy constructor called!" << endl;
    }
    void setLength(int l){
        length = l;
    }
    void display() {
        cout << "Length: " << length << ", Width: " << width << endl;
    }
};

int main() {
    int l, w;
    cout << "Enter length and width: ";
    cin >> l >> w;

    Rectangle rect1(l, w);
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
cout << "\nOriginal Rectangle (rect1):" << endl;

rect1.display();

Rectangle rect2 = rect1;
cout << "\nCopied Rectangle (rect2):" << endl;
rect2.display();

rect1.setLength(1 + 10);
cout << "\nAfter changing rect1's length:" << endl;
cout << "rect1: ";
rect1.display();
cout << "rect2: ";
rect2.display();

return 0;
}
```



A D I T Y A
U N I V E R S I T Y

Output:

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
C:\Users\sures\OneDrive\l  ×  +  -  □  ×
Enter length and width: 2 2
Original Rectangle (rect1):
Length: 2, Width: 2
Copy constructor called!

Copied Rectangle (rect2):
Length: 2, Width: 2

After changing rect1's length:
rect1: Length: 12, Width: 2
rect2: Length: 2, Width: 2

-----
Process exited after 36.19 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO2

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1



A D I T Y A
UNIVERSITY

Program No : 6.1

Develop a C++ program that demonstrates how to overload both unary and binary operators using member functions.

Aim: Develop a C++ program that demonstrates how to overload both unary and binary operators using member functions.

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Description :

Operator overloading using member functions allows us to redefine unary and binary operators inside a class so that they work with user-defined objects, ensuring proper return types and supporting chaining of operations.

Program:

```
#include<iostream>
using namespace std;
class complex
{
    private:
        int real;
        int imaj;
    public:
        complex(int r,int i) : real(r) , imaj(i) { }
        complex operator-()
        {
            return complex(-real,-imaj);
        }
        complex operator+(complex& obj)
        {
            return complex(real+obj.real,imaj+obj.imaj);
        }
        string display()
        {
            return std::to_string(real)+"+"+std::to_string(imaj)+"i\n";
        }
};

int main()
{
    cout<<"24B11AI138"<<endl;
    complex c1(10,20);
    cout<<"complex number c1:"<<c1.display();
    complex c2(30,50);
    cout<<"complex number c2:"<<c2.display();
```

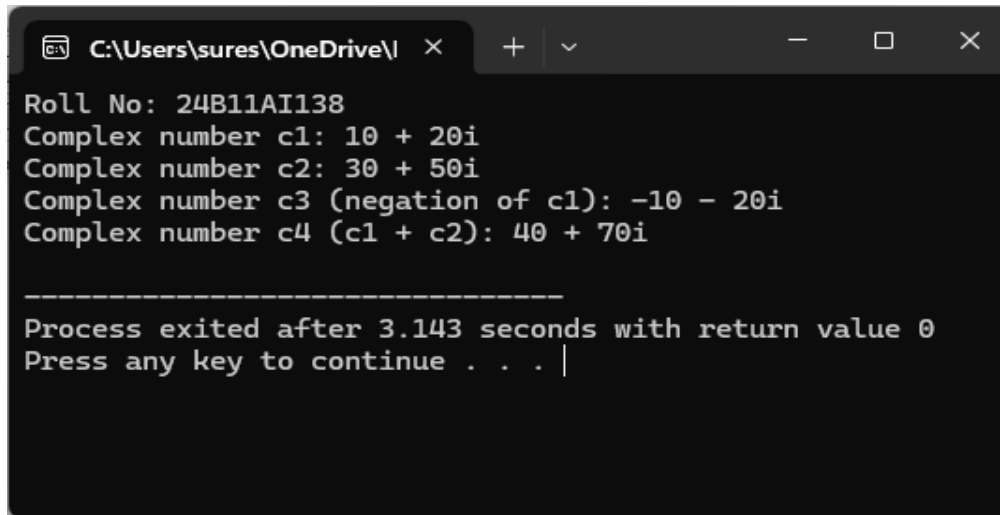

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
complex c3 = -c1;
cout<<"complex number c3:"<<c3.display();
complex c4 = c1+c2;
cout<<"complex number c4:"<<c4.display();
return 0;
}
```

