



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Winter Domain Camp Day 1

Student Name: Divya Sharma

UID: 22BCS14192

Branch: BE-CSE

Section/Group: 22BCS_IOT_603-B

Semester: 5th

Date of Performance: 19 December, 2024

Problem 1 (Very Easy): Sum of Natural Numbers up to N

Calculate the sum of all natural numbers from 1 to n, where n is a positive integer. Use the formula:

$$\text{Sum} = n \times (n+1) / 2.$$

Take n as input and output the sum of natural numbers from 1 to n .

Solution:

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

int main()
{
    cout<< "Sum upto:";
    int n;
    cin>> n;
    cout<< "Sum of " << n << " Natural Numbers: ";
    int sum = n * (n + 1) / 2;
    cout<< sum;
    return 0;
}
```

Output:

```
Sum upto:9
Sum of 9 Natural Numbers: 45

...Program finished with exit code 0
Press ENTER to exit console.
```

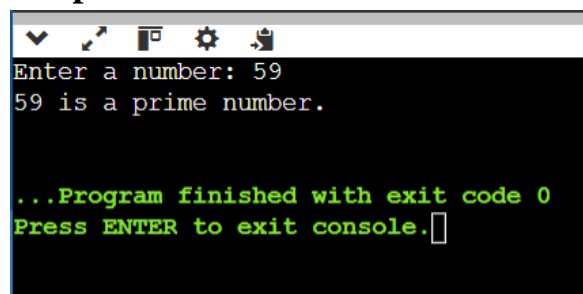
Problem 2(VeryEasy): Check if a Number is Prime.

Check if a given number n is a prime number. A prime number is a natural number greater than 1 that has no positive divisors other than 1 and itself.

Solution:

```
#include <iostream>
using namespace std;
bool isPrime(int n) {
    if (n <= 1) {
        return false;
    }
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}
int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;
    if (isPrime(n)) {
        cout << n << " is a prime number." << endl;
    } else {
        cout << n << " is not a prime number." << endl;
    }
    return 0;
}
```

Output:



```
Enter a number: 59
59 is a prime number.

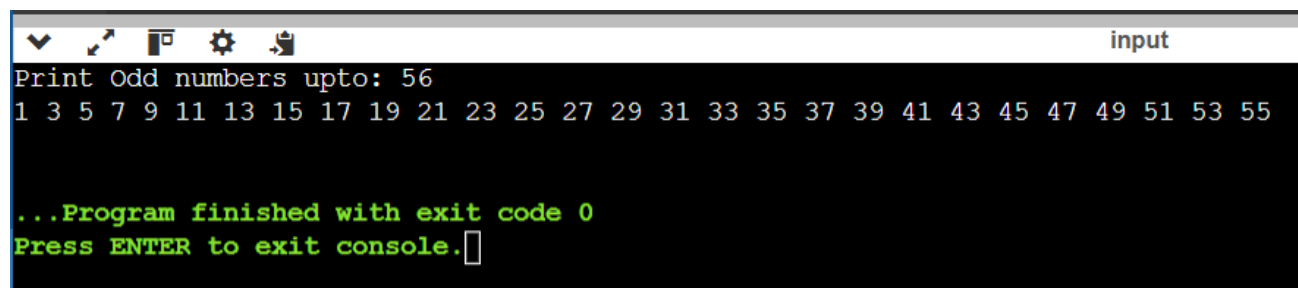
...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 3 (Very Easy): Print Odd Numbers up to N.

Print all odd numbers between 1 and n, inclusive. Odd numbers are integers that are not divisible by 2. These numbers should be printed in ascending order, separated by spaces.

Solution:

```
#include <iostream>
using namespace std;
void printOddNumbers(int n) {
    for (int i = 1; i <= n; i += 2) {
        cout << i << " ";
    }
    cout << endl;
}
int main() {
    int n;
    cout << "Print Odd numbers upto: ";
    cin >> n;
    if (n < 1) {
        cout << "Invalid input! n should be greater than or equal to 1." << endl;
        return 1;
    }
    printOddNumbers(n);
    return 0;
}
```

Output:

```
Print Odd numbers upto: 56
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51 53 55

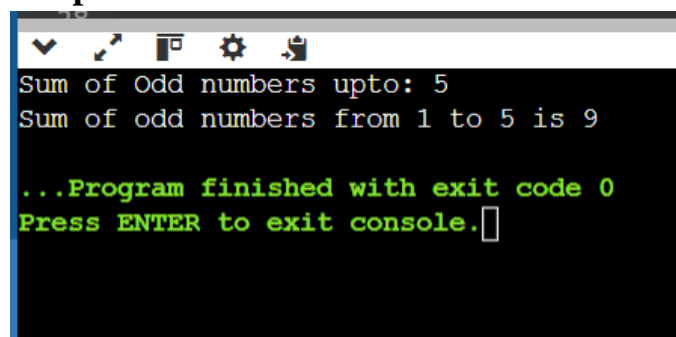
...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 4(Easy): Sum of Odd Numbers up to N.

Calculate the sum of all odd numbers from 1 to n. An odd number is an integer that is not divisible by 2. The sum of odd numbers, iterate through all the numbers from 1 to n, check if each number is odd, and accumulate the sum.

Solution:

```
#include <iostream>
using namespace std;
void SumOddNumbers(int n)
{
    int sum = 0;
    for (int i = 1; i <= n; i += 2)
    {
        sum = sum + i;
    }
    cout << "Sum of odd numbers from 1 to " << n << " is " << sum;
}
int main() {
    int n;
    cout << "Sum of Odd numbers upto: ";
    cin >> n;
    if (n < 1) {
        cout << "Invalid input! n should be greater than or equal to 1." << endl;
        return 1;
    }
    SumOddNumbers(n);
    return 0;
}
```

Output:

```
C++
Sum of Odd numbers upto: 5
Sum of odd numbers from 1 to 5 is 9
...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 5(Easy): Print Multiplication Table of a Number.

