

C++ PROGRAMMING LAB



Prepared by:

Name of Student: Sarthi Sanjaybhai Darji

Roll No:12

Batch: 2023-27

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Exp. No	List of Experiment
1	Write a program to find the roots of a quadratic equation.
2	Write a program to calculate the power of a number using a loop.
3	Write a program to check if a given string, is a palindrome.
4	Write a program that simulates a simple ATM machine, allowing users to check their balance, deposit, or withdraw money using a switch statement.
5	Write a program that finds the largest among three numbers using nested if-else statements
6	Write a program that determines the grade of a student based on their marks of 5 subjects using if-else-if ladder.
7	Write a program to find the sum of digits of a number until it becomes a single-digit number.
8	Write a program to print a Pascal's triangle using nested loops.
9	Write a program to calculate the sum of series $1/1! + 2/2! + 3/3! + \dots + N/N!$ using nested loops.
10	Write a program to create an array of strings and display them in alphabetical order.
11	Write a program that checks if an array is sorted in ascending order.
12	Write a program to calculate the sum of elements in each row of a matrix.
13	Write a program to generate all possible permutations of a string.

14	<p>Create a C++ program to print the following pattern:</p> <pre> ***** * * * * * * * * ***** </pre>
15	<p>Write a C++ program to display the following pattern:</p> <pre> 1 232 34543 4567654 34543 232 </pre>
16	<p>Write a program to creating an inventory management system for a small store. The system should use object-oriented principles in C++. Your program should have the following features:</p> <ul style="list-style-type: none"> • Create a Product class that represents a product in the inventory. Each Product object should have the following attributes: <ul style="list-style-type: none"> • Product ID (an integer) • Product Name (a string) • Price (a floating-point number) • Quantity in stock (an integer) • Implement a parameterized constructor for the Product class to initialize the attributes when a new product is added to the inventory.
17	<p>Write a program to manage student records. Create a class Student with attributes such as name, roll number, and marks. Implement methods for displaying student details, adding new students, and calculating the average marks of all students in the record system.</p>
18	<p>Write a program that implements a basic calculator. Use a class Calculator with methods to perform addition, subtraction, multiplication, and division of two numbers. The program should allow the user to input two numbers and select an operation to perform.</p>

19	Write a program to simulate a simple online shop. Create a class Product with attributes like name, price, and quantity in stock. Implement methods for adding products to the shopping cart, calculating the total cost, and displaying the contents of the cart.
20	Write a program to manage student grades for a classroom. Create a class Student with attributes for student name and an array to store grades. Implement methods for adding grades, calculating the average grade, and displaying the student's name and grades. Use constructors and destructors to initialize and release resources.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 1

Title: Write a program to find the roots of a quadratic equation.

Theory: This program is a quadratic equation solver. It takes three coefficients (a, b, and c) as input and outputs the nature of the roots (real, distinct, repeated, complex) and their values.

The program first calculates the discriminant, which is $b^2 - 4ac$. The discriminant determines the nature of the roots:

- If the discriminant is greater than 0, then there are two real and distinct roots.
- If the discriminant is equal to 0, then there is one real root (repeated).
- If the discriminant is less than 0, then there are two complex roots (conjugate pairs).

Once the program knows the nature of the roots, it calculates them using the quadratic formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

The program then outputs the roots to the console.

Code:

```
#include <iostream>
```

```
#include <cmath>
```

```
using namespace std;
```

```
int main() {  
  
    double a, b, c;  
  
    // Input coefficients from the user  
  
    cout << "Enter the coefficients (a, b, c) of the quadratic equation ( $ax^2 + bx + c = 0$ ):" << endl;  
  
    cout << "a = ";  
  
    cin >> a;  
  
    cout << "b = ";  
  
    cin >> b;  
  
    cout << "c = ";  
  
    cin >> c;  
  
  
    // Calculate the discriminant (D)  
  
    double discriminant = b * b - 4 * a * c;  
  
  
    if (discriminant > 0) {  
  
        // Two real and distinct roots  
  
        double root1 = (-b + sqrt(discriminant)) / (2 * a);
```

```

double root2 = (-b - sqrt(discriminant)) / (2 * a);

cout << "Roots are real and distinct:" << endl;

cout << "Root 1 = " << root1 << endl;

cout << "Root 2 = " << root2 << endl;

} else if (discriminant == 0) {

    // One real root (repeated)

    double root = -b / (2 * a);

    cout << "Roots are real and repeated:" << endl;

    cout << "Root 1 = Root 2 = " << root << endl;

} else {

    // Complex roots

    double realPart = -b / (2 * a);

    double imaginaryPart = sqrt(-discriminant) / (2 * a);

    cout << "Roots are complex and conjugate:" << endl;

    cout << "Root 1 = " << realPart << " + " << imaginaryPart << "i" << endl;

    cout << "Root 2 = " << realPart << " - " << imaginaryPart << "i" << endl;

}

return 0;