Output:



```
C:\Users\sures\OneDrive\
Roll No: 24B11AI138
Complex number c1: 10 + 20i
Complex number c2: 30 + 50i
Complex number c3 (negation of c1): -10 - 20i
Complex number c4 (c1 + c2): 40 + 70i

-----
Process exited after 3.143 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO3

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

A D I T Y A
UNIVERSITY

Program No:6.2

Create a C++ program to demonstrate operator overloading for unary and binary operators using friend functions

Aim: Create a C++ program to demonstrate operator overloading for unary and binary operators using friend functions

Description :

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Operator overloading using friend functions lets us define operators outside the class while still accessing private members, useful for cases where the left operand is not an object of the class.

Program:

```
#include<iostream>
using namespace std;
class complex
{
    private:
        int real;
        int imaj;
    public:
        complex(int r,int i) : real(r) , imaj(i) { }
        friend complex operator-(complex& obj);
        friend complex operator+(complex& obj1,complex& obj2);
        string display()
        {
            return std::to_string(real)+"+"+std::to_string(imaj)+"i\n";
        }
};

complex operator-(complex& obj)
{
    return complex(-obj.real,-obj.imaj);
}

complex operator+(complex& obj1,complex& obj2)
{
    return complex(obj1.real+obj2.real,obj1.imaj+obj2.imaj);
}

int main()
{
    cout<<"24B11AI138"<<endl;
    complex c1(10,20);
    cout<<"complex number c1:"<<c1.display();
    complex c2(30,50);
    cout<<"complex number c2:"<<c2.display();
```

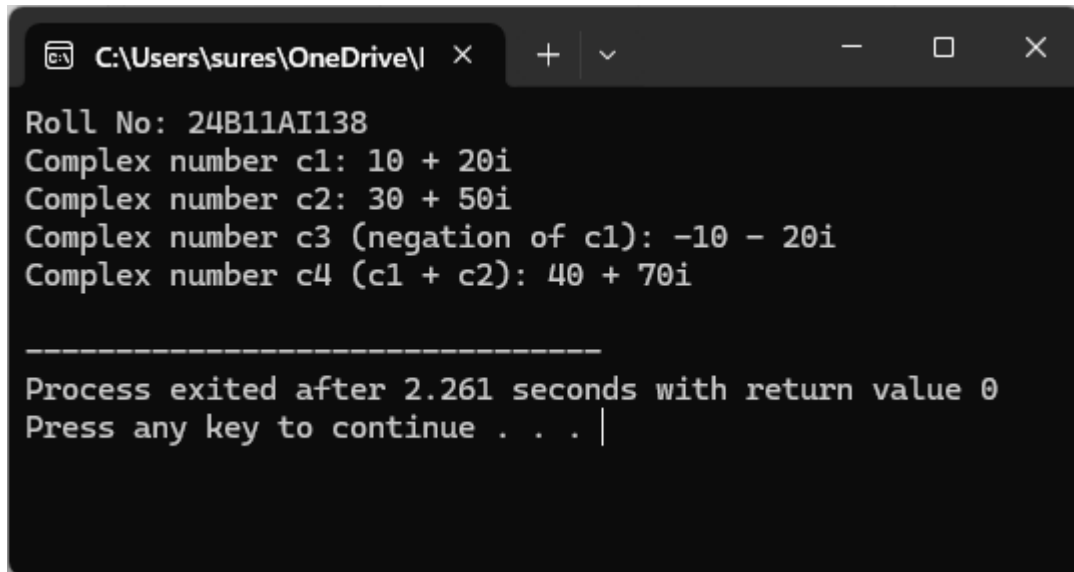
Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
complex c3 = -c1;
cout<<"complex number c3:"<<c3.display();
complex c4 = c1+c2;
cout<<"complex number c4:"<<c4.display();
return 0;
}
```

