

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

WEEK 1

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 logic
    }
    else
    {
        return function_name(modified_parameters);
    }
}
```



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

Program:

```
#include <iostream>
#include <cstdlib>
using namespace std;
int fact(int n);
int main()
{
    system("color F0");
    int n;
    cout << "Roll No: 24B11AI439" << endl;
    cout << "Enter n value: ";
    cin >> n;
    int result = fact(n);
    if (result == -1)
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
    cout << "Factorial is not defined" << endl;
else
    cout << "Factorial: " << result << endl;
return 0;
}
```

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

Output:

Roll No: 24B11AI439

Enter n value: 6

Factorial: 720

==== Code Execution Successful ====



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

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>
```

```
#include <cstdlib>
```

```
using namespace std;
```

```
{
```

```
    int temp = first;
```

```
    first = second;
```

```
    second = temp;
```

```
}
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
int main()
{
    system("color F0");
    cout << "Roll No: 24B11AI439" << 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);
    cout << "\nAfter Swapping:\n";
    cout << "a = " << a << endl;
    cout << "b = " << b << endl;
    return 0;
}
```

Output_:

Roll No: 24B11AI439

Enter two numbers (a and b): 10 20

Before Swapping:

a = 10

b = 20

After Swapping:

a = 20

b = 10

=== Code Execution Successful ===



ADITYA
UNIVERSITY

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

WEEK 2

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;

void fun() {
    int localVar = 5;
}
```



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;

void fun() {
    int x = 10;
    cout << ::x;
}
```

c) Namespaces :

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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

Syntax:

```
namespace MyNamespace {  
    int value = 80;  
}  
  
cout << MyNamespace::value;
```

Program :

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

```
{  
    int a = 90;  
}  
  
namespace N2  
{  
    int a = 90;  
}
```

```
{  
    int a = 90;
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
cout << "Fun A : " << a << " " << ::b << endl; // local a, global b  
}
```

```
int main()
```

```
{
```

```
    system("color F0");
```

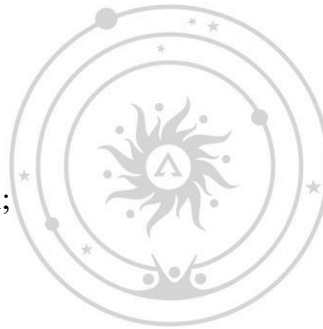
```
    cout << "Roll No: 24B11AI439" << endl;
```

```
{
```

```
    int a = 90;
```

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

```
}
```



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

ADITYA
UNIVERSITY

```
    fun();
```

```
    cout << "Name space 1 : " << N1::a << endl;
```

```
    cout << "Name space 2 : " << N2::a << endl;
```

```
    return 0;
```

```
}
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

Output_:

Roll No: 24B11AI439

Inner A : 90

Outer B : 45 Fun

A : 90 45

Name space 1 : 90

Name space 2 : 90

=== Code Execution Successful ===



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

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) {
```

```
}
```

Program :

```
#include <iostream>
```

```
#include <cstdlib>
```

```
using namespace std;
```

```
inline int square(int x)
```

```
{
```

```
    return x * x;
```

```
}
```

```
int main()
```

```
{
```

```
    system("color F0");
```

```
    cout << "Roll No: 24B11AI439" << endl;
```

```
    int num = 10;
```

```
    cout << "Square of " << num << " is " << square(num) << endl;
```

```
    return 0;
```

```
}
```



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

Output_:

Roll No: 24B11AI439

Square of 10 is 100

=== Code Execution Successful ===



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

WEEK 3

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{  
  
  
};
```



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;
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

public:

```
void deposit(double);
```

```
};
```

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
```

```
BankAccount { private:
```

```
int accountNumber;
```

```
string name;
```

```
float balance;
```

```
public:
```

```
void createAccount() {
```

```
cout << "Enter Account Number: ";
```

```
cin >> accountNumber; cin.ignore();
```

```
cout << "Enter Account Holder Name: ";
```