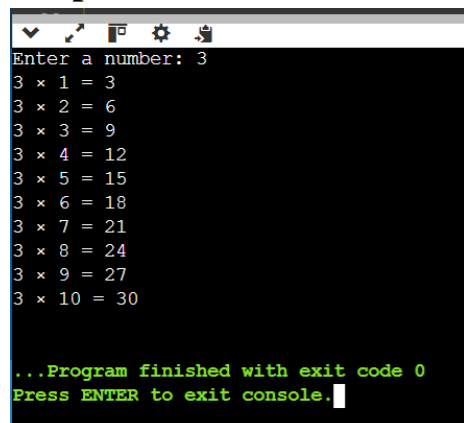
Print the multiplication table of a given number n. A multiplication table for a number n is a list of products of n with integers from 1 to 10. For example, the multiplication table for 3 is:

$$3 \times 1 = 3, 3 \times 2 = 6, \dots, 3 \times 10 = 30$$

Solution:

```
#include <iostream>
using namespace std;
void multiplicationTable(int n) {
    for (int i = 1; i <= 10; i++) {
        cout << n << " x " << i << " = " << n * i << endl;
    }
}
int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;
    multiplicationTable(n);
    return 0;
}
```

Output:



```
Enter a number: 3
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
3 x 10 = 30

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 6(Easy): Count Digits in a Number

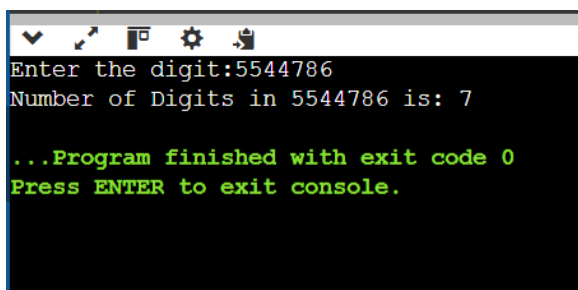
Count the total number of digits in a given number n . The number can be a positive integer. For example, for the number 12345, the count of digits is 5. For a number like 900000, the count of digits is 6.

Given an integer n , your task is determining how many digits are present in n . This task will help you practice working with loops, number manipulation, and conditional logic.

Solution:

```
#include<iostream>
using namespace std;
int Count(int n)
{
    if (n == 0) {
        return 1;
    }
    int count = 0;
    while (n != 0) {
        n /= 10;
        count++;
    }
    return count;
}
int main()
{
    cout<< "Enter the digit:";
    int n;
    cin>> n;
    cout<< "Number of Digits in " << n << " is: ";
    cout<< Count(n);
    return 0;
}
```

Output:



```
Enter the digit:5544786
Number of Digits in 5544786 is: 7

...Program finished with exit code 0
Press ENTER to exit console.
```

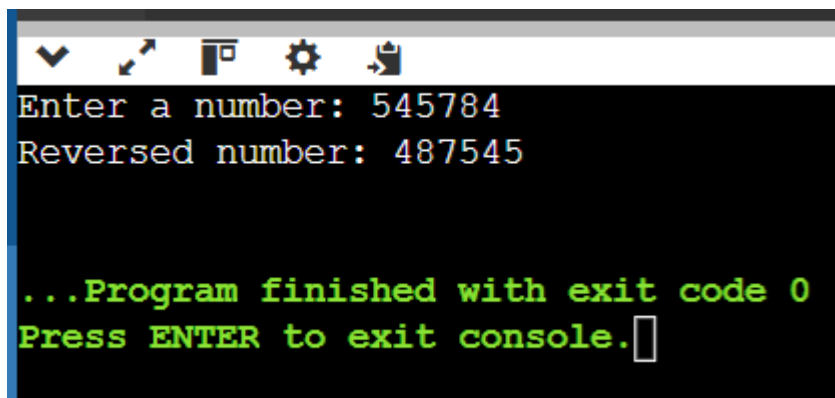
Problem 7(Easy): Reverse a Number.

Reverse the digits of a given number n. For example, if the input number is 12345, the output should be 54321. The task involves using loops and modulus operators to extract the digits and construct the reversed number.

Solution:

```
#include <iostream>
using namespace std;
int reverseNumber(int n) {
    int reversed = 0;
    while (n != 0) {
        int digit = n % 10;
        reversed = reversed * 10 + digit;
        n = n / 10;
    }
    return reversed;
}
int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;
    int reversedNumber = reverseNumber(n);
    cout << "Reversed number: " << reversedNumber << endl;
    return 0;
}
```

Output:



```
Enter a number: 545784
Reversed number: 487545

...Program finished with exit code 0
Press ENTER to exit console.
```

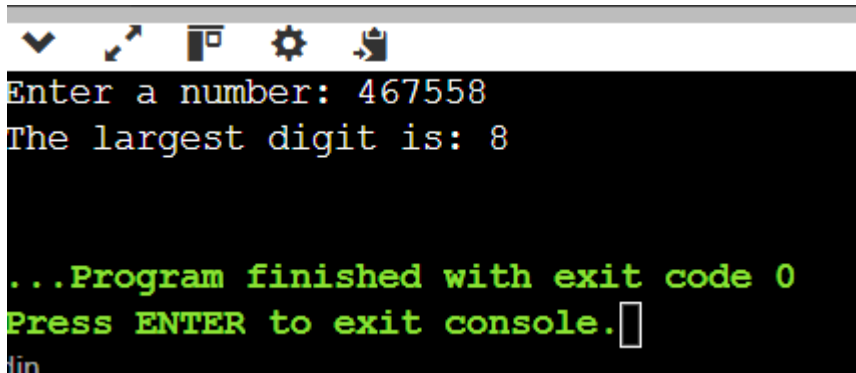
Problem 8(Easy): Find the Largest Digit in a Number

Find the largest digit in a given number n. For example, for the number 2734, the largest digit is 7. You need to extract each digit from the number and determine the largest one. The task will involve using loops and modulus operations to isolate the digits.

Solution:

```
#include <iostream>
using namespace std;
int largestDigit(int n) {
    int largest = 0;
    while (n != 0) {
        int digit = n % 10;
        if (digit > largest) {
            largest = digit;
        }
        n = n / 10;
    }
    return largest;
}
int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;
    int largest = largestDigit(n);
    cout << "The largest digit is: " << largest << endl;
    return 0;
}
```

Output:



```
Enter a number: 467558
The largest digit is: 8

...Program finished with exit code 0
Press ENTER to exit console.
```

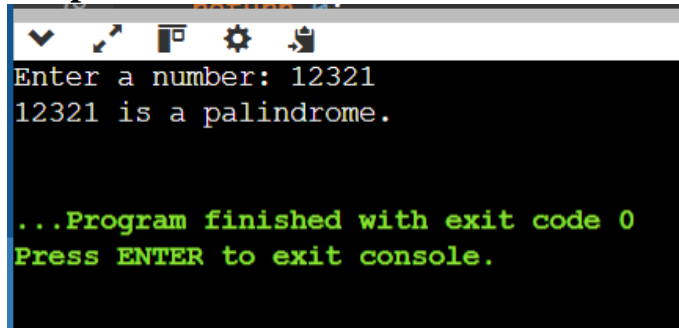

Problem 9(Easy): Check if a Number is a Palindrome

Check whether a given number is a palindrome or not. A number is called a palindrome if it reads the same backward as forward. For example, 121 is a palindrome because reading it from left to right is the same as reading it from right to left. Similarly, 12321 is also a palindrome, but 12345 is not.

Solution:

```
#include <iostream>
using namespace std;
bool isPalindrome(int n) {
    int original = n;
    int reversed = 0;
    while (n != 0) {
        int digit = n % 10;
        reversed = reversed * 10 + digit;
        n = n / 10;
    }
    return original == reversed;
}
int main() {
    int n;
    cout<< "Enter a number: ";
    cin>> n;
    if (isPalindrome(n)) {
        cout<< n << " is a palindrome." <<endl;
    } else {
        cout<< n << " is not a palindrome." <<endl;
    }
    return 0;
}
```

Output:



```
Enter a number: 12321
12321 is a palindrome.

...Program finished with exit code 0
Press ENTER to exit console.
```

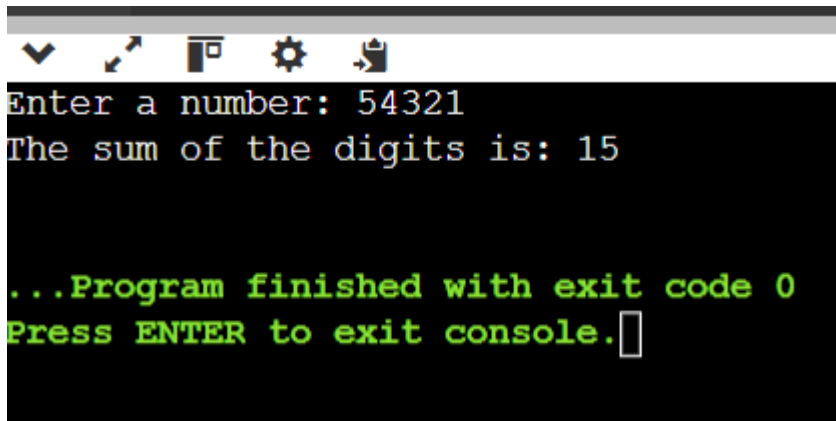
Problem 10(Easy): Find the Sum of Digits of a Number

Calculate the sum of the digits of a given number n. For example, for the number 12345, the sum of the digits is $1+2+3+4+5=15$. To solve this, you will need to extract each digit from the number and calculate the total sum.

Solution:

```
#include <iostream>
using namespace std;
int sumOfDigits(int n) {
    int sum = 0;
    while (n != 0) {
        sum += n % 10;
        n = n / 10;
    }
    return sum;
}
int main() {
    int n;
    cout << "Enter a number: ";
    cin >> n;
    int sum = sumOfDigits(n);
    cout << "The sum of the digits is: " << sum << endl;
    return 0;
}
```

Output:



```
Enter a number: 54321
The sum of the digits is: 15

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 11 (Moderate): Function Overloading for Calculating Area.

