Nithin S
221IT085

IT150 Lab Assignment 3

1. Write a C++ program to add two numbers and demonstrate the working of
i. Pass by value function call
ii. Pass by reference function call

CODE

```cpp
#include <iostream>
int addByValue(int num1, int num2);
void addByReference(int num1, int num2, int &result);

int main()
{
    int number1, number2;

    std::cout << "Enter the first number: ";
    std::cin >> number1;

    std::cout << "Enter the second number: ";
    std::cin >> number2;

    int sumByValue = addByValue(number1, number2);
    std::cout << "Sum (Pass by value): " << sumByValue << std::endl;

    int sumByReference;
    addByReference(number1, number2, sumByReference);
    std::cout << "Sum (Pass by reference): " << sumByReference << std::endl;

    return 0;
}

int addByValue(int num1, int num2)
{
    return num1 + num2;
}

void addByReference(int num1, int num2, int &result)
{
    result = num1 + num2;
}
```

OUTPUT

```
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$ g++ 1.cpp
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$ ./a.out
Enter the first number: 4
Enter the second number: 5
Sum (Pass by value): 9
Sum (Pass by reference): 9
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$
```

2. Write a C++ program to overloaded the functions with different arguments to calculate,
i. the area of a square, rectangle, or circle.
ii. Overloading Using Different Types of Parameters
iii.Overloading using a Different Number of Parameters

CODE

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


double calculateArea(double side);
double calculateArea(double length, double width);
double calculateAreaCircle(double radius);
int calculateArea(int side);

int main() {
    double length, width, radius;
    int side;


    std::cout << "Enter the side length of the square: ";
    std::cin >> side;
    std::cout << "Area of the square: " << calculateArea(side) << std::endl;


    std::cout << "Enter the length of the rectangle: ";
    std::cin >> length;
    std::cout << "Enter the width of the rectangle: ";
    std::cin >> width;
    std::cout << "Area of the rectangle: " << calculateArea(length, width) << std::endl;


    std::cout << "Enter the radius of the circle: ";
    std::cin >> radius;
    std::cout << "Area of the circle: " << calculateArea(radius) << std::endl;

    return 0;
}


double calculateArea(double side) {
    return side * side;
}


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


double calculateAreaCircle(double radius) {
    return M_PI * radius * radius;
}
```

```cpp
double calculateArea(double side) {
    return side * side;
}


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


double calculateAreaCircle(double radius) {
    return M_PI * radius * radius;
}


int calculateArea(int side) {
    return side * side;
}
```

OUTPUT

```
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$ g++ 2.cpp
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$ ./a.out
Enter the side length of the square: 4
Area of the square: 16
Enter the length of the rectangle: 5
Enter the width of the rectangle: 6
Area of the rectangle: 30
Enter the radius of the circle: 5
Area of the circle: 25
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$
```

3.Write a C++ program to create Account class to manage saving account of a bank.
Define methods display_balance() and withdraw() in account class. Also define a friend function display_info() to display basic information of account holder. Use appropriate data types for data members. Use appropriate access specifiers in your program. Make necessary assumptions if required.

CODE

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

class Account;

void display_info(const Account &account);

class Account
{
private:
    std::string accountHolderName;
    long accountNumber;
    double balance;

public:
    Account(const std::string &name, long number, double initialBalance)
        : accountHolderName(name), accountNumber(number), balance(initialBalance) {}

    void display_balance() const
    {
        std::cout << "Account Balance: $" << balance << std::endl;
    }

    void withdraw(double amount)
    {
        if (amount > 0 && amount <= balance)
        {
            balance -= amount;
            std::cout << "Withdrawal successful. Remaining balance: $" << balance << std::endl;
        }
        else
        {
            std::cout << "Invalid withdrawal amount or insufficient balance." << std::endl;
        }
    }

    friend void display_info(const Account &account);
};

void display_info(const Account &account)
{
    std::cout << "Account Holder Name: " << account.accountHolderName << std::endl;
    std::cout << "Account Number: " << account.accountNumber << std::endl;
    account.display_balance();
}
```

```cpp
int main()
{

    std::string name;
    long number;
    double initialBalance;

    std::cout << "Enter Account Holder Name: ";
    std::getline(std::cin, name);

    std::cout << "Enter Account Number: ";
    std::cin >> number;

    std::cout << "Enter Initial Balance: $";
    std::cin >> initialBalance;

    Account myAccount(name, number, initialBalance);

    display_info(myAccount);

    double withdrawalAmount;
    std::cout << "Enter Withdrawal Amount: $";
    std::cin >> withdrawalAmount;

    myAccount.withdraw(withdrawalAmount);

    display_info(myAccount);

    return 0;
}
```