```

}

Output: (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/output"
> ./"1"
Enter the coefficients (a, b, c) of the quadratic equation (
ax^2 + bx + c = 0):
a = 2
b = 1
c = 12
Roots are complex and conjugate:
Root 1 = -0.25 + 2.4367i
Root 2 = -0.25 - 2.4367i
```

Test Case: Any two (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/output"
> ./"1"
Enter the coefficients (a, b, c) of the quadratic equation (
ax^2 + bx + c = 0):
a = 3
b = 5
c = 7
Roots are complex and conjugate:
Root 1 = -0.833333 + 1.28019i
Root 2 = -0.833333 - 1.28019i
```

```
> cd "/Users/s.d./Desktop/College/20cpp/output"
> ./"1"
Enter the coefficients (a, b, c) of the quadratic equation (
ax^2 + bx + c = 0):
a = 5
b = 6
c = 2
Roots are complex and conjugate:
Root 1 = -0.6 + 0.2i
Root 2 = -0.6 - 0.2i
```

Conclusion: This program is a simple and effective way to solve quadratic equations. It is easy to understand and use, and it can be used to solve a wide range of quadratic equations.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 2

Title: Write a program to calculate the power of a number using a loop.

Theory: power(base, exponent) function raises a base number to any exponent (positive or negative).

Handles positive exponents by repeated multiplication.

Handles negative exponents by division (reciprocal of base raised to positive exponent).

Main function inputs base and exponent, calls the function, and displays the result.

Code:

```
#include <iostream>

using namespace std;

// Function to calculate the power of a number

double power(double base, int exponent) {

    double result = 1.0;

    // Check if the exponent is positive

    if (exponent > 0) {

        for (int i = 0; i < exponent; i++) {

            result *= base;
```

```
}
```

```
}
```

```
// Check if the exponent is negative
```

```
else if (exponent < 0) {
```

```
    for (int i = 0; i < -exponent; i++) {
```

```
        result /= base;
```

```
    }
```

```
}
```

```
return result;
```

```
}
```

```
int main() {
```

```
    double base;
```

```
    int exponent;
```

```
// Get user input for base number and exponent
```

```
cout << "Enter a base number: ";

cin >> base;

cout << "Enter an exponent: ";

cin >> exponent;


// Calculate the power of the base number using the function 'power'

double result = power(base, exponent);

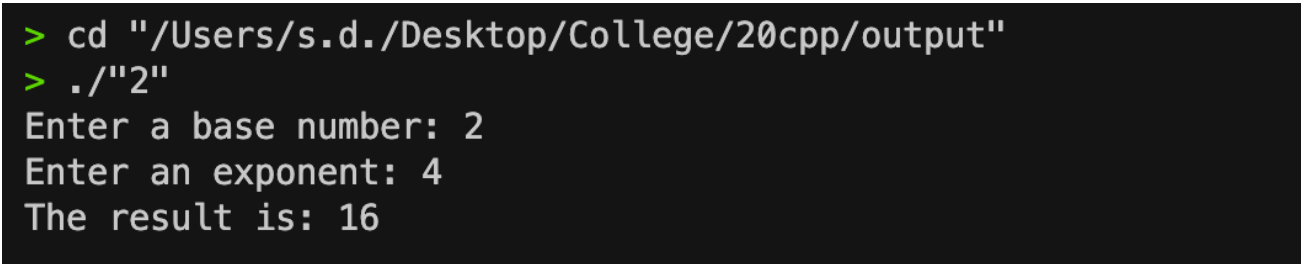

// Display the result

cout << "The result is: " << result << endl;


return 0;


}
```

Output: (screenshot)



```
> cd "/Users/s.d./Desktop/College/20cpp/output"
> ./"2"
Enter a base number: 2
Enter an exponent: 4
The result is: 16
```

Test Case: Any two (screenshot)



```
> cd "/Users/s.d./Desktop/College/20cpp/output"
> ./"2"
Enter a base number: 3
Enter an exponent: 3
The result is: 27
```

```
> cd "/Users/s.d./Desktop/College/20cpp/output"
> ./"2"
Enter a base number: 4
Enter an exponent: 2
The result is: 16
```

Conclusion:

This code effectively demonstrates a method to calculate powers of numbers in C++ using a user-defined function.

- **It handles both positive and negative exponents, ensuring a comprehensive solution for power calculations.**
- **It employs clear variable names and a logical structure, making it relatively easy to understand and follow.**

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 3

Title: Write a program to check if a given string, is a palindrome.

