DELHI TECHNOLOGICAL UNIVERSITY

(Formerly Delhi College of Engineering)
Shahabad Daulatpur, Bawana Road, Delhi 110042

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Lab Code: CO-203

Submitted To:

Mr. Nipun Bansal

Submitted By:

Rajat Sharda

2K21/EC/176

INDEX

EXPERIMENT-1

AIM: Write a C++ Program to print your personal details by taking input from the user.

THEORY:

Class:

The building block of C++ that leads to Object Oriented programming is a Class It is a user- defined data type, which holds its own data members and member functions, which can be accessed and used by creating an instance of that class. A class is like a blueprint for an object.

Object:

An Object is an identifiable entity with some characteristics and behavior. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e., an object is created) memory is allocated.

ALGORITHM:

- Start.
- Create a class having salary, id and age as public members.
- Enter the number of employees.
- Create an employee array followed by sum of their salary.
- Output the total salary.
- End.

CODE –

```
//Rajat Sharda
//2K21/EC/176

#include<bits/stdc++.h>
using namespace std;

class Employee{
    public:
    int sal; int id; int age;
};
void sumSal(Employee emp[], int n)
{
    int sum = 0;
    for(int i = 0; i<n; i++)
        sum += emp[i].sal;
    cout << "Sum of salaries = " << sum << endl;
}
int main()
{
    init();
    cout << "Number of Employees => " << endl;
    int n; cin >> n;
    Employee emp[n];
    for(int i = 0; i < n; i++)
    {
        cout << "Emp id - ";
        cin>> emp[i].id;
        cout << endl << "Emp age - ";
```

```
        cin>> emp[i].age;
        cout << endl << " Emp Salary - ";
        cin >> emp[i].sal;
    }
    sumSal(emp, n);
    return 0;
}
```

OUTPUT :

```
#include<bits/stdc++.h>
#include<iostream>
using namespace std;

class Employee{
    public:
    int sal; int id; int age;
};

void sumSal(Employee emp[], int n){int sum = 0;
for(int i = 0; i<n; i++){sum += emp[i].sal;
}
cout << "Sum of salaries = " << sum << endl;
}

int main(){
cout << "Number of Employees => " << endl;int n; cin >> n;

Employee emp[n];

for(int i=0; i<n; i++){ cout
<< "Emp id - ";cin>> emp[i].id;

cout << endl << "Emp age - ";cin>> emp[i].age;

cout << endl << " Emp Salary - ";cin >> emp[i].sal;
}

sumSal(emp, n);

}
```

```
Number of Employees =>
4
Emp id - 121

Emp age - 32

 Emp Salary - 3000
Emp id - 221

Emp age - 12

 Emp Salary - 3333
Emp id - 6969

Emp age - 23

 Emp Salary - 6900
Emp id - 132

Emp age - 18

 Emp Salary - 6000
Sum of salaries = 19233
```

EXPERIMENT-2

AIM : Write a Program to return absolute value of variable types integer and float using function overloading.

THEORY:
int absolute(int num) takes an integer as an argument and returns its absolute value using abs().
float absolute(float num) takes a float as an argument and returns its absolute value using fabs().
In the main function, we demonstrate how to use these overloaded functions to find the absolute values of both integer and float variables. When you run the program, it will return the absolute values of the given numbers and display them.

ALGORITHM:

1. Start
2. Declare two overloaded functions: absolute(int num) and absolute(float num) to find the absolute value of an integer and a float, respectively.
3. In the absolute(int num) function:
   - Receive an integer num as a parameter.
   - Use the abs() function to calculate the absolute value of num.
   - Return the absolute value.
4. In the absolute(float num) function:
   - Receive a float num as a parameter.
   - Use the fabs() function to calculate the absolute value of num.
   - Return the absolute value.
5. In the main function:
   - Declare an integer variable intNumber and a float variable floatNumber.
   - Initialize intNumber with a negative integer value (e.g., -5) and floatNumber with a negative float value (e.g., -7.5).
   - Call the absolute(intNumber) function to calculate the absolute value of intNumber and store it in absInt.
   - Call the absolute(floatNumber) function to calculate the absolute value of floatNumber and store it in absFloat.
   - Display the results, showing the absolute values for both integer and float.
6. End

This algorithm outlines the step-by-step process of the program to find and display the absolute values of both integer and float variables using function overloading.

CODE :

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;

class Employee {
public:
    int EmpNumber;
    string EmpName;

    // Member function to input employee data
    void getData() {
        cout << "Enter Employee Number: ";
        cin >> EmpNumber;
```

```
        cout << "Enter Employee Name: ";
        cin.ignore();  // Clear the input buffer
        getline(std::cin, EmpName);
    }

    // Member function to display employee data
    void display() {
        cout << "Employee Number: " << EmpNumber << endl;
        cout << "Employee Name: " << EmpName << endl;
    }
};

int main() {
    init();
    const int numEmployees = 6;
    Employee employees[numEmployees];

    // Input data for each employee
    for (int i = 0; i < numEmployees; i++) {
        cout << "Enter details for Employee " << i + 1 << ":\n";
        employees[i].getData();
    }

    cout << "Employee Details:\n";

    // Display data for each employee
    for (int i = 0; i < numEmployees; i++) {
        cout << "Employee " << i + 1 << ":\n";
        employees[i].display();
        cout << endl;
    }

    return 0;
}
```

OUTPUT:

```cpp
#include <iostream>
#include <cmath>

// Function to return the absolute value of an integer
int absolute(int num) {
    return std::abs(num);
}

// Function to return the absolute value of a float
float absolute(float num) {
    return std::fabs(num);
}

int main() {
    int intNumber = -5;
    float floatNumber = -7.5;
    int absInt = absolute(intNumber);
    float absFloat = absolute(floatNumber);

    std::cout << "Absolute value of integer: " << absInt << std::endl;
    std::cout << "Absolute value of float: " << absFloat << std::endl;