OUTPUT

```
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$ g++ 3.cpp
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$ ./a.out
Enter Account Holder Name: Nithin
Enter Account Number: 172933
Enter Initial Balance: $89
Account Holder Name: Nithin
Account Number: 172933
Account Balance: $89
Enter Withdrawal Amount: $9
Withdrawal successful. Remaining balance: $80
Account Holder Name: Nithin
Account Number: 172933
Account Balance: $80
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$
```

4. Write a C++ program to create a friend function for adding two matrices using two different classes.

CODE

```cpp
#include <iostream>

class Matrix;

void addMatrices(const Matrix &mat1, const Matrix &mat2);

class Matrix
{
private:
    int rows, cols;
    int **data;

public:
    Matrix(int rows, int cols) : rows(rows), cols(cols)
    {

        data = new int *[rows];
        for (int i = 0; i < rows; ++i)
        {
            data[i] = new int[cols];
        }
    }

    void inputMatrix()
    {
        std::cout << "Enter matrix elements:" << std::endl;
        for (int i = 0; i < rows; ++i)
        {
            for (int j = 0; j < cols; ++j)
            {
                std::cout << "Enter element at position (" << i + 1 << ", " << j + 1 << "): ";
                std::cin >> data[i][j];
            }
        }
    }

    void displayMatrix() const
    {
        std::cout << "Matrix:" << std::endl;
        for (int i = 0; i < rows; ++i)
        {
            for (int j = 0; j < cols; ++j)
            {
                std::cout << data[i][j] << " ";
            }
            std::cout << std::endl;
        }
    }
```

```
            std::cout << std::endl;
        }
    }

    friend void addMatrices(const Matrix &mat1, const Matrix &mat2);
};

void addMatrices(const Matrix &mat1, const Matrix &mat2)
{
    if (mat1.rows != mat2.rows || mat1.cols != mat2.cols)
    {
        std::cout << "Matrix addition is not possible. Matrices must have the same dimensions." << std
        return;
    }

    Matrix result(mat1.rows, mat1.cols);

    for (int i = 0; i < mat1.rows; ++i)
    {
        for (int j = 0; j < mat1.cols; ++j)
        {
            result.data[i][j] = mat1.data[i][j] + mat2.data[i][j];
        }
    }

    std::cout << "Resultant Matrix after addition:" << std::endl;
    result.displayMatrix();
}

int main()
{
    int rows, cols;

    std::cout << "Enter the number of rows for matrices: ";
    std::cin >> rows;

    std::cout << "Enter the number of columns for matrices: ";
    std::cin >> cols;
```

```cpp
int main()
{
    int rows, cols;

    std::cout << "Enter the number of rows for matrices: ";
    std::cin >> rows;

    std::cout << "Enter the number of columns for matrices: ";
    std::cin >> cols;

    Matrix matrix1(rows, cols);
    Matrix matrix2(rows, cols);

    std::cout << "For Matrix 1:" << std::endl;
    matrix1.inputMatrix();

    std::cout << "For Matrix 2:" << std::endl;
    matrix2.inputMatrix();

    std::cout << "Entered matrices:" << std::endl;
    matrix1.displayMatrix();
    matrix2.displayMatrix();

    addMatrices(matrix1, matrix2);

    return 0;
}
```

OUTPUT

```
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$ g++ 4.cpp
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$ ./a.out
Enter the number of rows for matrices: 3
Enter the number of columns for matrices: 3
For Matrix 1:
Enter matrix elements:
Enter element at position (1, 1): 1
Enter element at position (1, 2): 2
Enter element at position (1, 3): 3
Enter element at position (2, 1): 4
Enter element at position (2, 2): 5
Enter element at position (2, 3): 6
Enter element at position (3, 1): 7
Enter element at position (3, 2): 8
Enter element at position (3, 3): 9
For Matrix 2:
Enter matrix elements:
Enter element at position (1, 1): 1
Enter element at position (1, 2): 2
Enter element at position (1, 3): 3
Enter element at position (2, 1): 4
Enter element at position (2, 2): 5
Enter element at position (2, 3): 6
Enter element at position (3, 1): 7
Enter element at position (3, 2): 8
Enter element at position (3, 3): 9
Entered matrices:
Matrix:
1 2 3
4 5 6
7 8 9
Matrix:
1 2 3
4 5 6
7 8 9
Resultant Matrix after addition:
Matrix:
2 4 6
8 10 12
14 16 18
nithin@nithin1729s:~/Codes/Sem4/IT150/Lab/Lab_3$
```