Theory:

isPalindrome(str) function:

Converts input string to lowercase for case-insensitive

comparison. Compares characters from both ends, moving inward.

Returns true if characters match, false otherwise.

Main function:

Prompts user for a string.

Calls isPalindrome to check if it's a palindrome.

Prints the result.

Code:

```
#include <iostream>
#include <algorithm>
using namespace std;
bool isPalindrome(string str) {
    transform(str.begin(), str.end(), str.begin(), ::tolower);
    int start = 0;
    int end = str.length() - 1;
    while (start < end) {
        if (str[start] != str[end]) {
            return false;
        }
        start++;
        end--;
    }
    return true;
```

```

}

int main() {
    string str;

    cout << "Enter a string: ";


    cin >> str;

    if (isPalindrome(str)) {
        cout << "The given string is a palindrome." << endl;
    } else {
        cout << "The given string is not a palindrome." << endl;
    }

    return 0;
}

```

Output: (screenshot)

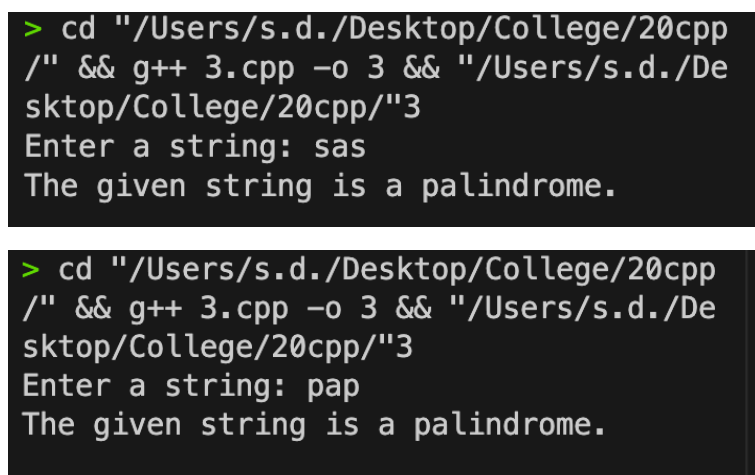


```

> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 3.cpp -o 3 && "/Users/s.d./Desktop/College/20cpp/"3
Enter a string: sarathi
The given string is not a palindrome.

```

Test Case: Any two (screenshot)



```

> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 3.cpp -o 3 && "/Users/s.d./Desktop/College/20cpp/"3
Enter a string: sas
The given string is a palindrome.

> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 3.cpp -o 3 && "/Users/s.d./Desktop/College/20cpp/"3
Enter a string: pap
The given string is a palindrome.

```

Conclusion: This code effectively determines whether a given string is a palindrome, ignoring case sensitivity. It uses a clear algorithm to compare characters from both ends, ensuring accurate results.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 4

Title: Write a program that simulates a simple ATM machine, allowing users to check their balance, deposit, or withdraw money using a switch statement.

Theory:

creating a "Account" class, resembling a real-life bank account. Imagine this class as a blueprint for your financial data. Inside, it securely stores your balance (hidden away as a private variable) and provides tools like depositing and withdrawing money. add money .The main function acts as your personal banking assistant. It presents you with a menu like a friendly ATM, allowing you to check your balance, deposit, or withdrawal. Based on your choice, it directs your request to the appropriate function within your trusty "Account" class. This interaction between the menu and the class.

Code :

```
#include <iostream>

using namespace std;

class Account {

public:

Account(int balance)

{

    this->balance = balance;

}

int get_balance()
```

```
{  
    return balance;  
}  
  
void deposit(int amount)  
{  
    balance += amount;  
}  
  
void withdraw(int amount)  
{  
    if (amount <= balance)  
    {  
        balance -= amount;  
    }  
    else  
    {  
        cout << "Insufficient funds" << endl;  
    }  
}
```

private:

int balance;

};

int main() {

Account account(1000);

cout << "1. Check balance" << endl;

cout << "2. Deposit money" << endl;

cout << "3. Withdraw money" << endl;

cout << "4. Quit" << endl;

int choice;

cin >> choice;

switch (choice)

{

case 1:

cout << "Your balance is ₹" << account.get_balance() << endl;

break;

case 2:

```
cout << "How much money would you like to deposit?" << endl;
```

```
int amount;
```

```
cin >> amount;
```

```
account.deposit(amount);
```

```
cout << "Your new balance is ₹" << account.get_balance() << endl;
```

```
break;
```

case 3:

```
cout << "How much money would you like to withdraw?" << endl;
```

```
cin >> amount;
```

```
account.withdraw(amount);
```

```
cout << "Your new balance is ₹" << account.get_balance() << endl;
```

```
break;
```

case 4:

```
cout<<"thanks for visiting";
```

```
break;
```

```
}
```

```
return 0;
```

```
}
```