    return 0;
}
```

```
Absolute value of integer: 5
Absolute value of float: 7.5
```

Rajat Sharda 2K21/EC/176

EXPERIMENT-3

AIM: Create a class "employee" that has "EmpNumber" and "EmpName" as data members and member functions get data() to input data display() to output data. Write a main function to create an array of "Employee" objects. Accept and print the details of atleast 6 employees.

THEORY:

In this C++ program, we create a class named "Employee" with two data members, "EmpNumber" (to store the employee number) and "EmpName" (to store the employee name). The program allows the user to input and display the details of at least 6 employees. Here's an explanation of the program's components:

1. Employee Class:
    - The "Employee" class is defined to encapsulate employee data. It has the following data members:
        - EmpNumber: An integer variable to store the employee number.
        - EmpName: A string variable to store the employee's name.
    - The class also includes member functions:
        - getData(): A function that allows the user to input employee data, including the employee number and name.
        - display(): A function that displays the employee's details, including the employee number and name.
2. Main Function:
    - In the main function:
        - An array of "Employee" objects is created to store the details of at least 6 employees.
        - A loop is used to iterate through each element of the array.
        - Inside the loop, the getData() function is called for each object to input the employee's data, including the employee number and name.
        - Another loop is used to display the details of each employee by calling the display() function for each object.
3. Execution Flow:
    - The program prompts the user to input the employee details for each employee.
    - It creates and populates objects of the "Employee" class with the provided data.
    - The program then displays the details of each employee, allowing the user to view the information that has been entered.

This program demonstrates the use of object-oriented programming in C++ by creating a class to represent employees' data and utilizing member functions to input and display this data. It also showcases the concept of encapsulation, where the data and related functions are contained within a class, providing a structured and organized approach to managing employee details.

ALGORITHM:

1. Start

2. Define a class named "Employee" with the following data members:

    - EmpNumber: Integer to store the employee number.

    - EmpName: String to store the employee name.

3. Define member functions within the "Employee" class:

    - getData(): Function to input employee data, prompting the user to enter the employee number and name.

    - display(): Function to display the employee's details, including EmpNumber and EmpName.

4. In the main function:
    - Create an array of "Employee" objects to store employee details (e.g., Employee employees[6] to store details for 6 employees).
    - Use a loop to iterate through each element of the array to accept data for each employee using the getData() function.
    - Inside the loop, call the getData() function for each object to input the employee's data.
    - Use another loop to display the details of each employee by calling the display() function for each object.
    - End the program.

CODE :

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;

class Employee {
public:
    int EmpNumber;
    string EmpName;

    // Member function to input employee data
    void getData() {
        cout << "Enter Employee Number: ";
        cin >> EmpNumber;
        cout << "Enter Employee Name: ";
        cin.ignore();  // Clear the input buffer
        getline(std::cin, EmpName);
    }

    // Member function to display employee data
    void display() {
        cout << "Employee Number: " << EmpNumber << endl;
        cout << "Employee Name: " << EmpName << endl;
    }
};

int main() {
    init();
    const int numEmployees = 6;
    Employee employees[numEmployees];

    // Input data for each employee
    for (int i = 0; i < numEmployees; i++) {
        cout << "Enter details for Employee " << i + 1 << ":\n";
        employees[i].getData();
    }

    cout << "Employee Details:\n";
```

```
    // Display data for each employee
    for (int i = 0; i < numEmployees; i++) {
        cout << "Employee " << i + 1 << ":\n";
        employees[i].display();
        cout << endl;
    }
    return 0;
}
```

OUTPUT :

```
#include <iostream>
#include <string>
class Employee {
public:
    int EmpNumber;
    std::string EmpName;
    void getData() {
        std::cout << "Enter Employee Number: ";
        std::cin >> EmpNumber;
        std::cout << "Enter Employee Name: ";
        std::cin.ignore();  // Clear the input buffer
        std::getline(std::cin, EmpName);
    }
    void display() {
        std::cout << "Employee Number: " << EmpNumber << std::endl;
        std::cout << "Employee Name: " << EmpName << std::endl;
    }
};

int main() {
    const int numEmployees = 6;
    Employee employees[numEmployees];
    for (int i = 0; i < numEmployees; i++) {
        std::cout << "Enter details for Employee " << i + 1 << ":\n";
        employees[i].getData();
    }
    std::cout << "Employee Details:\n";
    for (int i = 0; i < numEmployees; i++) {
        std::cout << "Employee " << i + 1 << ":\n";
        employees[i].display();
        std::cout << std::endl;
    }
    return 0;
}
```

```
Enter details for Employee 1:
Enter Employee Number: 125
Enter Employee Name: Kurush
Enter details for Employee 2:
Enter Employee Number: 132
Enter Employee Name: Rajat
Enter details for Employee 3:
Enter Employee Number: 176
Enter Employee Name: Madhav
Enter details for Employee 4:
Enter Employee Number: 240
Enter Employee Name: Divyanshi Bhalla
Enter details for Employee 5:
Enter Employee Number: 13
Enter Employee Name: Shelly Bhatia
Enter details for Employee 6:
Enter Employee Number: 23
Enter Employee Name: Rishit Gaur
Employee Details:
Employee 1:
Employee Number: 125
Employee Name: Kurush

Employee 2:
Employee Number: 132
Employee Name: Rajat

Employee 3:
Employee Number: 176
Employee Name: Madhav

Employee 4:
Employee Number: 240
Employee Name: Divyanshi Bhalla

Employee 5:
Employee Number: 13
Employee Name: Shelly Bhatia

Employee 6:
Employee Number: 23
Employee Name: Rishit Gaur
```

EXPERIMENT-4

AIM- Write a C++ program to swap two numbers by both call by value and call by reference mechanism

THEORY:

Swapping by Value: In the "call by value" mechanism, the values of the actual parameters are copied into the formal parameters of the function. Swapping is done locally within the function, and it doesn't affect the original variables outside the function.
Swapping by Reference: In the "call by reference" mechanism, the memory addresses (references) of the actual parameters are passed to the function. This allows the function to directly modify the values at those memory addresses, which results in swapping the original variables.

Algorithm (Swap by Value):

1. Start
2. Declare two integer variables, **a** and **b**.
3. Prompt the user to enter the values of **a** and **b**.
4. Call the **swapByValue** function, passing **a** and **b** as arguments.
5. Inside the **swapByValue** function:
    - Create a temporary variable, **temp**, and assign the value of **a** to **temp**.
    - Assign the value of **b** to **a**.
    - Assign the value of **temp** to **b**.
6. Display the swapped values of **a** and `b.
7. End.

Algorithm (Swap by Reference):

1. Start
2. Declare two integer variables, **a** and **b**.
3. Prompt the user to enter the values of **a** and **b**.
4. Call the **swapByReference** function, passing **a** and **b** as references.
5. Inside the **swapByReference** function:
    - Swap the values of the variables **a** and **b** directly using references.
6. Display the swapped values of **a** and `b.
7. End.

CODE:

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;
void swapByValue(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

// Function to swap two numbers by reference
void swapByReference(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
```

```
    init();
    int num1, num2;
    // Input two numbers
    cout << "Enter the first number: ";
    cin >> num1;
    cout << "Enter the second number: ";
    cin >> num2;

    // Swap by value
    int a = num1, b = num2;
    swapByValue(a, b);
    cout << "Swapped values by value: " << a << ", " << b << "\n";

    // Swap by reference
    swapByReference(num1, num2);
    cout << "Swapped values by reference: " << num1 << ", " << num2
<< "\n";

    return 0;
}
```

OUTPUT :