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
getline(cin, name);
cout << "Enter Initial Balance: ₹"; cin
>> balance;
cout << "\nAccount Created Successfully!\n";
}
void deposit() {
    float amount;
    cout << "Enter amount to deposit: ₹"; cin
    >> amount;
    if (amount > 0) { balance
        += amount;
        cout << "₹" << amount << " deposited successfully.\n";
    } else {
        cout << "Invalid deposit amount.\n";
    }
}
void withdraw() {
    float amount;
    cout << "Enter amount to withdraw: ₹"; cin
    >> amount;
    if (amount > 0 && amount <= balance) {
        balance -= amount;
        cout << "₹" << amount << " withdrawn successfully.\n";
    } else {
        cout << "Insufficient balance or invalid amount.\n";
    }
}
void displayBalance() const {
    cout << "\n--- Account Details ---\n";
    cout << "Account Number : " << accountNumber << endl;
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
cout << "Account Holder : " << name << endl; cout
<< "Current Balance : ₹" << balance << endl;
}
};
int main() {
    BankAccount myAccount; int
    choice;
    myAccount.createAccount();
    do {
        cout << "\n--- Bank Menu ---\n"; cout
        << "1. Deposit\n";
        cout << "2. Withdraw\n";
        cout << "3. Display Balance\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: "; cin
        >> choice;
        switch (choice) {
            case 1:
                myAccount.deposit();
                break;
            case 2:
                myAccount.withdraw(); break;
            case 3:
                myAccount.displayBalance();
                break;
            case 4:
                cout << "Thank you for using our banking system.\n";
                break;
            default:
```



ADITYA
UNIVERSITY

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
cout << "Invalid choice. Please try again.\n";  
}  
  
} while (choice != 4);  
  
return 0;  
}
```

Output:

Enter Account Number:1234567

Enter Account Holder Name: Vivek

Enter Initial Balance: ₹ 60000

Account Created Successfully!

--- Bank Menu ---

1. Deposit

2. Withdraw

3. Display Balance

4. Exit

Enter your choice: 1

Enter amount to deposit: 1000

₹ 1000 deposited successfully.

--- Bank Menu ---

1. Deposit

2. Withdraw

3. Display Balance

4. Exit

Enter your choice: 3



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

--- Account Details ---

Account Number : 1234567

Account Holder : Vivek

Current Balance : ₹ 61000

--- Bank Menu ---

1. Deposit

2. Withdraw

3. Display Balance

4. Exit

Enter your choice: 2

Enter amount to withdraw: ₹ 1000

₹ 1000 withdrawn successfully.



--- Bank Menu ---

1. Deposit

2. Withdraw

3. Display Balance

4. Exit

Enter your choice: 4

Thank you for using our banking system.

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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

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;
```

```
public:
```

```
    int b;
```

```
};
```

Program :

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
class Student {
```

```
private:
```

```
    string studentName;
```

```
    int rollNumber; float
```

```
    marks;
```

```
public:
```

```
    string college; void
```

```
    setDetails() {
```

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

```
        getline(cin, studentName);
```

```
        cout << "Enter roll number: "; cin
```

```
        >> rollNumber;
```

```
        cout << "Enter marks (out of 100): ";
```



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


Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
        cin >> marks;
    }
    void displayDetails() {
        cout << "\n--- Student Details ---\n";
        cout << "Name      : " << studentName << endl;
        cout << "Roll No   : " << rollNumber << endl;
        cout << "Marks     : " << marks << endl;
        cout << "College   : " << college << endl;
    }
};