Output: (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" &&
g++ 4.cpp -o 4 && "/Users/s.d./Desktop/College/20cpp/"4
1. Check balance
2. Deposit money
3. Withdraw money
4. Quit
1
Your balance is ₹1000
```

Test Case: Any two (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" &&
g++ 4.cpp -o 4 && "/Users/s.d./Desktop/College/20cpp/"4
1. Check balance
2. Deposit money
3. Withdraw money
4. Quit
2
How much money would you like to deposit?
100
Your new balance is ₹1100
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" &&
g++ 4.cpp -o 4 && "/Users/s.d./Desktop/College/20cpp/"4
1. Check balance
2. Deposit money
3. Withdraw money
4. Quit
3
How much money would you like to withdraw?
500
Your new balance is ₹500
```

Conclusion : This code effectively simulates a basic bank account system using OOP concepts. It demonstrates encapsulation, information hiding, and class structure, building a foundation for understanding OOP. It provides a clear example of object interaction and menu-driven user interfaces.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 5

Title: Write a program that finds the largest among three numbers using nested if-else statements

Theory :Read 3 numbers.Nested if statements to compare pairs.Prints the biggest number out of the 3.

Code :

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    double num1, num2, num3;
```

```
    // Input three numbers
```

```
    cout << "Enter three numbers: ";
```

```
    cin >> num1 >> num2 >> num3;
```

```
    if (num1 >= num2) {
```

```
        if (num1 >= num3) {
```

```
            cout << "The largest number is: " << num1 << endl;
```

```
        } else {
```

```

        cout << "The largest number is: " << num3 << endl;

    }

} else {

    if (num2 >= num3) {

        cout << "The largest number is: " << num2 << endl;

    } else {

        cout << "The largest number is: " << num3 << endl;

    }

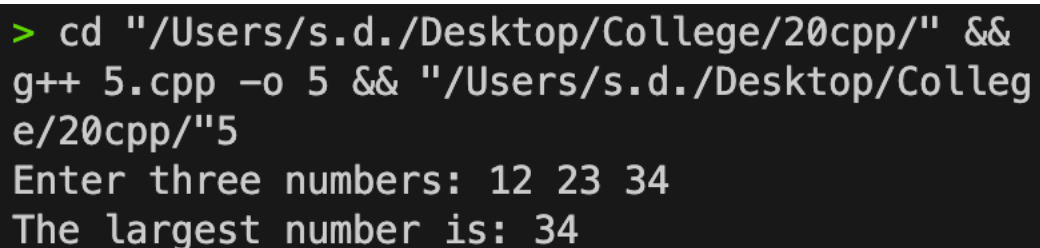
}

return 0;

}

```

Output: (screenshot)



```

> cd "/Users/s.d./Desktop/College/20cpp/" &&
g++ 5.cpp -o 5 && "/Users/s.d./Desktop/Colleg
e/20cpp/"5
Enter three numbers: 12 23 34
The largest number is: 34

```


Test Case: Any two (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" &&  
g++ 5.cpp -o 5 && "/Users/s.d./Desktop/College/20cpp/"5  
Enter three numbers: 23 12 0  
The largest number is: 23
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" &&  
g++ 5.cpp -o 5 && "/Users/s.d./Desktop/College/20cpp/"5  
Enter three numbers: 12 34 8  
The largest number is: 34
```

Conclusion : This code demonstrates a simple and effective method for finding the largest of three numbers using conditional statements. use of nested if statements for decision-making in C++.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 6

Title : Write a program that determines the grade of a student based on their marks of 5 subjects using if-else-if ladder.

Theory: Reads marks for 5 subjects from the user.Calculation: Computes the average using $(\text{sub1} + \text{sub2} + \text{sub3} + \text{sub4} + \text{sub5}) / 5$.

Grading: Assigns letter grades based on average:

A: 90 or above,B: 80-89,C: 70-79,D: 60-69,F: Below 60

Output: Prints the assigned grade.