Write a program to calculate the area of different shapes using function overloading. Implement overloaded functions to compute the area of a circle, a rectangle, and a triangle.

Solution:

```
#include <iostream>
#include <cmath>
using namespace std;
const double PI = 3.14159;
double area(double radius) {
    return PI * radius * radius;
}

double area(double length, double breadth) {
    return length * breadth;
}

double area(double base, double height, double n ) {
    return n * base * height;
}

int main() {
    double radius, length, breadth, base, height;

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

    cout<< "Enter length and breadth of the rectangle: ";
    cin>> length >> breadth;
    cout<< "Area of the rectangle: " << area(length, breadth) << endl;

    cout<< "Enter base and height of the triangle: ";
    cin>> base >> height;
    double n = 0.5;
    cout<< "Area of the triangle: " << area(base, height, n) << endl;

    return 0;
}
```

Output:

```
Enter radius of the circle: 7
Area of the circle: 153.938
Enter length and breadth of the rectangle: 10
4
Area of the rectangle: 40
Enter base and height of the triangle: 12
45
Area of the triangle: 270

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 12(Moderate): Function Overloading with Hierarchical Structure.

Write a program that demonstrates function overloading to calculate the salary of employees at different levels in a company hierarchy. Implement overloaded functions to compute salary for:

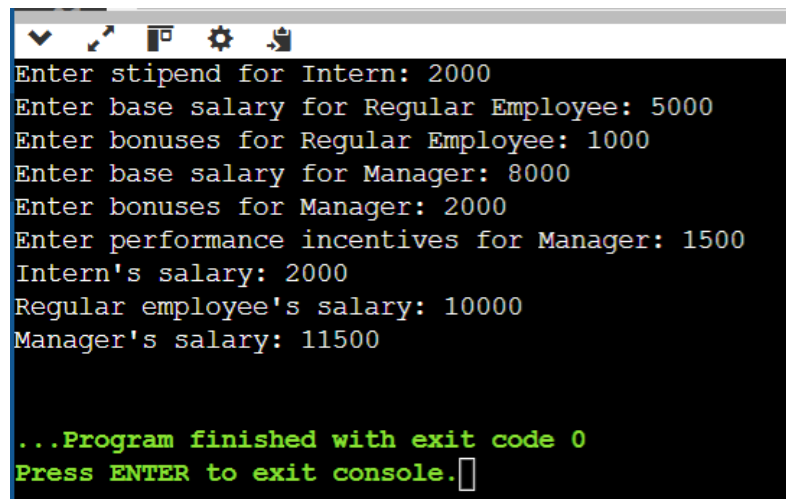
- Intern (basic stipend).
- Regular employee (base salary + bonuses).
- Manager (base salary + bonuses + performance incentives).

Solution:

```
#include <iostream>
using namespace std;
class Employee {
public:
    double calculateSalary(double stipend) {
        return stipend;
    }
    double calculateSalary(double baseSalary, double bonuses) {
        return baseSalary + bonuses;
    }
    double calculateSalary(double baseSalary, double bonuses, double
performanceIncentives) {
        return baseSalary + bonuses + performanceIncentives;
    }
};
int main() {
    Employee emp;
    double stipend, baseSalary, bonuses, performanceIncentives;
    cout<< "Enter stipend for Intern: ";
    cin>> stipend;
    cout<< "Enter base salary for Regular Employee: ";
    cin>> baseSalary;
```

```
cout<< "Enter bonuses for Regular Employee: ";
cin>> bonuses;
cout<< "Enter base salary for Manager: ";
cin>> baseSalary;
cout<< "Enter bonuses for Manager: ";
cin>> bonuses;
cout<< "Enter performance incentives for Manager: ";
cin>> performanceIncentives;
cout<< "Intern's salary: " << emp.calculateSalary(stipend) << endl;
cout<< "Regular employee's salary: " << emp.calculateSalary(baseSalary,
bonuses) << endl;
cout<< "Manager's salary: " << emp.calculateSalary(baseSalary, bonuses,
performanceIncentives) << endl;
return 0;
}
```

Output:



```
Enter stipend for Intern: 2000
Enter base salary for Regular Employee: 5000
Enter bonuses for Regular Employee: 1000
Enter base salary for Manager: 8000
Enter bonuses for Manager: 2000
Enter performance incentives for Manager: 1500
Intern's salary: 2000
Regular employee's salary: 10000
Manager's salary: 11500

...Program finished with exit code 0
Press ENTER to exit console.
```

Problem 13(Moderate)

Create a C++ program that demonstrates function overloading to calculate the area of different geometric shapes. Implement three overloaded functions named calculateArea that compute the area for the following shapes:

Circle: Accepts the radius.

Rectangle: Accepts the length and breadth.

Triangle: Accepts the base and height.

Additionally, use a menu-driven program to let the user choose the type of shape and input the respective parameters. Perform necessary validations on the input values.

Input Format

- An integer $1 \leq \text{choice} \leq 3$ representing the shape type:

1. Circle
2. Rectangle
3. Triangle

- For each shape:

- Circle: A positive floating-point number for the radius.
- Rectangle: Two positive floating-point numbers for length and breadth.
- Triangle: Two positive floating-point numbers for base and height.

Constraints

- $1 \leq \text{choice} \leq 3$.
- $0.0 < \text{parameters} \leq 10^6$.

Code:

```
#include <iostream>
#include <iomanip>
#include <limits>
#include <cmath>
```

```
// Function prototypes for overloaded functions to calculate area
double calculateArea(double radius);           // Circle
double calculateArea(double length, double breadth); // Rectangle
double calculateArea(double base, double height); // Triangle
```

```
// Input validation function
```

```
template <typename T>
T getPositiveInput(const std::string &prompt) {
    T value;
    while (true) {
        std::cout << prompt;
        std::cin >> value;

        if (std::cin.fail() || value <= 0.0 || value > 1e6) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cout << "Invalid input. Please enter a positive number (0 < value <= 1,000,000).\n";
        } else {
            break;
        }
    }
    return value;
}
```

```
}

int main() {
    int choice;

    std::cout<< "Choose a shape to calculate the area:\n";
    std::cout<< "1. Circle\n";
    std::cout<< "2. Rectangle\n";
    std::cout<< "3. Triangle\n";

    while (true) {
        std::cout<< "Enter your choice (1-3): ";
        std::cin>> choice;

        if (std::cin.fail() || choice < 1 || choice > 3) {
            std::cin.clear();
            std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            std::cout<< "Invalid choice. Please enter a number between 1 and 3.\n";
        } else {
            break;
        }
    }

    double area;
    switch (choice) {
        case 1: {
            double radius = getPositiveInput<double>("Enter the radius of the circle: ");
            area = calculateArea(radius);
            std::cout<< "The area of the circle is: " <<std::fixed <<std::setprecision(2) <<
            area << "\n";
            break;
        }
        case 2: {
            double length = getPositiveInput<double>("Enter the length of the rectangle: ");
            double breadth = getPositiveInput<double>("Enter the breadth of the rectangle: ");
            area = calculateArea(length, breadth);
            std::cout<< "The area of the rectangle is: " <<std::fixed <<std::setprecision(2) <<
            area << "\n";
            break;
        }
    }
}
```

```
    }  
    case 3: {  
        double base = getPositiveInput<double>("Enter the base of the triangle:  
");  
        double height = getPositiveInput<double>("Enter the height of the  
triangle: ");  
        area = calculateArea(base, height);  
        std::cout<< "The area of the triangle is: " <<std::fixed <<std::setprecision(2) <<  
area << "\n";  
        break;  
    }  
}  
  
return 0;  
}  
  
// Overloaded function definitions  
double calculateArea(double radius) {  
    return M_PI * radius * radius; // Area of circle  
}  
  
double calculateArea(double length, double breadth) {  
    return length * breadth; // Area of rectangle  
}  
  
double calculateArea(double base, double height) {  
    return 0.5 * base * height; // Area of triangle  
}
```

Output:

Output

```
Choose a shape to calculate the area:  
1. Circle  
2. Rectangle  
3. Triangle  
Enter your choice (1-3): 1  
Enter the radius of the circle: 5  
The area of the circle is: 78.54
```

=== Code Execution Successful ===

Problem 14(Moderate): Create a C++ program using multiple inheritance to simulate a library system. Design two base classes:

- Book to store book details (title, author, and ISBN).
- Borrower to store borrower details (name, ID, and borrowed book).

Create a derived class Library that inherits from both Book and Borrower. Use this class to track the borrowing and returning of books.

Code:

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

```
// Base class to store book details
```

```
class Book {
```

```
protected:
```

```
    string title;
    string author;
    string ISBN;
```

```
public:
```

```
    void setBookDetails(const string &bookTitle, const string &bookAuthor, const
string &bookISBN) {
        title = bookTitle;
        author = bookAuthor;
        ISBN = bookISBN;
    }
```

```
    void displayBookDetails() const {
        cout<< "Book Details:\n";
        cout<< "Title: " << title <<endl;
        cout<< "Author: " << author <<endl;
        cout<< "ISBN: " << ISBN <<endl;
    }
};
```

```
// Base class to store borrower details
```

```
class Borrower {
```

```
protected:
```

```
    string name;
    string ID;
    string borrowedBook;
```

```
public:
    void setBorrowerDetails(const string &borrowerName, const string
&borrowerID) {
        name = borrowerName;
        ID = borrowerID;
    }

    void borrowBook(const string &bookTitle) {
borrowedBook = bookTitle;
    }

    void returnBook() {
borrowedBook = "";
    }

    void displayBorrowerDetails() const {
cout<< "Borrower Details:\n";
cout<< "Name: " << name <<endl;
cout<< "ID: " << ID <<endl;
        if (!borrowedBook.empty()) {
cout<< "Borrowed Book: " <<borrowedBook<<endl;
        } else {
cout<< "No book currently borrowed." <<endl;
        }
    }
};

// Derived class to simulate the library system
class Library : public Book, public Borrower {
public:
    void borrowBookFromLibrary() {
        if (borrowedBook.empty()) {
borrowBook(title);
cout<< name << " has borrowed the book: " << title <<endl;
        } else {
cout<< name << " already has a borrowed book: " <<borrowedBook<< ". Return
it before borrowing a new one." <<endl;
        }
    }

    void returnBookToLibrary() {
        if (!borrowedBook.empty()) {
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
cout<< name << " has returned the book: " <<borrowedBook<<endl;
returnBook();
    } else {
cout<< "No book to return." <<endl;
    }
}
};
```