Output:



```
C:\Users\sures\OneDrive\ x + v - □ ×
Roll No: 24B11AI138
Complex number c1: 10 + 20i
Complex number c2: 30 + 50i
Complex number c3 (negation of c1): -10 - 20i
Complex number c4 (c1 + c2): 40 + 70i
-----
Process exited after 2.261 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO3

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

A D I T Y A
U N I V E R S I T Y

Program No : 7.1

Develop C++ programs to demonstrate different forms of inheritance

Aim : To develop C++ programs to demonstrate different forms of inheritance

Description :

1. Single Inheritance :

In this type, a child class derives properties and methods from only one parent class.

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

It is the simplest form of inheritance and avoids ambiguity.

2. Multiple Inheritance :

A child class inherits from two or more parent classes.

This allows the child class to combine features of multiple classes.

However, it can sometimes lead to conflicts (for example, if two parent classes have the same method name).

3. Multilevel Inheritance :

Here, a class inherits from another class, which itself is derived from another class.

It forms a chain-like structure (grandparent → parent → child).

This shows the concept of transitive inheritance, where the child indirectly inherits features from the grandparent class.

4. Hierarchical Inheritance :

In this form, multiple child classes inherit from a single parent class.

Each child has access to the parent's features, but they can also define their own.

This is useful when different classes share common characteristics.

5. Hybrid Inheritance :

A combination of two or more types of inheritance.

For example, a class may use both multiple and multilevel inheritance at the same time.

It is powerful but can create ambiguity, which is resolved differently in different programming languages (e.g., Method Resolution Order in Python).

Program :

```
//1. Single Inheritance:
#include <iostream>
using namespace std;
class Parent
{
    public:
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
void display() {
    cout << "This is Parent class" << endl;
}

};

class Child :
public Parent
{
    public:
    void show() {
        cout << "This is Child class" << endl;
    }
};

int main() {
    cout<<"24B11AI138"<<endl;
    Child obj;
    obj.display();
    obj.show();
    return 0;
}

//2 .Multiple Inheritance :
#include <iostream>
using namespace std;
class Father {
public:
void quality() {
cout << "Quality from Father" << endl;
}
};

class Mother {
public:
void skill() {
cout << "Skill from Mother" << endl;
}
};
```



A D I T Y A
U N I V E R S I T Y

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
class Child : public Father, public Mother {
public:
void own() {
    cout << "Own feature of Child" << endl;
}
};

int main() {
    cout<<"24B11AI138"<<endl;
    Child obj;
    obj.quality();
    obj.skill();
    obj.own();

return 0;
}
```

//3. Multilevel Inheritance :

```
#include <iostream>
using namespace std;
class Grandparent {
public:
void heritage() {
    cout << "This is Grandparent heritage" << endl;
}
};

class Parent : public Grandparent {
public:
void property() {
    cout << "This is Parent property" << endl;
}
};

class Child : public Parent {
public:
void own() {
    cout << "This is Child's own feature"<<endl;
}
}
```



ADITYA
UNIVERSITY

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
};

int main() {
    cout<<"24B11AI138"<<endl;
    Child obj;
    obj.heritage();
    obj.property();
    obj.own();
    return 0;
}

//4. Hierarchical Inheritance :
#include <iostream>
using namespace std;
class Parent {
public:
void commonFeature() {
    cout << "Common feature from Parent" << endl;
}
};
class Child1 : public Parent {
public:
void feature1() {
    cout << "Feature of Child1" << endl;
}
};
class Child2 : public Parent {
public:
void feature2() {
    cout << "Feature of Child2" << endl;
}
};
int main() {
    cout<<"24B11AI138"<<endl;
    Child1 c1;
    c1.commonFeature();
}
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
c1.feature1();
Child2 c2;
c2.commonFeature();
c2.feature2();
return 0;
}