Code :

```
#include<iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    float marks[5];
```

```
    float sub1;
```

```
    float sub2;
```

```
    float sub3;
```

```
    float sub4;
```

```
    float sub5;
```

```
    cout << "Enter marks of subject 1: ";
```

```
cin>>sub1;

cout << "Enter marks of subject 2: ";

cin>>sub2;

cout << "Enter marks of subject 3: ";

cin>>sub3;

cout << "Enter marks of subject 4: ";

cin>>sub4;

cout << "Enter marks of subject 5: ";

cin>>sub5;

float average = 0;

average=(sub1+sub2+sub3+sub4+sub5)/5;


if(average >= 90)

{

    cout << "Grade = A";

}

else if(average >= 80)

{

    cout << "Grade = B";
```

```
}  
  
else if(average >= 70)  
  
{  
  
    cout << "Grade = C";  
  
}  
  
else if(average >= 60)  
  
{  
  
    cout << "Grade = D";  
  
}  
  
else  
  
{  
  
    cout << "Grade = F";  
  
}  
  
  
return 0;  
  
}
```

Output: (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" &&  
g++ 6.cpp -o 6 && "/Users/s.d./Desktop/College/20cpp/"6  
Enter marks of subject 1: 12  
Enter marks of subject 2: 34  
Enter marks of subject 3: 56  
Enter marks of subject 4: 76  
Enter marks of subject 5: 99  
Grade = F%
```

Test Case: Any two (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" &&  
g++ 6.cpp -o 6 && "/Users/s.d./Desktop/College/20cpp/"6  
Enter marks of subject 1: 99  
Enter marks of subject 2: 89  
Enter marks of subject 3: 79  
Enter marks of subject 4: 94  
Enter marks of subject 5: 76  
Grade = B%
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" &&  
g++ 6.cpp -o 6 && "/Users/s.d./Desktop/College/20cpp/"6  
Enter marks of subject 1: 99  
Enter marks of subject 2: 97  
Enter marks of subject 3: 95  
Enter marks of subject 4: 98  
Enter marks of subject 5: 99  
Grade = A%
```

Conclusion : This code effectively calculates grade based on 5 subject marks. It demonstrates basic input, calculation, and conditional statements. It could be enhanced by using arrays for marks and a function for grading logic.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 7

Title : Write a program to find the sum of digits of a number until it becomes a single-digit number.

Theory: Takes a number. Loops until it's single-digit. Each loop, adds all its digits together. Updates the number with the sum. Prints the final single-digit sum.

Code :

```
#include <iostream>

using namespace std;

int main() {

    int num;

    cout<<"Enter a number: ";

    cin>>num;

    while(num>=10) {

        int sum=0;

        while (num>0) {

            sum+=num%10;

            num/=10;

        }

        num=sum;

    }

}
```

```
    cout<<"The sum of digits until it becomes a single-digit number  
is:"<<num<<endl;
```

```
    return 0;
```

```
}
```

Output: (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cp  
p/" && g++ 7.cpp -o 7 && "/Users/s.d./  
Desktop/College/20cpp/"7  
Enter a number: 99  
The sum of digits until it becomes a s  
ingle-digit number is:9
```

Test Case: Any two (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cp  
p/" && g++ 7.cpp -o 7 && "/Users/s.d./  
Desktop/College/20cpp/"7  
Enter a number: 89  
The sum of digits until it becomes a s  
ingle-digit number is:8
```

```
> cd "/Users/s.d./Desktop/College/20cp  
p/" && g++ 7.cpp -o 7 && "/Users/s.d./  
Desktop/College/20cpp/"7  
Enter a number: 11  
The sum of digits until it becomes a s  
ingle-digit number is:2
```


Conclusion : This code demonstrates a method for iteratively calculating the sum of digits until a single-digit result is obtained. It uses nested loops for repeated processing and digit manipulation. It highlights the use of the modulo operator for digit extraction and integer division for digit removal.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 8

Title : Write a program to print a Pascal's triangle using nested loops.

Theory: Asks for number of rows in Pascal's Triangle. Builds a 2D array, filling it with values based on Pascal's formula (adding top neighbors). Prints the Triangle with spaces for proper alignment, showing numbers in a triangle shape.

Code :

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    int numRows;
```

```
    cout << "Enter the number of rows for Pascal's Triangle: ";
```

```
    cin >> numRows;
```

```
    if (numRows <= 0) {
```

```
        cout << "Invalid input. Please enter a positive integer." << endl;
```

```
    } else {
```

```
        int triangle[numRows][numRows];
```

```
for (int i = 0; i < numRows; ++i) {  
  
    for (int j = 0; j <= i; ++j) {  
  
        if (j == 0 || j == i) {  
  
            triangle[i][j] = 1;  
  
        } else {  
  
            triangle[i][j] = triangle[i - 1][j - 1] + triangle[i - 1][j];  
  
        }  
  
    }  
  
}
```

```
cout << "Pascal's Triangle:" << endl;
```

```
for (int i = 0; i < numRows; ++i) {  
  
    for (int space = 0; space < numRows - i; ++space) {  
  
        cout << " ";  
  
    }  
  

```

```
    for (int j = 0; j <= i; ++j) {  
  
        cout << triangle[i][j] << " ";  
  
    }  
  
}
```

```
    }

    cout << endl;

}

}

return 0;