int main() {
    Student s1;
    cout << "Enter college name: ";
    getline(cin, s1.college);
    s1.setDetails(); s1.displayDetails();
    s1.college = "Aditya University";
    cout << "\nUpdated College Name: " << s1.college << endl;
    return 0;
}
```



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

Output:

Enter college name: aditya university

Enter student name: V.Vivek

Enter roll number: 439

Enter marks (out of 100): 96

--- Student Details ---

Name : V.Vivek

Roll No : 439

Marks : 96

College : aditya university

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

Updated College Name: Aditya University



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

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>  
#include <string>  
using namespace std;  
class Student {  
private:  
    string name; int  
    rollNumber;  
    float marks;  
public:  
    Student() {  
        name = "";  
        rollNumber = 0;  
        marks = 0.0;  
    }  
};
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
void setDetails(string name, int rollNumber, float marks) {
    this->name = name;
    this->rollNumber = rollNumber;
    this->marks = marks;
}

Student& updateName(string name) {
    this->name = name;
    return *this;
}

Student& updateMarks(float marks) {
    this->marks = marks;
    return *this;
}

void display() {
    cout << "\n--- Student Record ---\n"; cout
    << "Name          : " << name << endl;
    cout << "Roll No.   : " << rollNumber << endl;
    cout << "Marks      : " << marks << "/100\n";
}

};

int main() {
    Student s1;
    string n; int
    r;
    float m;
    cout << "Enter Student Name: ";
    getline(cin, n);
    cout << "Enter Roll Number: ";
    cin >> r;
    cout << "Enter Marks (out of 100): ";
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
cin >> m;
s1.setDetails(n, r, m);
s1.display();
string updatedName;
float updatedMarks;
cout << "\nUpdate Name: ";
getline(cin, updatedName);
cout << "Update Marks: "; cin
>> updatedMarks;
s1.updateName(updatedName).updateMarks(updatedMarks);
s1.display();
return 0;
}
```

Output:

Enter Student Name: Vivek

Enter Roll Number: 439

Enter Marks (out of 100): 98

--- Student Record ---

Name : Vivek

Roll No. : 439

Marks : 98/100

Update Name: Vardhan

Update Marks: 99

--- Student Record ---

Name : Vardhan

Roll No. : 439

Marks : 99/100



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

WEEK 4

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>
#include <string>
using namespace std;

void printDetails(string name, int rollNumber) { cout
    << "\nStudent Details:\n";
    cout << "Name: " << name << "\nRoll Number: " << rollNumber << endl;
}

void printDetails(string name, string subject) {
    cout << "\nTeacher Details:\n";
    cout << "Name: " << name << "\nSubject: " << subject << endl;
}

void printDetails(string courseName, float duration) { cout
    << "\nCourse Details:\n";
    cout << "Course Name: " << courseName << "\nDuration: " << duration << " months" << endl;
}

int main() {
    int choice;
    cout << "Select Option:\n";
    cout << "1. Print Student Details\n";
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
cout << "2. Print Teacher Details\n"; cout
<< "3. Print Course Details\n"; cout <<
"Enter your choice (1-3): "; cin >>
choice;
cin.ignore();
if (choice == 1) {
    string name;
    int roll;
    cout << "Enter Student Name: ";
    getline(cin, name);
    cout << "Enter Roll Number: "; cin
    >> roll;
    printDetails(name, roll);
}
else if (choice == 2) {
    string name, subject;
    cout << "Enter Teacher Name: ";
    getline(cin, name);
    cout << "Enter Subject: ";
    getline(cin, subject);
    printDetails(name, subject);
}
else if (choice == 3) {
    string course; float
    duration;
    cout << "Enter Course Name: ";
    getline(cin, course);
    cout << "Enter Duration (in months): "; cin
    >> duration;
    printDetails(course, duration);
```



ADITYA
UNIVERSITY

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
}  
else {  
    cout << "Invalid Choice!" << endl;  
}  
return 0;  
}
```

Output:

Select Option:

- 1. Print Student Details**
- 2. Print Teacher Details**
- 3. Print Course Details**

Enter your choice (1-3): 1

Enter Student Name: Vivek

Enter Roll Number: 439

Student Details:

Name: Vivek

Roll Number: 439



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

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;

void greet(string name, string greeting = "Hello", int times = 1) { for
    (int i = 0; i < times; ++i) {
        cout << greeting << ", " << name << "!" << endl;
    }
}

int main() {
    string name, greeting; int
    choice, times;
    cout << "Enter your name: "; getline(cin,
    name);
    cout << "\nChoose Greeting Type:\n";
    cout << "1. Just Name (use default greeting and times)\n"; cout <<
    "2. Name and Custom Greeting (use default times)\n"; cout << "3.
    Name, Custom Greeting, and Times\n";
    cout << "Enter your choice (1-3): "; cin
    >> choice;
    cin.ignore();
    if(choice == 1) {
        greet(name);
    }
    else if (choice == 2) {
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
cout << "Enter your custom greeting (e.g., Hi, Welcome, Good Morning): ";
getline(cin, greeting);
greet(name, greeting);
}
else if (choice == 3) {
    cout << "Enter your custom greeting: ";
    getline(cin, greeting);
    cout << "How many times do you want the greeting? "; cin
    >> times;
    greet(name, greeting, times);
}
else {
    cout << "Invalid choice!" << endl;
}

return 0;
}
```



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

Output:

Enter your name: Vivek

Choose Greeting Type:

- 1. Just Name (use default greeting and times)**
- 2. Name and Custom Greeting (use default times)**
- 3. Name, Custom Greeting, and Times**

Enter your choice (1-3): 3

**Enter your custom greeting: jai sri ram! How
many times do you want the greeting? 2 jai sri
ram,Vivek!Jai sri ram,Vivek!**

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

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.

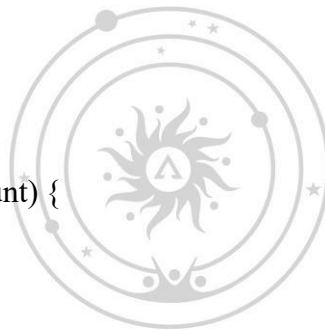
Program :

```
#include <iostream>

#include <string> using
namespace std; class
BankAccount { private:
    string accountHolder;
    float balance;
public:
    BankAccount(string name, float amount) {
        accountHolder = name;
        balance = amount;
    }
    friend void showAccountDetails(BankAccount acc);
};

void showAccountDetails(BankAccount acc) {
    cout << "\n--- Account Details ---\n";
    cout << "Account Holder: " << acc.accountHolder << endl; cout
    << "Balance: ₹" << acc.balance << endl;
}

int main() {
    string name;
    float amount;
    cout << "Enter Account Holder Name: ";
```



ADITYA
UNIVERSITY

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
getline(cin, name);  
cout << "Enter Initial Balance: ₹"; cin  
>> amount;  
BankAccount userAccount(name, amount);  
showAccountDetails(userAccount);  
return 0;  
}
```

Output:

Enter Account Holder Name: Vivek

Enter Initial Balance: 60000

--- Account Details ---

Account Holder: Vivek

Balance: ₹ 60000



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

WEEK-6

Program.6.1

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

Aim

To demonstrate how to overload unary and binary operators in C++ using member functions.

Description

- Unary Operator Overloading: We'll overload the - operator to negate the values of a class.
- Binary Operator Overloading: We'll overload the + operator to add two objects of the class.