//5. Hybrid Inheritance:
#include <iostream>
using namespace std;
class A {
public:
void showA() {
cout << "Class A feature" << endl;
}
};
class B : public A {
public:
void showB() {
cout << "Class B feature" << endl;
}
};
class C : public A {
public:
void showC() {
cout << "Class C feature" << endl;
}
};
class D : public B, public C {
public:
void showD() {
cout << "Class D feature" << endl;
}
};
int main() {
```



A D I T Y A
U N I V E R S I T Y

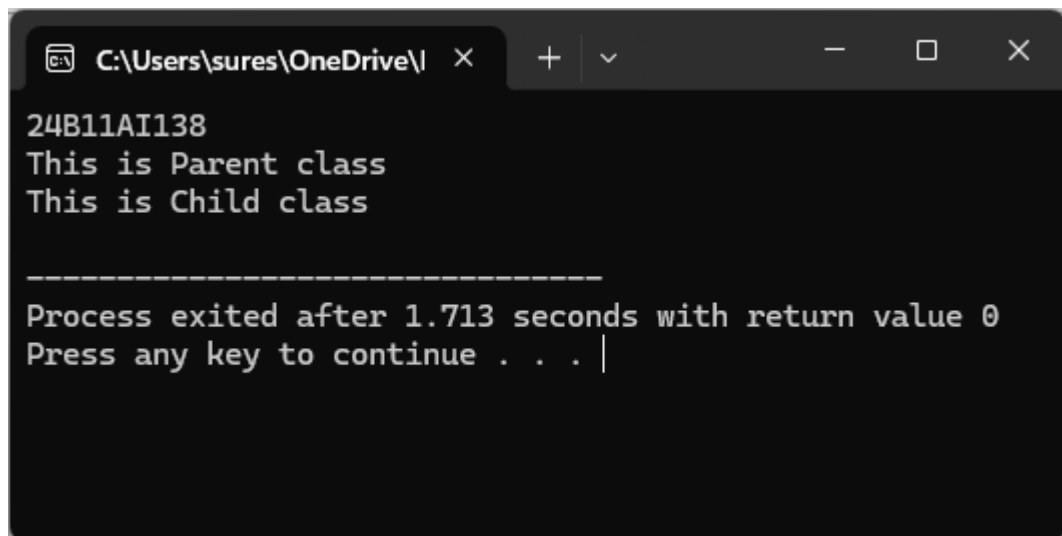
Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

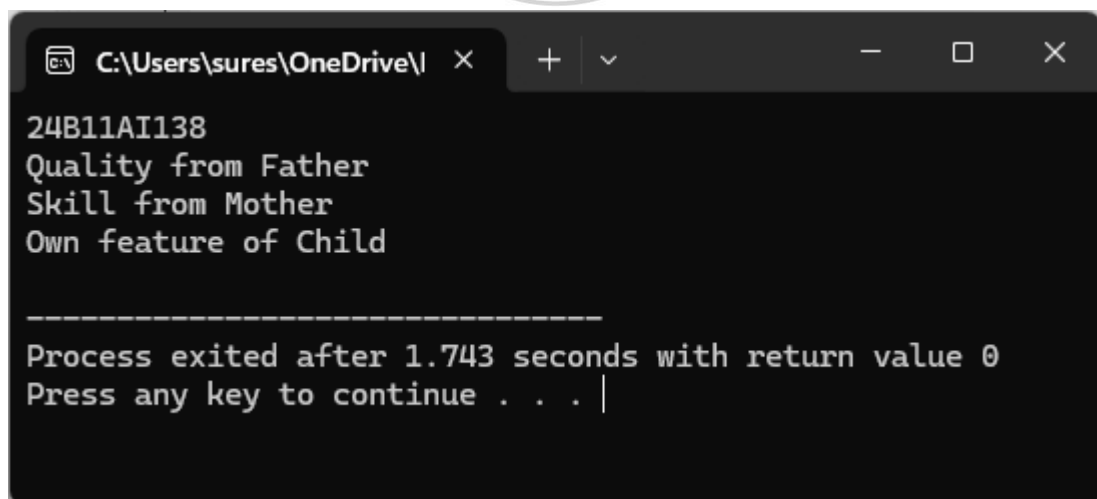
```
cout<<"24B11AI138"<<endl;
D obj;
obj.showB();
obj.showC();
obj.showD();
return 0;
}
```

OUTPUT :