}
```

Output: (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 8.cpp
-o 8 && "/Users/s.d./Desktop/College/20cpp/"8
Enter the number of rows for Pascal's Triangle: 5
Pascal's Triangle:
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
```

Test Case: Any two (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 8.cpp  
-o 8 && "/Users/s.d./Desktop/College/20cpp/"8  
Enter the number of rows for Pascal's Triangle: 4  
Pascal's Triangle:  
      1  
     1 1  
    1 2 1  
   1 3 3 1
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 8.cpp  
-o 8 && "/Users/s.d./Desktop/College/20cpp/"8  
Enter the number of rows for Pascal's Triangle: 2  
Pascal's Triangle:  
  1  
 1 1
```

Conclusion : This code effectively demonstrates the generation and printing of Pascal's Triangle. It highlights the use of nested loops, conditional logic, and array. It provides a clear example of a mathematical concept being translated into code.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 9

Title : Write a program to calculate the sum of series $1/1! + 2/2! + 3/3! + \dots + N/N!$ using nested loops.

Theory: Reads number N. Loops from 1 to N, calculating each $N/\text{factorial}(N)$ and adding it to a total sum. Prints the final sum of the series.

Code :

```
#include <iostream>

using namespace std;

int main()

{

    int N;


    cout << "Enter the value of N: ";

    cin >> N;


    double sum = 0.0;

    double factorial = 1.0;


    for (int i = 1; i <= N; i++)

    {

        factorial *= i;
```

```
        sum += (i / factorial);  
  
    }  
  
    cout << "Sum of the series is: " << sum << endl;  
  
    return 0;  
  
}
```

Output: (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 9.cpp  
-o 9 && "/Users/s.d./Desktop/College/20cpp/"9  
Enter the value of N: 5  
Sum of the series is: 2.70833
```


Test Case: Any two (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 9.cpp  
-o 9 && "/Users/s.d./Desktop/College/20cpp/"9  
Enter the value of N: 4  
Sum of the series is: 2.66667
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 9.cpp  
-o 9 && "/Users/s.d./Desktop/College/20cpp/"9  
Enter the value of N: 3  
Sum of the series is: 2.5
```

Conclusion : This code calculates a specific series involving factorials and sums its terms. It demonstrates iterative factorial calculation and series summation using loops. It highlights the use of variables to store intermediate results.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 10

Title : Write a program to create an array of strings and display them in alphabetical order.

Theory: Reads 4 strings from user. Uses built-in sort function to arrange them alphabetically in the array. Prints all strings, now in sorted order.

Code :

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    const int size = 4;
```

```
    string arr[size];
```

```
    cout << "Enter " << size << " strings:" << endl;
```

```
    for (int i = 0; i < size; i++)
```

```
    {
```

```
        cin >> arr[i];
```

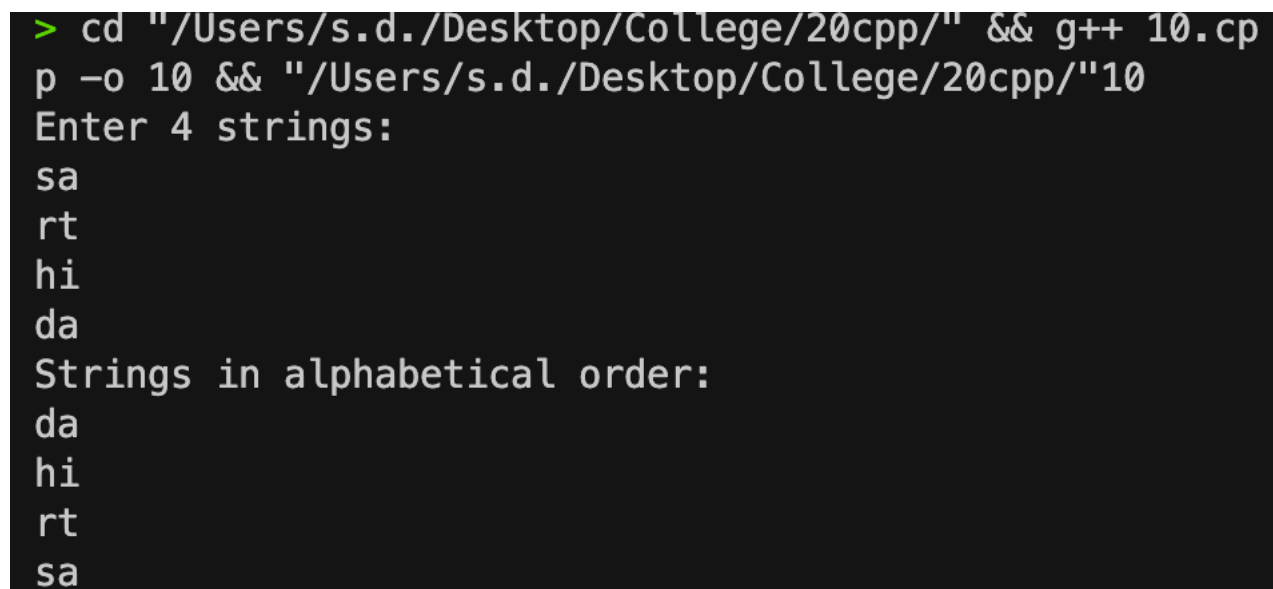
```
    }
```

```
    sort(arr, arr + size);
```

```
    cout << "Strings in alphabetical order:" << endl;
```

```
for (int i = 0; i < size; i++)  
  
    {  
  
        cout << arr[i] << endl;  
  
    }  
  
    return 0;  
  
}
```