Syntax

return_type ClassName::operator op() // Unary

return_type ClassName::operator op(const ClassName&) // Binary

Program.:

```
#include<iostream>
```

```
using namespace std;
```

```
class complex{
```

```
    private:
```

```
        float real,image;
```

```
    public:
```

```
        complex(float r=0,float i=0):real(r),image(i){ }
```

```
    void display()
```

```
{
```

```
    cout<<real<<"+"<<image<<"i"<<endl;
```

```
}
```

```
complex operator-(
```

```
{
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
        return complex(-real,-image);
    }

    complex operator++()
    {
        ++real;
        ++image;

        return *this;
    }

    complex operator+(const complex& obj){
        return complex(real+obj.real,image+obj.image);
    }

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

int main(){
    complex c1(3,4),c2(1,2),c3;

    cout<<"Roll No:24B11AI439"<<endl;
    cout<<"original complex numbers:"<<endl;
    cout<<"c1=";c1.display();

    cout<<"c2=";c2.display();

    c3=-c1;

    cout<<"\nAfter unary - on c1:"<<endl;

    cout<<"c3=";c3.display();

    ++c1;

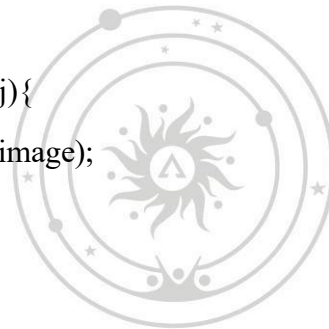
    cout<<"\n After unary ++ on c1:"<<endl;

    cout<<"c1=";c1.display();

    c3=c1+c2;

    cout<<"\nAfter c1+c2:"<<endl;

    cout<<"c3=";c3.display();
```



ADITYA
UNIVERSITY

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
c3=c1-c2;
cout<<"\nAfter c1-c2:"<<endl;

cout<<"c3=";

c3.display();

return 0;
}
```

Output:

Roll No:24B11AI439
original complex numbers:

$c1=3+4i$

$c2=1+2i$

After unary - on c1:

$c3=-3-4i$

After unary ++ on c1:

$c1=4+5i$

After $c1+c2$:

$c3=5+7i$

After $c1-c2$:

$c3=3+3i$



A D I T Y A
UNIVERSITY

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

Program 6.2:

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

Aim

To demonstrate operator overloading for unary and binary operators using friend functions in C++.

Description

- Unary Operator Overloading: We'll overload the unary - operator to negate the value of an object.
- Binary Operator Overloading: We'll overload the binary + operator to add two objects.

Unlike member functions, friend functions are declared with the friend keyword and are defined outside the class. They can access private members of the class.

Syntax

friend return_type operator op(const ClassName&);

friend return_type operator op(const ClassName&, const ClassName&);

Program:.

```
#include <iostream>

using namespace std;

class complex {
private:
    float real, image;
public:
    complex(float r = 0, float i = 0) : real(r), image(i) {}

    void display() const {
        cout << real << "+" << image << "i" << endl;
    }
};
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
}
```

```
friend complex operator-(const complex& c);
```

```
friend complex operator++(complex& c);
```

```
friend complex operator+(const complex& a, const complex& b);
```

```
friend complex operator-(const complex& a, const complex& b);
```

```
};
```

```
complex operator-(const complex& c) {
```

```
    return complex(-c.real, -c.image);
```

```
}
```

```
complex operator++(complex& c) {
```

```
    ++c.real;
```

```
    ++c.image;
```

```
    return c;
```

```
}
```



ADITYA
UNIVERSITY

```
complex operator+(const complex& a, const complex& b) {
```

```
    return complex(a.real + b.real, a.image + b.image);
```

```
}
```

```
complex operator-(const complex& a, const complex& b) {
```

```
    return complex(a.real - b.real, a.image - b.image);
```

```
}
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
int main() {
    complex c1(3, 4), c2(1, 2), c3;
    cout<<"Roll No:24B11AI439"<<endl;

    cout << "Original complex numbers:" << endl;
cout << "c1 = "; c1.display();

    cout << "c2 = "; c2.display();
    c3 = -c1;