```cpp
#include <iostream>

void swapByValue(int a, int b) {
    int temp = a;
    a = b;
    b = temp;
}

void swapByReference(int &a, int &b) {
    int temp = a;
    a = b;
    b = temp;
}

int main() {
    int num1, num2;

    std::cout << "Enter the first number: ";
    std::cin >> num1;
    std::cout << "Enter the second number: ";
    std::cin >> num2;

    int a = num1, b = num2;
    swapByValue(a, b);
    std::cout << "Swapped values by value: " << a << ", " << b << std::endl;

    swapByReference(num1, num2);
    std::cout << "Swapped values by reference: " << num1 << ", " << num2 << std::endl;

    return 0;
}
```

```
Enter the first number: 6
Enter the second number: 9
Swapped values by value: 6, 9
Swapped values by reference: 9, 6
```

EXPERIMENT-5

AIM: Write a C++ program to create a simple banking system in which the initial balance and the rate of interest are read from the keyboard and these values are initialised using constructor

THEORY:

- The "BankAccount" class encapsulates the data and operations related to a bank account.
- It has the following data members:
  - accountNumber: An integer to store the account number.
  - balance: A double to store the initial balance.
  - interestRate: A double to store the interest rate.
- The class also includes member functions:
  - Constructor: Initializes the account number, initial balance, and interest rate.
  - deposit(): Allows the user to deposit money into the account.
  - withdraw(): Allows the user to withdraw money from the account.
  - calculateInterest(): Calculates and displays the interest earned based on the balance and interest rate.
  - displayAccountDetails(): Displays the account number, balance, and interest rate.

2. Main Function:
   - In the main function:
     - The user is prompted to enter the account number, initial balance, and interest rate.
     - An object of the "BankAccount" class is created, passing the user-entered values to the constructor for initialization.
     - A menu is displayed to allow the user to choose from various banking operations.
     - The program uses a loop to repeatedly display the menu and perform operations based on the user's choice.
     - The user can deposit, withdraw, calculate interest, or display account details. The loop continues until the user decides to exit.

ALGORITHM:

1. Start
2. Define a class named "BankAccount" to represent a bank account with the following data members:
   - **accountNumber**: Integer to store the account number.
   - **balance**: Double to store the initial balance.
   - **interestRate**: Double to store the rate of interest.
3. Define a constructor within the "BankAccount" class to initialize the account number, initial balance, and interest rate. This constructor should take these values as parameters and initialize the corresponding data members.
4. Define member functions within the "BankAccount" class to perform operations on the account, such as:
   - **deposit()**: Function to allow the user to deposit money into the account.
   - **withdraw()**: Function to allow the user to withdraw money from the account.
   - **calculateInterest()**: Function to calculate and display the interest earned based on the balance and interest rate.
   - **displayAccountDetails()**: Function to display the account details, including the account number, balance, and interest rate.
5. In the **main** function:
   - Prompt the user to enter the account number, initial balance, and interest rate.
   - Create an object of the "BankAccount" class, passing the user-entered values to the constructor for initialization.
   - Display a menu to allow the user to perform operations on the account, such as deposit, withdrawal, interest calculation, and account details display.
   - Implement a loop to repeatedly display the menu and perform operations based on the user's choice until the user decides to exit.
6. End

CODE:

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;
class BankAccount {
public:
    int accountNumber;
    double balance;
    double interestRate;
    // Constructor to initialize account details
    BankAccount(int accNum, double initBalance, double rate)
        : accountNumber(accNum), balance(initBalance), interestRate(rate)
{}
    // Function to deposit money into the account
    void deposit(double amount)
    {
        balance += amount;
    }
    // Function to withdraw money from the account
    void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            cout << "Insufficient funds." << endl;
        }
    }
    // Function to calculate and display interest earned
    void calculateInterest() {
        double interest = balance * interestRate / 100.0;
        cout << "Interest earned: $" << fixed << std::setprecision(2) <<
interest << endl;
    }
    // Function to display account details
    void displayAccountDetails() {
        cout << "Account Number: " << accountNumber << endl;
        cout << "Balance: $" << fixed << setprecision(2) << balance <<
endl;
        cout << "Interest Rate: " << interestRate << "%" << endl;
    }
};
int main() {
    init();
    int accountNum;
    double initialBalance, interestRate;
    // Input account details
    cout << "Enter Account Number: ";
    cin >> accountNum;
    cout << "Enter Initial Balance: $";
    cin >> initialBalance;
    cout << "Enter Interest Rate (%): ";
    cin >> interestRate;
    // Create a BankAccount object with user-entered values
    BankAccount account(accountNum, initialBalance, interestRate);
```

```
    int choice;
    double amount;
    // Display a menu for banking operations
    do {
        cout << "\nBanking Menu:\n";
        cout << "1. Deposit\n";
        cout << "2. Withdraw\n";
        cout << "3. Calculate Interest\n";
        cout << "4. Display Account Details\n";
        cout << "5. Exit\n";
        cout << "Enter your choice: ";
        cin >> choice;
        switch (choice) {
            case 1:
                cout << "Enter deposit amount: $";
                cin >> amount;
              account.deposit(amount);
                break;
            case 2:
                cout << "Enter withdrawal amount: $";
                cin >> amount;
                account.withdraw(amount);
                break;
            case 3:
                account.calculateInterest();
                break;
            case 4:
                account.displayAccountDetails();
                break;
            case 5:
                cout << "Exiting the program. Goodbye!" << std::endl;
                break;
            default:
                cout << "Invalid choice. Please try again." << std::endl;
        }
    } while (choice != 5);
    return 0;
}
```

OUTPUT :

```cpp
#include <iostream>
#include <iomanip>
class BankAccount {
public:
    int accountNumber;
    double balance;
    double interestRate;

    BankAccount(int accNum, double initBalance, double rate)
        : accountNumber(accNum), balance(initBalance), interestRate(rate) {}

    void deposit(double amount) {
        balance += amount;
    }

    void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            std::cout << "Insufficient funds." << std::endl;
        }
    }

    void calculateInterest() {
        double interest = balance * interestRate / 100.0;
        std::cout << "Interest earned: $" << std::fixed << std::setprecision(2) << interest << std::endl;
    }

    void displayAccountDetails() {
        std::cout << "Account Number: " << accountNumber << std::endl;
        std::cout << "Balance: $" << std::fixed << std::setprecision(2) << balance << std::endl;
        std::cout << "Interest Rate: " << interestRate << "%" << std::endl;
    }
};
```

```cpp
int main() {
    int accountNum;
    double initialBalance, interestRate;

    std::cout << "Enter Account Number: ";
    std::cin >> accountNum;
    std::cout << "Enter Initial Balance: $";
    std::cin >> initialBalance;
    std::cout << "Enter Interest Rate (%): ";
    std::cin >> interestRate;

    BankAccount account(accountNum, initialBalance, interestRate);

    int choice;
    double amount;

    do {
        std::cout << "\nBanking Menu:\n";
        std::cout << "1. Deposit\n";
        std::cout << "2. Withdraw\n";
        std::cout << "3. Calculate Interest\n";
        std::cout << "4. Display Account Details\n";
        std::cout << "5. Exit\n";
        std::cout << "Enter your choice: ";
        std::cin >> choice;

        switch (choice) {
            case 1:
                std::cout << "Enter deposit amount: $";
                std::cin >> amount;
                account.deposit(amount);
                break;
            case 2:
                std::cout << "Enter withdrawal amount: $";
                std::cin >> amount;
                account.withdraw(amount);
                break;
            case 3:
                account.calculateInterest();
                break;
            case 4:
                account.displayAccountDetails();
                break;
            case 5:
                std::cout << "Exiting the program. Goodbye!" << std::endl;
                break;
            default:
                std::cout << "Invalid choice. Please try again." << std::endl;
        }
    } while (choice != 5);

    return 0;
}
```

```
Enter Account Number: 12345
Enter Initial Balance: $500
Enter Interest Rate (%): 10

Banking Menu:
1. Deposit
2. Withdraw
3. Calculate Interest
4. Display Account Details
5. Exit
Enter your choice: 1
Enter deposit amount: $300

Banking Menu:
1. Deposit
2. Withdraw
3. Calculate Interest
4. Display Account Details
5. Exit
Enter your choice: 2
Enter withdrawal amount: $300

Banking Menu:
1. Deposit
2. Withdraw
3. Calculate Interest
4. Display Account Details
5. Exit
Enter your choice: 3
Interest earned: $50.00
```

```
Banking Menu:
1. Deposit
2. Withdraw
3. Calculate Interest
4. Display Account Details
5. Exit
Enter your choice: 4
Account Number: 12345
Balance: $500.00
Interest Rate: 10.00%

Banking Menu:
1. Deposit
2. Withdraw
3. Calculate Interest
4. Display Account Details
5. Exit
Enter your choice: 5
Exiting the program. Goodbye!
```