```
C:\Users\sures\OneDrive\ 24B11AI138
This is Parent class
This is Child class

-----
Process exited after 1.713 seconds with return value 0
Press any key to continue . . . |
```



```
C:\Users\sures\OneDrive\ 24B11AI138
Quality from Father
Skill from Mother
Own feature of Child

-----
Process exited after 1.743 seconds with return value 0
Press any key to continue . . . |
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
C:\Users\sures\OneDrive\l x + v - □ ×
24B11AI138
This is Grandparent heritage
This is Parent property
This is Child's own feature

-----
Process exited after 1.589 seconds with return value 0
Press any key to continue . . . |
```

```
C:\Users\sures\OneDrive\l x + v - □ ×
24B11AI138
Common feature from Parent
Feature of Child1
Common feature from Parent
Feature of Child2

-----
Process exited after 1.667 seconds with return value 0
Press any key to continue . . . |
```

```
C:\Users\sures\OneDrive\l x + v - □ ×
Class B feature
Class C feature
Class D feature

-----
Process exited after 1.661 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO3

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO11

PSOs Mapped: PSO1

Program No : 7.2

Develop a C++ program that illustrates the order of execution for constructors and destructors in the context of inheritance.

AIM : To develop a C++ program that illustrates the order of execution for constructors and destructors in the context of inheritance.

Description :

Constructors

- When an object of a derived class is created, constructors are executed in the following order:
 - o First, the base class constructor is called.
 - o Then, constructors of intermediate derived classes (if any).
 - o Finally, the most derived class constructor.
- This ensures that base parts of the object are properly initialized before the derived parts.

Destructors

- When the object goes out of scope or is deleted, destructors are executed in the reverse order:
 - o First, the most derived class destructor.
 - o Then, destructors of intermediate classes.
 - o Finally, the base class destructor.
- This ensures that derived parts are cleaned up before the base class.

Program :

```
#include <iostream>
using namespace std;
class Base {
public:
    Base() {
        cout << "Base Constructor called" << endl;
    }
}
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
~Base() {
    cout << "Base Destructor called" << endl;
}

};

class Derived1 : public Base {
public:
    Derived1() {
        cout << "Derived1 Constructor called" << endl;
    }
    ~Derived1() {
        cout << "Derived1 Destructor called" << endl;
    }
};

class Derived2 : public Derived1 {
public:
    Derived2() {
        cout << "Derived2 Constructor called" << endl;
    }
    ~Derived2() {
        cout << "Derived2 Destructor called" << endl;
    }
};

int main() {
    cout << "24B11AI138" << endl;
    cout << "Creating object of Derived2..." << endl;
    Derived2 obj; // Constructors will run in order
    cout << "Exiting main..." << endl;
    return 0;
}
```

Date:

Roll No.:

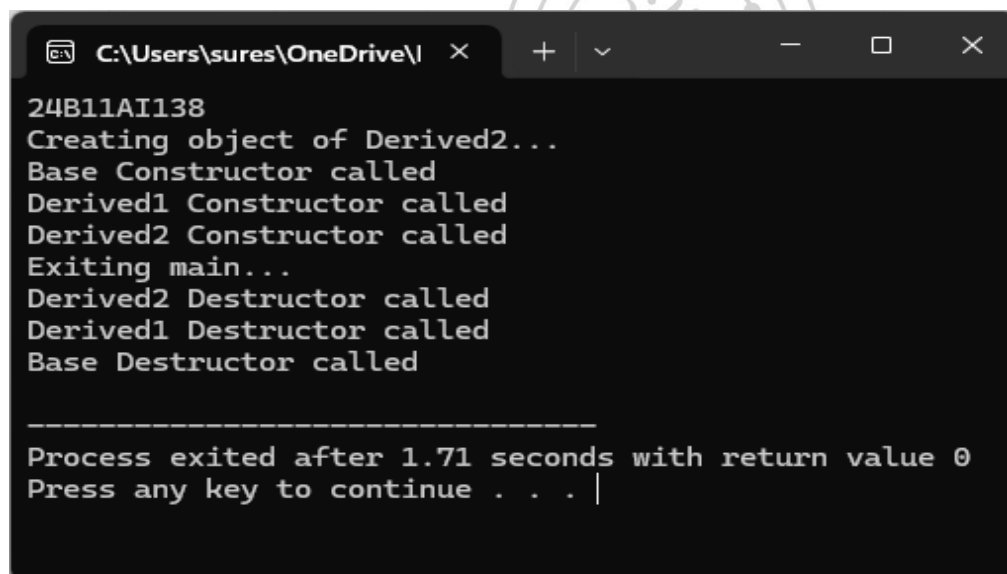
2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