Output: (screenshot)



```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 10.cpp  
p -o 10 && "/Users/s.d./Desktop/College/20cpp/"10  
Enter 4 strings:  
sa  
rt  
hi  
da  
Strings in alphabetical order:  
da  
hi  
rt  
sa
```

Test Case: Any two (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 10.cpp  
p -o 10 && "/Users/s.d./Desktop/College/20cpp/"10  
Enter 4 strings:  
s  
a  
r  
t  
Strings in alphabetical order:  
a  
r  
s  
t
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 10.cpp  
p -o 10 && "/Users/s.d./Desktop/College/20cpp/"10  
Enter 4 strings:  
sarthi  
sparsh  
jeevan  
lakhshay  
Strings in alphabetical order:  
jeevan  
lakhshay  
sarthi  
sparsh
```

Conclusion : This code effectively sorts an array of strings alphabetically using the sort function. It demonstrates the use of arrays for string storage, user input, and the sort function for efficient sorting. It provides a concise example of string manipulation and sorting.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 11

Title :Write a program that checks if an array is sorted in ascending order.

Theory:Function checks if numbers in an array are in ascending order, one by one. If any number is smaller than the one before it, it's not sorted, returns false. If all numbers are bigger than the ones before them, it's sorted, returns true. Main program uses this function on a pre-defined array to see if it's sorted and prints a message.

Code :

```
#include <iostream>
```

```
using namespace std;
```

```
bool isSorted(int arr[], int n) {
```

```
    for (int i = 1; i < n; i++) {
```

```
        if (arr[i] < arr[i - 1]) {
```

```
            return false;
```

```
        }
```

```
    }
```

```
    return true;
```

```
}
```

```
int main() {
```

```
    int arr[] = {22, 13, 34, 55, 63};
```

```
    int n = sizeof(arr) / sizeof(arr[0]);
```

```

if (isSorted(arr, n)) {

    cout << "The array is sorted in ascending order." <<endl;

} else {

    cout << "The array is not sorted in ascending order." <<endl;

}

return 0;

}

```

Output: (screenshot)

```

int arr[] = {2, 3, 4, 35, 6};

> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 11.cpp -o 11 && "/Users/s.d./Desktop/College/20cpp/"11
The array is not sorted in ascending order.

```

Test Case: Any two (screenshot)

```

int arr[] = {2, 3, 4, 5, 6};

> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 11.cpp -o 11 && "/Users/s.d./Desktop/College/20cpp/"11
The array is sorted in ascending order.

```

```
int arr[] = {22,13, 34, 55, 63};
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 11.cpp  
-o 11 && "/Users/s.d./Desktop/College/20cpp/"11  
The array is not sorted in ascending order.
```

Conclusion : This code demonstrates a simple method to check if an array is sorted in ascending order. It highlights the use of functions for logical checking. It could be improved by handling descending order and allowing for different sorting criteria.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 12

Title :Write a program to calculate the sum of elements in each row of a matrix.

Theory: Reads size of a matrix (rows and columns) and checks if valid (> 0). Takes inputs for each element of the matrix. For each row, calculates the sum of all its elements and prints it with the row number.

Code :

```
#include <iostream>

using namespace std;

int main()
{
    int rows, cols;

    cout << "Enter the number of rows: ";

    cin >> rows;

    cout << "Enter the number of columns: ";

    cin >> cols;

    if (rows <= 0 || cols <= 0)
    {
        cout << "Invalid input. Rows and columns should be positive integers." << endl;

        return 1;
    }
```

```
int matrix[rows][cols];

cout << "Enter the elements of the matrix:" << endl;

for (int i = 0; i < rows; ++i)

{

    for (int j = 0; j < cols; ++j)

    {

        cin >> matrix[i][j];

    }

    cout << endl;

}
```

```
cout << "Sum of elements in each row:" << endl;

for (int i = 0; i < rows; ++i)