EXPERIMENT-6

AIM : Write a program to accept five different numbers by creating a class called friendfunc1() and friendfunc2() and calculate the average of these numbers by passing object of the class to friend function.

THEORY:

In this C++ program, we create a class named "NumberCalculator" to accept and store five different numbers. We also declare two friend functions, "friendfunc1" and "friendfunc2," to perform calculations on the numbers.

- The "friendfunc1" function calculates the sum of the five numbers by accessing the private data members of the "NumberCalculator" class.
- The "friendfunc2" function calculates the average of the five numbers by dividing the sum (obtained from "friendfunc1") by 5.

These friend functions are declared outside the class but have access to the private members of the class, thanks to the "friend" keyword.

ALGORITHM:

1. Start
2. Define a class called "NumberCalculator."
3. Declare private data members to store five different numbers.
4. Declare a constructor to initialize the five numbers when an object is created.
5. Declare two friend functions, "friendfunc1" and "friendfunc2," outside the class.
6. In "friendfunc1," calculate the sum of the five numbers by accessing the private data members of the "NumberCalculator" class.
7. In "friendfunc2," calculate the average of the five numbers by dividing the sum (obtained from "friendfunc1") by 5.
8. Inside the "main" function:
    - Create an object of the "NumberCalculator" class and pass the five numbers as arguments to the constructor.
    - Call "friendfunc1" and "friendfunc2" by passing the object of the class as arguments.
    - Display the average calculated in "friendfunc2."
9. End.

CODE:

```
 //Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;

class NumberCalculator {
private:
    double num1, num2, num3, num4, num5;
public:
```

```
    NumberCalculator(double n1, double n2, double n3, double n4, double
n5)
        : num1(n1), num2(n2), num3(n3), num4(n4), num5(n5) {}
    friend double friendfunc1(NumberCalculator obj);
    friend double friendfunc2(NumberCalculator obj);
};
// Friend function to calculate the sum of numbers
double friendfunc1(NumberCalculator obj) {
    return obj.num1 + obj.num2 + obj.num3 + obj.num4 + obj.num5;
}
// Friend function to calculate the average of numbers
double friendfunc2(NumberCalculator obj) {
    double sum = friendfunc1(obj);
    return sum / 5;
}
int main() {
    init();
    double num1, num2, num3, num4, num5;
    // Input five different numbers
    cout << "Enter five different numbers: ";
    cin >> num1 >> num2 >> num3 >> num4 >> num5;
    NumberCalculator calculator(num1, num2, num3, num4, num5);
    double average = friendfunc2(calculator);
    cout << "Average of the five numbers: " << average << endl;
    return 0;

     return 0;

}
```

Output:

```cpp
#include <iostream>

class NumberCalculator {
private:
    double num1, num2, num3, num4, num5;

public:
    NumberCalculator(double n1, double n2, double n3, double n4, double n5)
        : num1(n1), num2(n2), num3(n3), num4(n4), num5(n5) {}

    friend double friendfunc1(NumberCalculator obj);
    friend double friendfunc2(NumberCalculator obj);
};

double friendfunc1(NumberCalculator obj) {
    return obj.num1 + obj.num2 + obj.num3 + obj.num4 + obj.num5;
}

double friendfunc2(NumberCalculator obj) {
    double sum = friendfunc1(obj);
    return sum / 5;
}
```

```cpp
int main() {
    double num1, num2, num3, num4, num5;

    std::cout << "Enter five different numbers: ";
    std::cin >> num1 >> num2 >> num3 >> num4 >> num5;

    NumberCalculator calculator(num1, num2, num3, num4, num5);

    double average = friendfunc2(calculator);
    std::cout << "Average of the five numbers: " << average << std::endl;

    return 0;
}
```

```
Enter five different numbers: 4 3 6 7 9
Average of the five numbers: 5.8
```

EXPERIMENT-7

AIM - WAP that takes students marks and display it's average using friend function.
THEORY:

In this C++ program, we create a class named "Student" to represent student information. The class has a private data member, "marks," to store the student's marks. We declare a friend function, "calculateAverage," which calculates the average of the student's marks.

The "calculateAverage" function has access to the private "marks" data member of the "Student" class due to the "friend" keyword.

In the **main** function, we create an object of the "Student" class, initialize it with the student's marks, and then call the friend function to calculate and display the average of these marks.

Algorithm:

1. Start
2. Define a class called "Student" to encapsulate student information.
3. Declare a private data member "marks" to store the student's marks.
4. Declare a constructor to initialize the "marks" when a "Student" object is created.
5. Declare a friend function "calculateAverage" outside the class to calculate the average of marks.
6. Inside the "calculateAverage" function, access the private "marks" data member of the "Student" class and calculate the average.
7. Inside the "main" function:
    - Create an object of the "Student" class and pass the student's marks as an argument to the constructor.
    - Call the "calculateAverage" function by passing the "Student" object as an argument.
    - Display the average calculated in the "calculateAverage" function.
8. End.