OUTPUT :

CO Mapped: CO3

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO11

PSOs Mapped: PSO1



```
C:\Users\sures\OneDrive\ 24B11AI138
Creating object of Derived2...
Base Constructor called
Derived1 Constructor called
Derived2 Constructor called
Exiting main...
Derived2 Destructor called
Derived1 Destructor called
Base Destructor called

-----
Process exited after 1.71 seconds with return value 0
Press any key to continue . . . |
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

Program No : 8.1

Develop a C++ program that demonstrates how to use pointers to access and manipulate objects of a class.

AIM : To develop a C++ program that demonstrates how to use pointers to access and manipulate objects of a class.

Description :

This program illustrates the use of **pointers to objects** in C++ by creating objects both on the stack and dynamically on the heap. It demonstrates **dereferencing and member access** using the -> operator, modifying object data through member functions, and applying proper **memory management** with new and delete.

Program :

```
#include <iostream>
#include <string>
using namespace std;
class student
{
private:
    string name;
    int age;
public:
    student(string n = "no name", int a = 0)
    {
        name = n;
        age = a;
    }
    void setplayer(string n)
    {
```

Date:

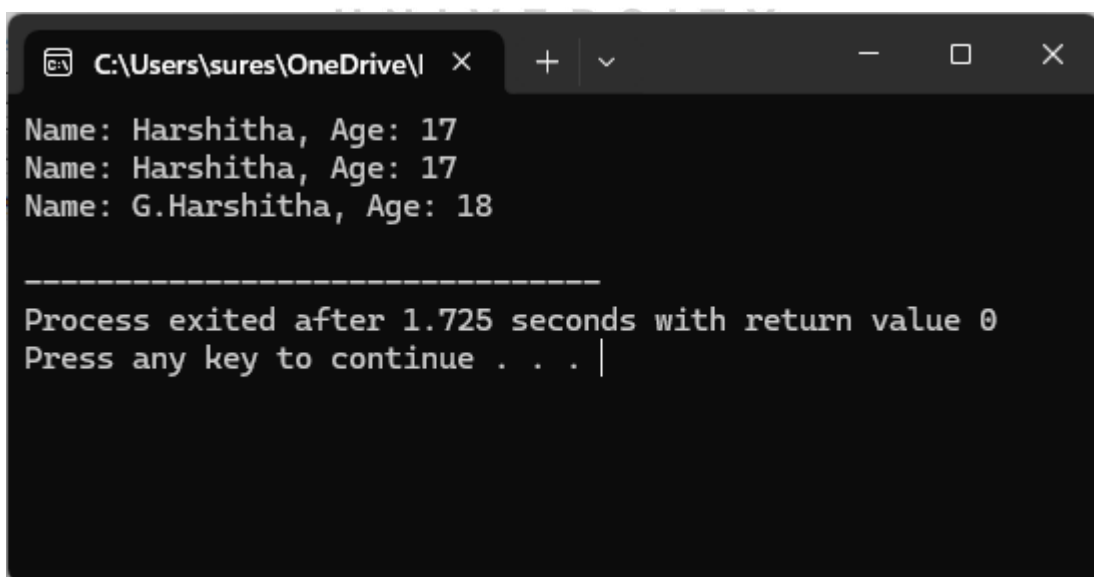
Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
        name = n;
    }
    void dispInfo()
    {
        cout << "Name: " << name << " Age: " << age << endl;
    }
};

int main()
{
    student s("Harshitha", 17);\
    student* sp;
    sp = &s;
    sp->dispInfo();
    sp->setplayer("Harshitha");
    sp->dispInfo();
    student* sp1 = new student("G.Harshitha", 18);
    sp1->dispInfo();
    delete sp1;
    return 0;
}
```