```
intmain() {
    Library library;

    // Setting book details
library.setBookDetails("The Great Gatsby", "F. Scott Fitzgerald",
"9780743273565");
```

```
    // Setting borrower details
library.setBorrowerDetails("Alice", "B001");
```

```
    // Display initial details
library.displayBookDetails();
library.displayBorrowerDetails();
```

```
    // Borrowing a book
library.borrowBookFromLibrary();
library.displayBorrowerDetails();
```

```
    // Returning a book
library.returnBookToLibrary();
library.displayBorrowerDetails();
```

```
    return 0;
}
```

Output:

```
Book Details:
Title: The Great Gatsby
Author: F. Scott Fitzgerald
ISBN: 9780743273565
Borrower Details:
Name: Alice
ID: B001
No book currently borrowed.
Alice has borrowed the book: The Great Gatsby
Borrower Details:
Name: Alice
ID: B001
Borrowed Book: The Great Gatsby
Alice has returned the book: The Great Gatsby
Borrower Details:
Name: Alice
ID: B001
No book currently borrowed.
```

Problem 15(Moderate): Implement matrix operations in C++ using function overloading. Write a function `operate()` that can perform:

- Matrix Addition for matrices of the same dimensions.
- Matrix Multiplication where the number of columns of the first matrix equals the number of rows of the second matrix.

Code:

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

```
class Matrix {
private:
    vector<vector<int>>> data;
    int rows, cols;
```

```
public:
```

```
    // Constructor to initialize matrix with given dimensions
    Matrix(int r, int c) : rows(r), cols(c) {
        data.resize(rows, vector<int>(cols));
    }
```

```
    // Function to set matrix elements
```

```
void setElements() {
    cout<< "Enter elements for a " << rows << "x" << cols << " matrix:\n";
    for (inti = 0; i< rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            cin>> data[i][j];
        }
    }
}

// Function to display matrix
void display() const {
    for (const auto &row : data) {
        for (const auto &elem : row) {
            cout<<elem<< " ";
        }
        cout<<endl;
    }
}

// Function to get the number of rows
int getRows() const { return rows; }

// Function to get the number of columns
int getCols() const { return cols; }

// Overloaded addition operator for matrix addition
Matrix operator+(const Matrix &other) {
    if (rows != other.rows || cols != other.cols) {
        throw invalid_argument("Matrix dimensions must match for addition.");
    }
    Matrix result(rows, cols);
    for (inti = 0; i< rows; ++i) {
        for (int j = 0; j < cols; ++j) {
            result.data[i][j] = data[i][j] + other.data[i][j];
        }
    }
    return result;
}

// Overloaded multiplication operator for matrix multiplication
Matrix operator*(const Matrix &other) {
    if (cols != other.rows) {
```

```
        throw invalid_argument("Number of columns of the first matrix must  
        equal the number of rows of the second matrix.");  
    }  
    Matrix result(rows, other.cols);  
    for (inti = 0; i < rows; ++i) {  
        for (int j = 0; j < other.cols; ++j) {  
result.data[i][j] = 0;  
            for (int k = 0; k < cols; ++k) {  
result.data[i][j] += data[i][k] * other.data[k][j];  
            }  
        }  
    }  
    return result;  
}  
};
```

```
int main() {  
    try {  
int r1, c1, r2, c2;
```

```
        // Input for the first matrix  
cout << "Enter the number of rows and columns for the first matrix: ";  
cin >> r1 >> c1;  
        Matrix mat1(r1, c1);  
        mat1.setElements();  
  
        // Input for the second matrix  
cout << "Enter the number of rows and columns for the second matrix: ";  
cin >> r2 >> c2;  
        Matrix mat2(r2, c2);  
        mat2.setElements();  
  
        // Perform matrix addition if dimensions match  
        if (r1 == r2 && c1 == c2) {  
cout << "Matrix Addition Result:\n";  
            Matrix sum = mat1 + mat2;  
sum.display();  
        } else {  
cout << "Matrix addition skipped (dimensions do not match).\n";  
        }  
  
        // Perform matrix multiplication if valid
```

```
        if (c1 == r2) {  
            cout<< "Matrix Multiplication Result:\n";  
            Matrix product = mat1 * mat2;  
            product.display();  
        } else {  
            cout<< "Matrix multiplication skipped (invalid dimensions).\n";  
        }  
    } catch (const exception &e) {  
        cerr<< "Error: " <<e.what() <<endl;  
    }  
  
    return 0;  
}
```

Output:

```
Enter the number of rows and columns for the first matrix: 3 4  
Enter elements for a 3x4 matrix:  
1  
23  
4
```

Problem 16 (Hard):

Create a program that demonstrates inheritance by defining:

- A base class Student to store details like Roll Number and Name.
- A derived class Result to store marks for three subjects and calculate the total and percentage.

Code:

```
#include <iostream>  
#include <string>  
using namespace std;  
  
// Base class to store student details  
class Student {  
protected:  
    int rollNumber;  
    string name;  
  
public:  
    void setDetails(int r, const string &n) {  
        rollNumber = r;  
        name = n;  
    }  
}
```