CODE:

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;
class Student {
private:
    double num1, num2, num3, num4, num5;
public:
    Student(double n1, double n2, double n3, double n4, double n5)
        : num1(n1), num2(n2), num3(n3), num4(n4), num5(n5) {}
    friend double Sum(Student obj);
    friend double Avg(Student obj);
};
// Friend function to calculate the sum of marks
double Sum(Student obj) {
    return obj.num1 + obj.num2 + obj.num3 + obj.num4 + obj.num5;
}
double Avg(Student obj) {
    double sum = Sum(obj);
    return (sum / 5);
}
int main() {
    init();
```

```
    double num1, num2, num3, num4, num5;
    cout << "Enter marks for five subjects: ";
    cin >> num1 >> num2 >> num3 >> num4 >> num5;
    Student calculator(num1, num2, num3, num4, num5);
    double average = Avg(calculator);
    cout << "Average of the marks: " << average << endl;
    return 0;
}
```

OUTPUT :

```
#include <iostream>

class Student {
private:
    double marks;

public:
    Student(double studentMarks) : marks(studentMarks) {}

    friend double calculateAverage(Student student);
};

// Friend function to calculate the average of student marks
double calculateAverage(Student student) {
    return student.marks;
}

int main() {
    double studentMarks;

    // Input the student's marks
    std::cout << "Enter the student's marks: ";
    std::cin >> studentMarks;

    // Create a Student object and pass the marks to the constructor
    Student student(studentMarks);

    // Calculate and display the average using the friend function
    double average = calculateAverage(student);
    std::cout << "Student's Average Marks: " << average << std::endl;

    return 0;
}
```

```
Enter marks for five subjects: 20 25 23 27 28 21
Average of the marks: 24.6
```

EXPERIMENT-8

AIM : WAP to perform different arithmetic operations using Inline functions

THEORY:

This C++ program defines a class called "ArithmeticOperations" to encapsulate different arithmetic operations. Inline functions are declared for addition, subtraction, multiplication, and division operations. Inline functions are used for small and frequently called functions to improve performance.

- Each inline function takes two operands as parameters and performs the corresponding arithmetic operation (addition, subtraction, multiplication, or division) directly within the function.

In the main function, the user is prompted to input two operands and select an operation. Based on the user's choice, the appropriate inline function is called to perform the operation, and the result is displayed.

ALGORITHM:

1. Start
2. Define a class called "ArithmeticOperations."
3. Declare inline functions for different arithmetic operations (addition, subtraction, multiplication, division).
4. Each inline function takes two operands as parameters and performs the corresponding arithmetic operation inline.
5. Inside each inline function, return the result of the operation.
6. In the main function:
    - Create an object of the "ArithmeticOperations" class.
    - Prompt the user to input two operands and select an operation.
    - Call the appropriate inline function based on the user's choice.
    - Display the result of the operation.
7. End.

CODE :

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;

class ArithmeticOperations {
public:
    // Inline function for addition
    inline double add(double a, double b) {
        return a + b;
    }
    // Inline function for subtraction
    inline double subtract(double a, double b) {
        return a - b;
    }
    // Inline function for multiplication
    inline double multiply(double a, double b) {
        return a * b;
```

```
    }
    // Inline function for division
    inline double divide(double a, double b) {
        if (b == 0) {
            std::cout << "Error: Division by zero." << std::endl;
            return 0.0;
        }
        return a / b;
    }
};
int main() {
    init();
    double operand1, operand2, result;
    int choice;
    // Create an object of the ArithmeticOperations class
    ArithmeticOperations operations;
    // Prompt the user to enter two operands and select an operation
    cout << "Enter operand 1: ";
    cin >> operand1;
    cout << "Enter operand 2: ";
    cin >> operand2;
    cout << "Select an operation:\n";
    cout << "1. Addition\n";
    cout << "2. Subtraction\n";
    cout << "3. Multiplication\n";
    cout << "4. Division\n";
    cout << "Enter your choice (1-4): ";
    cin >> choice;

    // Perform the selected operation and display the result
    switch (choice) {
        case 1:
            result = operations.add(operand1, operand2);
            cout << "Result of addition: " << result << endl;
            break;
        case 2:
            result = operations.subtract(operand1, operand2);
            cout << "Result of subtraction: " << result << endl;
            break;
        case 3:
            result = operations.multiply(operand1, operand2);
            cout << "Result of multiplication: " << result << endl;
            break;
        case 4:
            result = operations.divide(operand1, operand2);
            cout << "Result of division: " << result << endl;
            break;
        default:
            cout << "Invalid choice." << endl;
    }
    return 0;
}
```

OUTPUT:

```cpp
#include <iostream>
class ArithmeticOperations {
public:
    inline double add(double a, double b) {
        return a + b;
    }
    inline double subtract(double a, double b) {
        return a - b;
    }
    inline double multiply(double a, double b) {
        return a * b;
    }
    inline double divide(double a, double b) {
        if (b == 0) {
            std::cout << "Error: Division by zero." << std::endl;
            return 0.0;
        }
        return a / b;
    }
};

int main() {
    double operand1, operand2, result;
    int choice;
    ArithmeticOperations operations;
    std::cout << "Enter operand 1: ";
    std::cin >> operand1;
    std::cout << "Enter operand 2: ";
    std::cin >> operand2;

    std::cout << "Select an operation:\n";
    std::cout << "1. Addition\n";
    std::cout << "2. Subtraction\n";
    std::cout << "3. Multiplication\n";
    std::cout << "4. Division\n";
    std::cout << "Enter your choice (1-4): ";
    std::cin >> choice;


    switch (choice) {
        case 1:
            result = operations.add(operand1, operand2);
            std::cout << "Result of addition: " << result << std::endl;
            break;
        case 2:
            result = operations.subtract(operand1, operand2);
            std::cout << "Result of subtraction: " << result << std::endl;
            break;
        case 3:
            result = operations.multiply(operand1, operand2);
            std::cout << "Result of multiplication: " << result << std::endl;
            break;
        case 4:
            result = operations.divide(operand1, operand2);
            std::cout << "Result of division: " << result << std::endl;
            break;
        default:
            std::cout << "Invalid choice." << std::endl;
    }

    return 0;
}
```

```
Enter operand 1: 2.31
Enter operand 2: 6.969
Select an operation:
1. Addition
2. Subtraction
3. Multiplication
4. Division
Enter your choice (1-4): 1
Result of addition: 9.279
```

```
Enter operand 1: 4.20
Enter operand 2: 6.91
Select an operation:
1. Addition
2. Subtraction
3. Multiplication
4. Division
Enter your choice (1-4): 4
Result of division: 0.607815
```

EXPERIMENT-9

AIM - WAP to perform string operations using operator overloading in C++

THEORY:

In this C++ program, a class called "StringOperations" is defined to perform string operations using operator overloading. Operator overloading allows us to define the behavior of C++ operators for user-defined data types, in this case, strings.

- Private data members of the class store strings to be used as operands in string operations.
- Operator overloading functions are defined for various string operations, such as concatenation and comparison. These functions allow strings to be treated like built-in types, making code more intuitive and readable.

In the main function, objects of the "StringOperations" class are created and initialized with strings. Operator overloading is used to perform string operations, and the results are displayed.