OUTPUT :



```
C:\Users\sures\OneDrive\l x + v - □ X
Name: Harshitha, Age: 17
Name: Harshitha, Age: 17
Name: G.Harshitha, Age: 18

-----
Process exited after 1.725 seconds with return value 0
Press any key to continue . . . |
```

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

CO Mapped: CO3

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11

PSOs Mapped: PSO1

Program No : 8.2

Develop a C++ program to demonstrate the concept of virtual base classes in the context of multiple inheritance, which resolves ambiguity in the inheritance hierarchy

AIM : To develop a C++ program to demonstrate the concept of virtual base classes in the context of multiple inheritance, which resolves ambiguity in the inheritance hierarchy

Description :

This program demonstrates the diamond problem in multiple inheritance and how it is resolved using virtual base classes in C++. It also illustrates the order of constructor and destructor calls, showing how virtual inheritance ensures that the common base class is initialized only once, avoiding ambiguity in the inheritance hierarchy.

Program :

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

class person
{
protected:
    string name;

public:
    void setName(string n)
    {
```


Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
        name = n;
    }

    void displayPerson()
    {
        cout << "Name: " << name << endl;
    }
};

class student : virtual public person
{
protected:
    string course;

public:
    void setcourse(string c)
    {
        course = c;
    }

    void displayPerson()
    {
        cout << "Course: " << course << endl;
    }
};

class Employee : virtual public person
{
protected:
    string company;

public:
    void setcompany(string c)
```



A D I T Y A
U N I V E R S I T Y

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

```
{
    company = c;
}

void displayPerson()
{
    cout << "Company: " << company << endl;
}

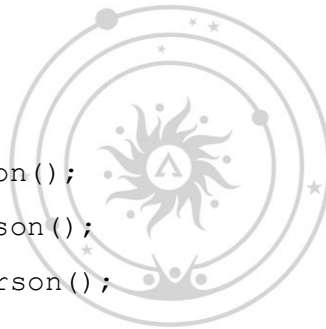
};

class Workingstudent : public student, public Employee
{
public:
    void displayInfo()
    {
        person::displayPerson();
        student::displayPerson();
        Employee::displayPerson();
    }
};

int main()
{
    cout<<"24B11AI138"<<endl;

    Workingstudent ws;
    ws.setName("Grace");
    ws.setcourse("C++");
    ws.setcompany("ADITYA");

    ws.displayInfo();
    return 0;
}
```



A D I T Y A
U N I V E R S I T Y

Date:

Roll No.:

2	4	B	1	1	A	I	1	3	8
---	---	---	---	---	---	---	---	---	---

OUTPUT :

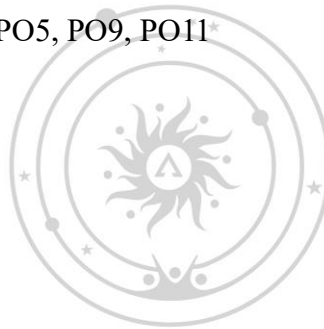
```
C:\Users\sures\OneDrive\l x + v - □ ×
Roll No: 24B11AI138
Name: Hashitha
Course: C++
Company: ADITYA

-----
Process exited after 2.22 seconds with return value 0
Press any key to continue . . . |
```

CO Mapped: CO3

POs Mapped: PO1, PO2, PO3, PO4, PO5, PO9, PO11*

PSOs Mapped: PSO1



A D I T Y A
U N I V E R S I T Y