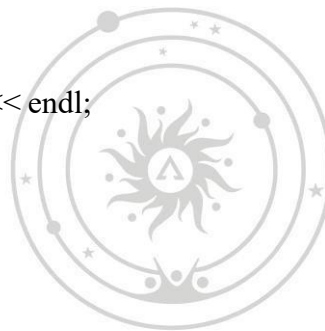
    cout << "\nAfter unary - on c1:" << endl;
    cout << "c3 = ";
    c3.display();
    ++c1;

    cout << "\nAfter unary ++ on c1:" << endl;
    cout << "c1 = "; c1.display();

    c3 = c1 + c2;
    cout << "\nAfter c1 + c2:" << endl;
    cout << "c3 = "; c3.display();

    c3 = c1 - c2;
    cout << "\nAfter c1 - c2:" << endl;
    cout << "c3 = "; c3.display();

    return 0;
}
```



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

Output:

Roll No:24B11AI439

Original complex numbers:

$$c1 = 4+5i$$

$$c2 = 1+2i$$

After unary - on c1:

$$c3 = -4+-5i$$

After unary ++ on c1:

$$c1 = 5+6i$$

After c1 + c2:

$$c3 = 7+8i$$

After c1 - c2:

$$c3 = 4+4i$$



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

WEEK-7

Program 7.1:

Develop C++ programs to demonstrate different forms of inheritance

Aim

To demonstrate the implementation and behavior of various forms of inheritance in C++ including single, multiple, multilevel, hierarchical, and hybrid inheritance.

Description

Inheritance allows a class (derived class) to acquire properties and behaviors (data and functions) from another class (base class). This promotes code reusability, modularity, and extensibility in object-oriented programming.

Types of Inheritance:

1. Single Inheritance: One derived class inherits from one base class.

Syntax:

```
class Base { // members };  
class Derived :  
public Base { // inherits Base };
```

Program:.

```
#include<iostream>  
using namespace std;  
class vehicle{  
    public:  
        vehicle(){  
            cout<<"This is a vehicle"<<endl;}  
  
};  
class car:public vehicle  
{
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
public:
    car(){
        cout<<"This vehicle is a car"<<endl;
    }
};

int main()
{
    cout<<"Roll No:24B11AI439"<<endl;
    car obj;
    return 0;
}
```

Output:

Roll No:24B11AI439

This is a vehicle

This vehicle is a car



2. Multiple Inheritance: One derived class inherits from more than one base class.

Syntax:

```
class Base1 { // members };
class Base2 { // members };
class Derived :
    public Base1, public Base2 { // inherits both Base1 and Base2 };
```

Program:

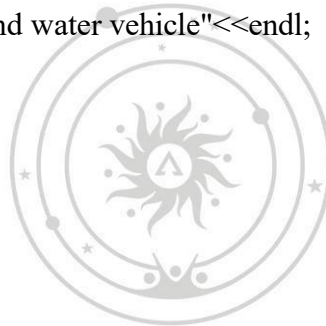
```
#include<iostream>
using namespace std;
class landvehicle{
public:
    landvehicle(){
        cout<<"This is a land vehicle"<<endl;
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
}  
};  
class watervehicle{  
public:  
watervehicle(){  
cout<<"This is a water vehicle"<<endl;  
}  
};  
class amphibiousvehicle:public watervehicle,public landvehicle{  
public:  
amphibiousvehicle(){  
    cout<<"This is an both land and water vehicle"<<endl;  
}  
};  
int main(){  
cout<<"Roll No:24B11AI439"<<endl;  
    amphibiousvehicle obj;  
    return 0;  
}
```



A D I T Y A
UNIVERSITY

Output:

Roll No:24B11AI439

This is a water vehicle

This is a land vehicle

This is an both land and water vehicle.

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

3. Multilevel Inheritance: A class is derived from a class which is also derived from another class.

Syntax:

```
class Base { // members };
```

```
class Intermediate : public Base { // inherits Base };
```

```
class Derived : public Intermediate { // inherits Intermediate (and indirectly Base) };
```

Program:

```
#include<iostream>
```

```
using namespace std;
```

```
class vehicle{
```

```
public:
```

```
vehicle(){
```

```
cout<<"This is a vehicle"<<endl;}
```

```
};
```

```
class fourwheeler:public vehicle{
```

```
public:
```

```
fourwheeler(){
```

```
    cout<<"4 wheeler vehicles"<<endl;
```

```
}
```

```
};
```

```
class car:public fourwheeler{
```

```
public:
```

```
car(){
```

```
    cout<<"This 4 wheeler vehicle is a car";
```

```
}
```

```
};
```

```
int main(){
```



ADITYA
UNIVERSITY

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
cout<<"Roll No:24B11AI439"<<endl;
car obj;

return 0;

}
```

Output:

Roll No:24B11AI439

This is a vehicle

4 wheeler vehicles

This 4 wheeler vehicle is a car.