ALGORITHM:

1. Start
2. Define a class called "StringOperations" to encapsulate string operations.
3. Declare private data members to store strings (e.g., str1 and str2) as operands.
4. Define operator overloading functions for various string operations (e.g., concatenation, comparison).
5. Inside each operator overloading function, perform the corresponding string operation.
6. In the main function:
    - Create objects of the "StringOperations" class and initialize them with strings.
    - Use operator overloading to perform string operations.
    - Display the results of the string operations.
7. End

CODE:

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;
class StringOperations {
private:
    string str1;
    string str2;
public:
    StringOperations(const string& s1, const string& s2)
        : str1(s1), str2(s2) {}

    string operator+(const StringOperations& other) {
        return str1 + other.str1;
    }

    // Overload the == operator for string comparison
    bool operator==(const StringOperations& other) {
        return str1 == other.str1;
    }
};

int main() {
    init();
```

```
    string input1, input2;

    // Input two strings
    cout << "Enter string 1: ";
    cin >> input1;
    cout << "Enter string 2: ";
    cin >> input2;

    // Create objects of the StringOperations class
    StringOperations stringObj1(input1, input2);
    StringOperations stringObj2(input2, input1);

    string concatenatedStr = stringObj1 + stringObj2;
    cout << "Concatenated String: " << concatenatedStr << endl;

    if (stringObj1 == stringObj2) {
        cout << "Strings are equal." << endl;
    } else {
        cout << "Strings are not equal." << endl;
    }

    return 0;
}
```

OUTPUT :

```cpp
#include <iostream>
#include <string>
class StringOperations {
private:
    std::string str1;
    std::string str2;
public:
    StringOperations(const std::string& s1, const std::string& s2)
        : str1(s1), str2(s2) {}
    std::string operator+(const StringOperations& other) {
        return str1 + other.str2;
    }
    bool operator==(const StringOperations& other) {
        return str1 == other.str2;
    }
};
int main() {
    std::string input1, input2;

    std::cout << "Enter string 1: ";
    std::cin >> input1;
    std::cout << "Enter string 2: ";
    std::cin >> input2;
    StringOperations stringObj1(input1, input2);
    StringOperations stringObj2(input2, input1);
    std::string concatenatedStr = stringObj1 + stringObj2;
    std::cout << "Concatenated String: " << concatenatedStr << std::endl;
    if (stringObj1 == stringObj2) {
        std::cout << "Strings are equal." << std::endl;
    } else {
        std::cout << "Strings are not equal." << std::endl;
    }
    return 0;
}
```

```
Enter string 1: Kurt
Enter string 2: Cobain
Concatenated String: KurtCobain
Strings are not equal.
```

```
Enter string 1: Rajat
Enter string 2: Rajat
Concatenated String: RajatRajat
Strings are equal.
```

EXPERIMENT-10

AIM: WAP to demonstrate the use of different access specifiers by means of member functions.

THEORY:

In this C++ program, a class named "AccessSpecifiersDemo" is defined to demonstrate different access specifiers (private, protected, and public) using member functions. Access specifiers control the visibility and accessibility of class members.

- Private member functions are accessible only within the class and not from outside.
- Protected member functions are accessible within the class and by derived classes.
- Public member functions are accessible from outside the class.

In the **main** function, an object of the "AccessSpecifiersDemo" class is created, and each member function is called to demonstrate the use of different access specifiers.

ALGORITHM:

1. Start
2. Define a class called "AccessSpecifiersDemo" to demonstrate different access specifiers.
3. Declare private, protected, and public member functions within the class.
4. Inside each member function, perform specific tasks and display messages to indicate the access specifier being used.
5. In the main function:
    - Create an object of the "AccessSpecifiersDemo" class.
    - Call each member function to demonstrate the use of different access specifiers.
6. End.

CODE :

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;

class AccessSpecifiersDemo {
private:
    void privateFunction() {
        cout << "This is a private member function." << endl;
    }
protected:
    void protectedFunction() {
        cout << "This is a protected member function." << endl;
    }
public:
    void publicFunction() {
        cout << "This is a public member function." << endl;
    }
public:
    void demonstrateAccessSpecifiers() {
        // Access private member function within the class
        privateFunction();
        cout << "Inside demonstrateAccessSpecifiers() function." <<
std::endl;
        // Access protected member function within the class
        protectedFunction();
        cout << "Inside demonstrateAccessSpecifiers() function." <<
std::endl;
        // Access public member function within the class
        publicFunction();
```

```
        cout << "Inside demonstrateAccessSpecifiers() function." <<
std::endl;
    }
};
int main() {
    init();
    AccessSpecifiersDemo demo;
    cout << "Demonstrating access specifiers:" << endl;
    // Access public member function from outside the class
    cout << "Calling a public member function using the object of it's
class \n";
    demo.publicFunction();
    // Call a function that demonstrates the use of access specifiers
    demo.demonstrateAccessSpecifiers();
    // Private and protected functions cannot be accessed directly from
outside the class
    // demo.privateFunction(); // Results in an error
    // demo.protectedFunction(); // Results in an error
    return 0;
}
```

OUTPUT :

```cpp
#include <iostream>

class AccessSpecifiersDemo {
private:
    void privateFunction() {
        std::cout << "This is a private member function." << std::endl;
    }
protected:
    void protectedFunction() {
        std::cout << "This is a protected member function." << std::endl;
    }
public:
    void publicFunction() {
        std::cout << "This is a public member function." << std::endl;
    }
public:
    void demonstrateAccessSpecifiers() {
        privateFunction();
        std::cout << "Inside demonstrateAccessSpecifiers() function." << std::endl;
        protectedFunction();
        std::cout << "Inside demonstrateAccessSpecifiers() function." << std::endl;
        publicFunction();
        std::cout << "Inside demonstrateAccessSpecifiers() function." << std::endl;
    }
};

int main() {
    AccessSpecifiersDemo demo;
    std::cout << "Demonstrating access specifiers:" << std::endl;
    demo.publicFunction();
    demo.demonstrateAccessSpecifiers();
    return 0;
}
```

```
Demonstrating access specifiers:
This is a public member function.
This is a private member function.
Inside demonstrateAccessSpecifiers() function.
This is a protected member function.
Inside demonstrateAccessSpecifiers() function.
This is a public member function.
Inside demonstrateAccessSpecifiers() function.
```

EXPERIMENT-11

AIM : WAP to create member function and invoke it using "this" pointer

THEORY:

In C++, the "this" pointer is a special pointer that points to the current instance (object) of a class. It is used to access the members (variables and functions) of the current object.

In this program, a class called "MemberFunctionDemo" is defined with a private data member. The class contains a member function called "displayData," which uses the "this" pointer to access the private data member and display its value. By using "this," you can explicitly refer to the data members of the current object, especially when there is a name conflict between local variables and class members.

In the **main** function, an object of the "MemberFunctionDemo" class is created, and the "displayData" member function is called using the object. This demonstrates the use of the "this" pointer to access the class members.

Algorithm:

1. Start
2. Define a class called "MemberFunctionDemo" with a private data member.
3. Declare a member function "displayData" within the class.
4. Inside the "displayData" function, use the "this" pointer to access the private data member and display its value.
5. In the **main** function:
   - Create an object of the "MemberFunctionDemo" class.
   - Call the "displayData" member function using the object.
6. End.