```
void displayDetails() const {
    cout<< "Student Details:\n";
    cout<< "Roll Number: " << rollNumber<<endl;
    cout<< "Name: " << name <<endl;
}

};

// Derived class to store marks and calculate total and percentage
class Result : public Student {
private:
    float marks[3];

public:
    void setMarks(float m1, float m2, float m3) {
        marks[0] = m1;
        marks[1] = m2;
        marks[2] = m3;
    }

    void calculateAndDisplayResult() const {
        float total = marks[0] + marks[1] + marks[2];
        float percentage = total / 3;

        cout<< "Result:\n";
        cout<< "Total Marks: " << total <<endl;
        cout<< "Percentage: " << percentage << "%" <<endl;
    }
};

int main() {
    Result student;

    // Setting student details
    int rollNumber;
    string name;
    cout<< "Enter Roll Number: ";
    cin>>rollNumber;
    cin.ignore(); // To clear the input buffer
    cout<< "Enter Name: ";
    getline(cin, name);
    student.setDetails(rollNumber, name);
```



```
// Setting marks
float marks[3];
cout<< "Enter marks for three subjects:\n";
for (inti = 0; i< 3; ++i) {
cout<< "Subject " << (i + 1) << ": ";
cin>> marks[i];
}
student.setMarks(marks[0], marks[1], marks[2]);

// Displaying details and result
cout<<endl;
student.displayDetails();
student.calculateAndDisplayResult();

return 0;
}
```

Output:

```
Enter Roll Number: 3
Enter Name: harry
Enter marks for three subjects:
Subject 1: e
Subject 2: Subject 3:
Student Details:
Roll Number: 3
Name: harry
Result:
Total Marks: 0
Percentage: 0%
```

Problem 17(Hard): Write a program that demonstrates encapsulation by creating a class Employee. The class should have private attributes to store: Employee ID.

Employee Name.

Employee Salary.

Provide public methods to set and get these attributes, and a method to display all details of the employee.

Code:

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

```
// Class to demonstrate encapsulation
class Employee {
private:
    int employeeID;
    string employeeName;
    double employeeSalary;

public:
    // Method to set employee details
    void setDetails(int id, const string &name, double salary) {
        if (id > 0 && salary > 0) {
            employeeID = id;
            employeeName = name;
            employeeSalary = salary;
        } else {
            cout<< "Invalid ID or salary. Both must be positive." <<endl;
        }
    }

    // Method to get employee ID
    int getEmployeeID() const {
        return employeeID;
    }

    // Method to get employee name
    string getEmployeeName() const {
        return employeeName;
    }

    // Method to get employee salary
    double getEmployeeSalary() const {
        return employeeSalary;
    }

    // Method to display all employee details
    void displayDetails() const {
        cout<< "Employee Details:\n";
        cout<< "ID: " <<employeeID<<endl;
        cout<< "Name: " <<employeeName<<endl;
        cout<< "Salary: " <<employeeSalary<<endl;
    }
}
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
};

int main() {
    Employee emp;

    // Input employee details
    int id;
    string name;
    double salary;

    cout<< "Enter Employee ID: ";
    cin>> id;
    cin.ignore(); // Clear input buffer
    cout<< "Enter Employee Name: ";
    getline(cin, name);
    cout<< "Enter Employee Salary: ";
    cin>> salary;

    // Set details and display
    emp.setDetails(id, name, salary);

    cout<< "\nFetching Employee Details:\n";
    emp.displayDetails();

    return 0;
}
```

Output:

```
Enter Employee ID: 3
Enter Employee Name: isga
Enter Employee Salary: 1000000

Fetching Employee Details:
Employee Details:
ID: 3
Name: isga
Salary: 1e+06

=== Code Execution Successful ===
```

Problem 18(Hard): Write a program that demonstrates function overloading to calculate the salary of employees at different levels in a company hierarchy.

Implement overloaded functions to compute salary for:

- Intern (basic stipend).
- Regular employee (base salary + bonuses).
- Manager (base salary + bonuses + performance incentives).

Code:

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

class SalaryCalculator {
public:
    // Function to calculate salary for Intern
    double calculateSalary(double stipend) {
        return stipend;
    }

    // Function to calculate salary for Regular Employee
    double calculateSalary(double baseSalary, double bonuses) {
        return baseSalary + bonuses;
    }

    // Function to calculate salary for Manager
    double calculateSalary(double baseSalary, double bonuses, double
performanceIncentives) {
        return baseSalary + bonuses + performanceIncentives;
    }
};

int main() {
    SalaryCalculator calculator;

    // Calculate salary for an Intern
    double internStipend;
    cout<< "Enter the stipend for the intern: ";
    cin>>internStipend;
    cout<< "Intern's Total Salary: " <<calculator.calculateSalary(internStipend)
<<endl;

    // Calculate salary for a Regular Employee
    double employeeBaseSalary, employeeBonuses;
    cout<< "Enter the base salary for the regular employee: ";
```

```
cin>>employeeBaseSalary;
cout<< "Enter the bonuses for the regular employee: ";
cin>>employeeBonuses;
cout<< "Regular Employee's Total Salary: "
<<calculator.calculateSalary(employeeBaseSalary, employeeBonuses) <<endl;

// Calculate salary for a Manager
double managerBaseSalary, managerBonuses, managerIncentives;
cout<< "Enter the base salary for the manager: ";
cin>>managerBaseSalary;
cout<< "Enter the bonuses for the manager: ";
cin>>managerBonuses;
cout<< "Enter the performance incentives for the manager: ";
cin>>managerIncentives;
cout<< "Manager's Total Salary: "
<<calculator.calculateSalary(managerBaseSalary, managerBonuses,
managerIncentives) <<endl;

return 0;
}
```