4. Hierarchical Inheritance: Multiple classes inherit from a single base class.

Syntax:

```
class Base { // members };

class Derived1 : public Base { // inherits Base };

class Derived2 : public Base { // inherits Base };
```

Program:

```
#include<iostream>

using namespace std;

class vehicle{

    public:

        vehicle(){

            cout<<"This is a vehicle"<< endl;

        }

};

class car : public vehicle{

    public:
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
        car(){
            cout<<"This vehicle is a car"<< endl;

        }

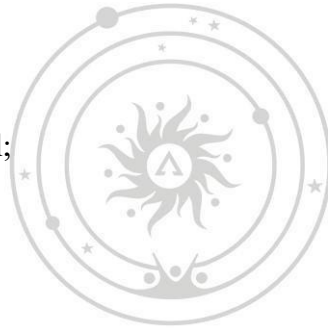
};

class bus : public vehicle{
    public:
        bus(){

            cout<<"This vehicle is bus"<< endl;

        }
};

int main(){
    cout<<"Roll No:24B11AI439"<<endl;
        car obj1;
        bus obj2;
}
```



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

Output:

Roll No:24B11AI439

This is a vehicle

This vehicle is a car

This is a vehicle

This vehicle is bus

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

5. Hybrid Inheritance: A combination of two or more types of inheritance.

Syntax:

```
class A { // base class };  
class B : public A { // inherits A };  
class C : public A { // inherits A };  
class D : public B, public C { // inherits both B and C (diamond problem may occur) };
```

Program:.

```
#include<iostream>  
using namespace std;  
class vehicle{  
public:  
vehicle(){  
cout<<"this is a vehicle"<<endl;  
  
}  
};  
class fare{  
public:  
fare(){  
    cout<<"fare of a vehicle"<<endl;  
  
}  
};  
class car:public vehicle{  
public:  
car(){  
    cout<<"this vehicle is a car"<<endl;  
  
}
```



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
};  
class bus:public vehicle,public fare{  
public:  
    bus(){  
        cout<<"this vehicle is a bus with fare"<<endl;  
  
    }  
};  
int main()  
{  
    cout<<"Roll No:24B11AI439"<<endl;  
    bus obj2;  
  
}
```

Output:

Roll No:24B11AI439

this is a vehicle

fare of a vehicle

this vehicle is a bus with fare.



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

Program 7.2:

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

Aim

To illustrate the order in which constructors and destructors are executed in a C++ program involving inheritance.

Description

In C++:

- Constructors are called from base to derived.
- Destructors are called from derived to base. This ensures proper initialization and cleanup of resources.

Syntax:

```
class Base {  
public:  
    Base() { /* constructor */ }  
    ~Base() { /* destructor */ };  
    class Derived : public Base {  
    public:  
        Derived() { /* constructor */ }  
        ~Derived() { /* destructor */ };  
    }  
};
```



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

Program:

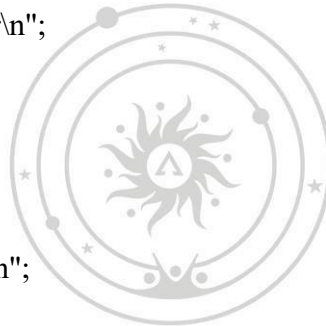
```
#include<iostream>  
using namespace std;  
class parent{  
public:  
    parent()  
{  
        cout<<"parent class constructor\n";  
    }  
};
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
}  
~parent()  
{  
cout<<"parent class destructor\n";  
}  
};  
class child :public parent{  
public:  
child()  
{  
    cout<<"child class constructor\n";  
}  
~child()  
{  
    cout<<"child class destructor\n";  
}  
};  
int main()  
{  
cout<<"Roll No:24B11AI439"<<endl;  
child c;  
return 0;  
}
```



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

Output:

Roll No:24B11AI439
parent class construtor
child class constructor
child class destructor
parent class destructor.

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

WEEK 8

Program 8.1:

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

Aim:

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

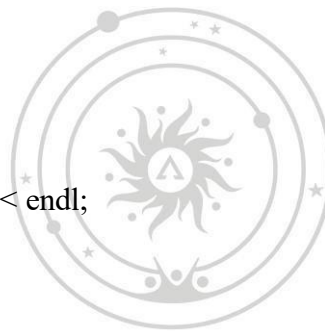
Description:

In C++, pointers can be used to access and manipulate class objects by storing the address of an object and using the arrow operator (->) to call its methods or access members. This is useful for dynamic memory allocation, passing objects to functions, and working with polymorphism.

Syntax:

```
class MyClass {  
public:  
    void show() {  
        cout << "Accessed via pointer!" << endl;  
    }  
};
```

```
int main() {  
    MyClass obj;  
    MyClass* ptr = &obj;  
    ptr->show();  
    return 0;  
}
```



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:  
        string name;  
        int age;  
    public:  
        void setdata(string n,int a)  
{  
            name=n;  
            age=a;  
        }  
}
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
void display()
{
    cout<<"name:"<<name<<endl;
    cout<<"age:"<<age<<endl;
}
};
int main()
{
    cout<<"Roll No:24B11AI439"<<endl;
    student s1;
    s1.setdata("tony",20);
    s1.display();
    student *ptr;
    ptr=&s1;
    ptr->display();
    student *ptr2=new student;
    ptr2->setdata("bruce",22);
    ptr2->display();
    delete ptr2;
    return 0;
}
```



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

Output:

Roll No:24B11AI439

name:tony

age:20

name:tony

age:20

name:bruce

age:22

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

Program 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:

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:

Virtual base classes in C++ are used in multiple inheritance to avoid duplication of a common base class. When two classes inherit from the same base and a third class inherits from both, virtual inheritance ensures only one copy of the base class exists, resolving ambiguity and preventing the "diamond problem."

Syntax:

```
class A {  
};  
  
class B : virtual public A {  
};  
  
class C : virtual public A {  
};  
  
class D : public B, public C {  
};
```



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

Program:

```
#include <iostream>  
using namespace std;  
class A {  
  
public:  
  
A()  
{  
    cout<<"A is constructor"<<endl;  
}  
~A()  
{  
  
    cout<<"A is destructor"<<endl;  
}
```

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

```
void show()
{
    cout << "Hello from A \n";
}
};
```

```
class B : public virtual A
{
public:
    B()
    {
        cout<<"B is constructor"<<endl;
    }
    ~B()
    {
        cout<<"B is destructor"<<endl;
    }
};
```

```
class C : public virtual A {
public:
    C()
    {
        cout<<"C is constructor"<<endl;
    }
    ~C()
    {
        cout<<"C is destructor"<<endl;
    }
};
```

```
class D : public B, public C {
public:
    D()
    {
        cout<<"D is constructor"<<endl;
    }
    ~D()
    {
        cout<<"D is destructor"<<endl;
    }
};
```

```
int main()
{
    cout<<"Roll No:24B11AI439"<<endl;
    D obj;
    obj.show();
    return 0;
}
```



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

Date:

Roll No.:

--	--	--	--	--	--	--	--	--	--

Output:

Roll No:24B11AI439

A is constructor

B is constructor

C is constructor

D is constructor

Hello from A

D is destructor

C is destructor

B is destructor

A is destructor



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