CODE:

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;
class MemberFunctionDemo {
private:
    int data;
public:
    MemberFunctionDemo(int value) : data(value) {}

    void displayData() {
        // Access the private data member using "this" pointer
        cout << "Value of data: " << this->data << endl;
    }
};
int main() {
    init();
    // Create an object of the MemberFunctionDemo class
    MemberFunctionDemo obj(79);
    cout << "Using 'this' pointer to display data:" << endl;
    // Call the displayData member function using the object
    obj.displayData();

    return 0;
}
```

OUTPUT:

DS.cpp

```cpp
#include <iostream>

class MemberFunctionDemo {
private:
    int data;

public:
    MemberFunctionDemo(int value) : data(value) {}

    void displayData() {
        // Access the private data member using "this" pointer
        std::cout << "Value of data: " << this->data << std::endl;
    }
};

int main() {
    // Create an object of the MemberFunctionDemo class
    MemberFunctionDemo obj(42);

    std::cout << "Using 'this' pointer to display data:" << std::endl;

    // Call the displayData member function using the object
    obj.displayData();

    return 0;
}
```

```
Using 'this' pointer to display data:
Value of data: 42
```

EXPERIMENT-12

AIM : WAP to explain virtual function and calculate area of triangle and rectangle respectively

Theory:

In C++, a virtual function is a member function of a class that is declared with the keyword **virtual** in the base class and is intended to be overridden by derived classes. Virtual functions enable dynamic polymorphism, allowing objects of derived classes to be treated as objects of the base class.

In this program, a base class called "Shape" is defined with a virtual function "calculateArea," which is meant to be overridden by derived classes. Two derived classes, "Triangle" and "Rectangle," inherit from the "Shape" base class and provide their own implementations of the "calculateArea" function to calculate the area of a triangle and rectangle, respectively.

In the **main** function, objects of the "Triangle" and "Rectangle" classes are created, and the "calculateArea" function is called on each object. Since these functions are overridden in the derived classes, the appropriate area calculation is performed for each shape.

Algorithm:

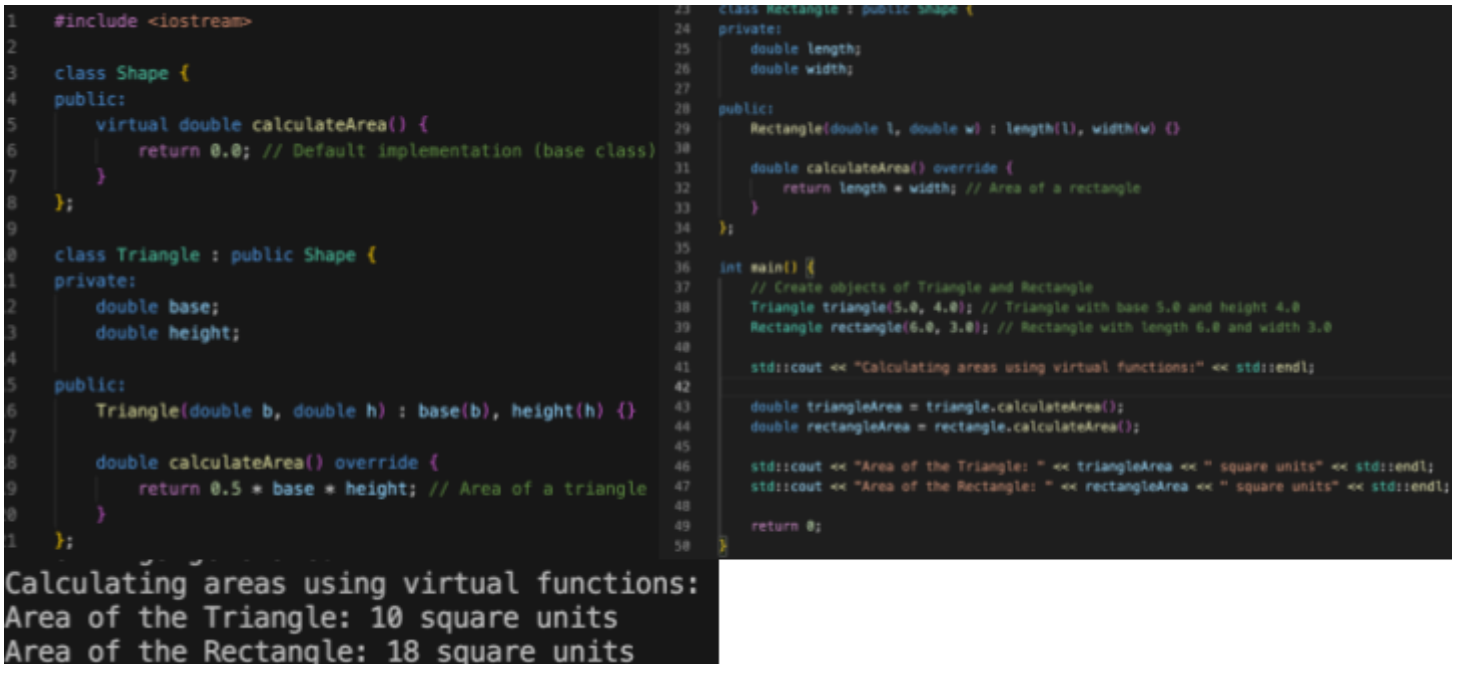
1. Start
2. Define a base class called "Shape" with a virtual function "calculateArea."
3. In the "calculateArea" function, return 0.0 (default implementation).
4. Define two derived classes: "Triangle" and "Rectangle," both inheriting from the "Shape" base class.
5. In the "Triangle" class, override the "calculateArea" function to calculate the area of a triangle.
6. In the "Rectangle" class, override the "calculateArea" function to calculate the area of a rectangle.
7. In the **main** function:
    - Create objects of the "Triangle" and "Rectangle" classes.
    - Call the "calculateArea" function on each object.
    - Display the calculated areas for the triangle and rectangle.
8. End.

CODE:

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;
class Shape {
public:
    virtual double calculateArea() {
        return 0.0; // Default implementation (base class)
    }
};

class Triangle : public Shape {
private:
    double base;
    double height;

public:
    Triangle(double b, double h) : base(b), height(h) {}

    double calculateArea() override {
        return 0.5 * base * height; // Area of a triangle
    }
};
class Rectangle : public Shape {
private:
```

```
    double length;
    double width;

public:
    Rectangle(double l, double w) : length(l), width(w) {}

    double calculateArea() override {
        return length * width; // Area of a rectangle
    }
};

int main() {
    init();
    // Create objects of Triangle and Rectangle
    Triangle triangle(5.0, 4.0); // Triangle with base 5.0 and height 4.0
    Rectangle rectangle(6.0, 3.0); // Rectangle with length 6.0 and width
3.0

    cout << "Calculating areas using virtual functions:" << endl;

    // Call calculateArea function on Triangle and Rectangle objects
    double triangleArea = triangle.calculateArea();
    double rectangleArea = rectangle.calculateArea();

    cout << "Area of the Triangle: " << triangleArea << " square units"
<< endl;
    cout << "Area of the Rectangle: " << rectangleArea << " square units"
<< endl;

    return 0;
}
```

OUTPUT:



EXPERIMENT-13

AIM : WAP to explain Class Template T and member function get_max() return greatest of 2 numbers

Theory:

In C++, class templates allow us to create classes that can work with different data types. In this program, we define a class template called "MaxFinder" that is parameterized by a data type T. The class template has a single member function, "get_max," that is designed to find the greatest of two numbers.