Output:

```
Enter the stipend for the intern: 50000
Intern's Total Salary: 50000
Enter the base salary for the regular employee: 50000
Enter the bonuses for the regular employee: 15000
Regular Employee's Total Salary: 65000
Enter the base salary for the manager: 70002
Enter the bonuses for the manager: 34324
Enter the performance incentives for the manager: 2323
Manager's Total Salary: 106649
```

```
=== Code Execution Successful ===|
```

Problem 19(Hard): Implement Polymorphism for Banking Transactions

Design a C++ program to simulate a banking system using polymorphism. Create a base class Account with a virtual method calculateInterest(). Use the derived

classes SavingsAccount and CurrentAccount to implement specific interest calculation logic:

SavingsAccount: $\text{Interest} = \text{Balance} \times \text{Rate} \times \text{Time}$.

CurrentAccount: No interest, but includes a maintenance fee deduction.

Solution:

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

```
class Account {
protected:
    double balance;
```

```
public:
    Account(double bal) : balance(bal) {}
    virtual void calculateInterest() = 0;
    virtual ~Account() {}
};
```

```
class SavingsAccount : public Account {
    double rate;
    int time;
```

```
public:
    SavingsAccount(double bal, double rate, int time) : Account(bal), rate(rate),
    time(time) {}
```

```
    void calculateInterest() override {
        double interest = balance * (rate / 100) * time;
        cout<< "Savings Account:\n";
        cout<< "Initial Balance: " << balance << endl;
        cout<< "Interest Earned: " << interest << endl;
        cout<< "Total Balance: " << (balance + interest) << endl;
    }
};
```

```
class CurrentAccount : public Account {
    double maintenanceFee;
```

```
public:
    CurrentAccount(double bal, double fee) : Account(bal), maintenanceFee(fee) {}
```

```
void calculateInterest() override {
    double finalBalance = balance - maintenanceFee;
    cout<< "Current Account:\n";
    cout<< "Initial Balance: " << balance <<endl;
    cout<< "Maintenance Fee Deducted: " << maintenanceFee<<endl;
    cout<< "Total Balance: " << finalBalance<<endl;
}

};

int main() {
    int accountType;
    double balance;

    cout<< "Enter Account Type (1 for Savings, 2 for Current): ";
    cin>> accountType;

    if (accountType < 1 || accountType > 2) {
        cout<< "Invalid account type!" <<endl;
        return 1;
    }

    cout<< "Enter Account Balance: ";
    cin>> balance;

    if (balance < 1000 || balance > 1000000) {
        cout<< "Invalid balance!" <<endl;
        return 1;
    }

    Account* account = nullptr;

    if (accountType == 1) {
        double rate;
        int time;

        cout<< "Enter Interest Rate (in %): ";
        cin>> rate;
        cout<< "Enter Time (in years): ";
        cin>> time;

        if (rate < 1 || rate > 15 || time < 1 || time > 10) {
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

```
cout<< "Invalid interest rate or time!" <<endl;
    return 1;
}

account = new SavingsAccount(balance, rate, time);

} else if (accountType == 2) {
    double fee;

cout<< "Enter Monthly Maintenance Fee: ";
cin>> fee;

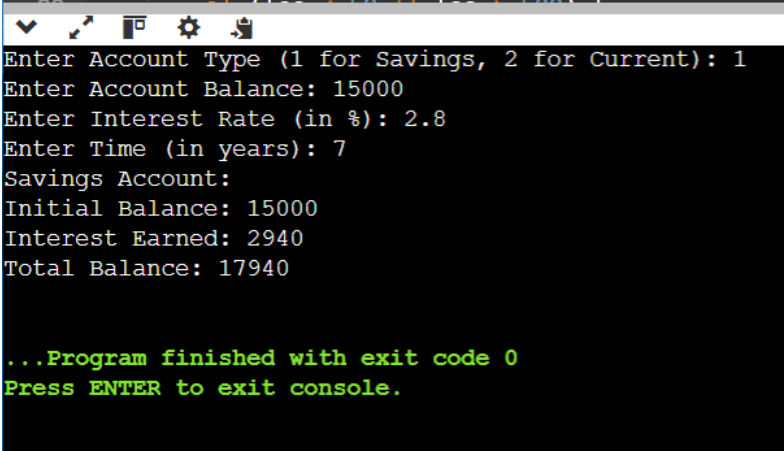
    if (fee < 50 || fee > 500) {
cout<< "Invalid maintenance fee!" <<endl;
        return 1;
    }

    account = new CurrentAccount(balance, fee);
}

if (account) {
    account->calculateInterest();
    delete account;
}

return 0;
}
```

Output:



```
Enter Account Type (1 for Savings, 2 for Current): 1
Enter Account Balance: 15000
Enter Interest Rate (in %): 2.8
Enter Time (in years): 7
Savings Account:
Initial Balance: 15000
Interest Earned: 2940
Total Balance: 17940

...Program finished with exit code 0
Press ENTER to exit console.
```




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.