The member function "get_max" takes two numbers of type T and returns the greatest among them. The class template is designed to work with various data types, such as integers, floating-point numbers, and more.

In the main function, we create two objects of the "MaxFinder" class, one for integers and another for floating-point numbers. We then initialize these objects with two numbers each and call the "get_max" member function to determine and display the greatest number.

Algorithm:

1. Start
2. Define a class template called "MaxFinder" with a single member function "get_max."
3. Inside the "MaxFinder" class template, declare two private data members of type **T** to store the two numbers for comparison.
4. Implement the "get_max" member function that returns the greatest of the two numbers.
5. In the **main** function:
    - Create two objects of "MaxFinder" class, one for integers and another for floating-point numbers.
    - Initialize the objects with two numbers each.
    - Call the "get_max" member function on each object and display the greatest number.
6. End.

CODE:

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;
template <typename T>
class MaxFinder {
private:
    T num1;
    T num2;

public:
    MaxFinder(T a, T b) : num1(a), num2(b) {}

    T get_max() {
        return (num1 > num2) ? num1 : num2;
    }
};

int main() {
    init();
    // Create an object for integers
    MaxFinder<int> intMaxFinder(15, 7);
    // Create an object for floating-point numbers
    MaxFinder<double> doubleMaxFinder(3.14, 2.718);
    MaxFinder<string> lexicographicallyLarge("kunal", "abhishek");
```

```
    cout << "Finding the maximum using class templates:" << endl;
    // Find and display the maximum integer
    int maxInt = intMaxFinder.get_max();
    cout << "Maximum Integer: " << maxInt << endl;

    // Find and display the maximum floating-point number
    double maxDouble = doubleMaxFinder.get_max();
    cout << "Maximum Double: " << maxDouble << endl;

    string str = lexicographicallyLarge.get_max();
    cout << "Lexicographically Larger String: " << str << endl;

    return 0;
}
```

```cpp
#include <iostream>

template <typename T>
class MaxFinder {
private:
    T num1;
    T num2;

public:
    MaxFinder(T a, T b) : num1(a), num2(b) {}

    T get_max() {
        return (num1 > num2) ? num1 : num2;
    }
};

int main() {
    MaxFinder<int> intMaxFinder(15, 7);
    MaxFinder<double> doubleMaxFinder(3.14, 2.718);

    std::cout << "Finding the maximum using class templates:" << std::endl;

    int maxInt = intMaxFinder.get_max();
    std::cout << "Maximum Integer: " << maxInt << std::endl;
    double maxDouble = doubleMaxFinder.get_max();
    std::cout << "Maximum Double: " << maxDouble << std::endl;

    return 0;
}
```

```
Finding the maximum using class templates:
Maximum Integer: 15
Maximum Double: 3.14
```

EXPERIMENT-14

AIM : WAP to illustrate

I) Division by Zero
II) Array Index
III) also use multiple catch blocks

Theory:

In C++, exceptions are used to handle runtime errors and unexpected situations that can occur during program execution. The program demonstrates the handling of the following types of exceptions: I) Division by zero: This exception occurs when attempting to divide by zero. II) Array index: This exception occurs when trying to access an element at an invalid index in an array.
Multiple catch blocks are used to handle different types of exceptions. Each catch block specifies the type of exception it can handle and provides custom error messages or error-handling logic.

- The program demonstrates division by zero by attempting to divide by zero and accessing an array element at an invalid index.
- Multiple catch blocks are used to catch and handle exceptions of different types: a. The first catch block catches "runtime_error" exceptions and displays a custom error message. b. The second catch block catches "out_of_range" exceptions and displays a custom error message. c. The last catch block with ellipsis (...) is a catch-all block that handles any other exceptions not explicitly caught and displays a generic error message.

Algorithm:

1. Start
2. Set up a try-catch block to handle exceptions.
3. Inside the try block: a. Perform division by zero to trigger an exception. b. Access an array element that is out of bounds to trigger an exception.
4. Use multiple catch blocks to catch and handle different types of exceptions. a. Catch and handle the "runtime_error" exception for division by zero. b. Catch and handle the "out_of_range" exception for array index.
5. Display appropriate error messages for each caught exception.
6. End.

CODE:

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;
void init()
int main() {

    try {
        cout << "Illustrating division by zero and array index
exceptions:" << endl;
        // Division by zero
        int numerator = 10;
        int denominator = 0;
        if (denominator == 0) {
            throw runtime_error("Division by zero exception.");
        }
```

```
    }
    catch (const runtime_error& e) {
        cerr << "Error: " << e.what() << endl;
    }
    catch (...) {
        cerr << "Unknown error occurred." << endl;
    }
    try{
        // Array index
        vector<int> numbers = {23, 45, 76, 84, 20};
        int index = 8;
        if (index < 0 || index >= numbers.size()) {
            throw out_of_range("Array index out of bounds
exception.");
        }
    }
    catch (const out_of_range& e) {
        cerr << "Error: " << e.what() << endl;
    }
    catch (...) {
        cerr << "Unknown error occurred." << endl;
    }

    return 0;
}
```

OUTPUT :

```
ERROR!
Illustrating division by zero and array index exceptions:
Error: Division by zero exception.
ERROR!
Error: Array index out of bounds exception.
```

EXPERIMENT-15

AIM : WAP to implement function overloading.

Theory:

Function overloading in C++ allows you to define multiple functions with the same name but different parameter lists. This enables you to create functions that perform similar tasks but with different inputs. The compiler determines which function to call based on the number and types of arguments provided.

In this program, a class called "FunctionOverloadingDemo" is defined, and multiple member functions with the same name are declared but with different parameters. Each overloaded function performs a specific task based on the provided arguments and displays a relevant message.

Algorithm:

1. Start
2. Define a class called "FunctionOverloadingDemo."
3. Declare multiple member functions with the same name but different parameters to demonstrate function overloading.
4. Inside each member function, perform a specific task based on the provided arguments and display a relevant message.
5. In the **main** function:
    - Create an object of the "FunctionOverloadingDemo" class.
    - Call the overloaded member functions with different sets of arguments.
6. End.

CODE:

```
//Rajat Sharda
//2K21/EC/176
#include<bits/stdc++.h>
using namespace std;
class FunctionOverloadingDemo {
public:
    // Overloaded function with no parameters
    void displayMessage() {
        cout << "No argument provided. Default message." << endl;
    }
    // Overloaded function with an integer parameter
    void displayMessage(int number) {
        cout << "Integer argument provided: " << number << endl;
    }
    // Overloaded function with two integer parameters
    void displayMessage(int num1, int num2) {
        cout << "Two integer arguments provided: " << num1 << " and " <<
num2 << endl;
        cout << "The Sum of these two integers is: " << num1 + num2 <<
endl;
    }
};
```

```
Demonstrating function overloading:
No argument provided. Default message.
Integer argument provided: 42
Two integer arguments provided: 7 and 11
```