{

    int rowSum = 0;

    for (int j = 0; j < cols; ++j)

    {

        rowSum = rowSum+matrix[i][j];

    }

}
```

```
    }  
  
    cout << "Row " << i + 1 << ": " << rowSum << endl;  
  
}  
  
return 0;  
  
}
```

Output: (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 12.cpp  
-o 12 && "/Users/s.d./Desktop/College/20cpp/"12  
Enter the number of rows: 3  
Enter the number of columns: 3  
Enter the elements of the matrix:  
1 2 3  
  
4 5 6  
  
7 8 9  
  
Sum of elements in each row:  
Row 1: 6  
Row 2: 15  
Row 3: 24
```

Test Case: Any two (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 12.cpp -o 12 && "/Users/s.d./Desktop/College/20cpp/"12
Enter the number of rows: 3
Enter the number of columns: 3
Enter the elements of the matrix:
10 10 10

20 20 20

30 30 30

Sum of elements in each row:
Row 1: 30
Row 2: 60
Row 3: 90
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 12.cpp -o 12 && "/Users/s.d./Desktop/College/20cpp/"12
Enter the number of rows: 3
Enter the number of columns: 3
Enter the elements of the matrix:
11 22 33

1 2 3

10 20 30

Sum of elements in each row:
Row 1: 66
Row 2: 6
Row 3: 60
```

Conclusion : This code effectively calculates and prints the sum of elements in each row of a user-defined matrix. It demonstrates matrix input, validation, nested loops for row-wise processing, and output formatting. It could be extended to calculate column sums or overall matrix sum for further analysis.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 13

Title :Write a program to generate all possible permutations of a string.

Theory:Takes a string. Swaps each letter with all other letters in turn, one by one. For each swap, it tries all possible swaps further down the string (recursion). If it reaches the end without further swaps, prints the result as a complete permutation. Does this for all possible initial swaps, generating all unique permutations of the original string.

Code :

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
void swap(char &a, char &b) {
```

```
    char temp = a;
```

```
    a = b;
```

```
    b = temp;
```

```
}
```

```
void generatePermutations(string str, int left, int right) {
```

```
    if (left == right) {
```

```
        cout << str << endl;
```

```
        return;
```

```
}
```

```
for (int i = left; i <= right; i++) {  
  
    swap(str[left], str[i]);  
  
    generatePermutations(str, left + 1, right);  
  
    swap(str[left], str[i]);  
  
}  
  
}
```

```
int main() {  
  
    string input;  
  
    cout << "Enter a string: ";  
  
    cin >> input;  
  
    int length = input.length();  
  
    if (length == 0) {  
  
        cout << "Please enter a non-empty string." << endl;  
  
        return 1;  
  
    }  
  
    cout << "Permutations of \" " << input << "\":" << endl;  
  
    generatePermutations(input, 0, length - 1);  
  
    return 0;
```

}

Output: (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 13.cp
p -o 13 && "/Users/s.d./Desktop/College/20cpp/"13
Enter a string: sas
Permutations of "sas":
sas
ssa
ass
ass
sas
ssa
```

Test Case: Any two (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 13.cp
p -o 13 && "/Users/s.d./Desktop/College/20cpp/"13
Enter a string: Tank
Permutations of "Tank":
Tank
Takn
Tnak
Tnka
Tkna
Tkan
aTnk
aTkn
anTk
ankT
aknT
akTn
naTk
nakT
nTak
nTka
nkTa
nkaT
kanT
kaTn
knaT
knTa
kTna
kTan
```



```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 13.cpp  
p -o 13 && "/Users/s.d./Desktop/College/20cpp/"13  
Enter a string: cat  
Permutations of "cat":  
cat  
cta  
act  
atc  
tac  
tca
```

Conclusion : This code effectively generates and prints all permutations of a given string using recursion. It highlights the use of recursion for backtracking and exploring different combinations. It demonstrates string manipulation and character swapping techniques. It provides a clear example of solving a combinatorial problem with recursion.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 14

Title : Create a C++ program to print the following pattern:

* *

* *

* *

Theory: Makes a rectangle of stars using loops. Only prints stars on the edges (top, bottom, left, right). Fills the rest with spaces.

Code :

```
#include <iostream>
```

```
using namespace std;
```

```
int main() {
```

```
    int rows = 8;
```

```
    int cols = 8;
```

```
    for (int i = 1; i <= rows; i++) {
```

```
        for (int j = 1; j <= cols; j++) {
```

```

        if (i == 1 || i == rows || j == 1 || j == cols) {

            cout << "*";

        } else {

            cout << " ";

        }

    }

    cout << endl;

}

return 0;

}

```

Output: (screenshot)

```

int rows = 6;
int cols = 6;

```

```

> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 14.cpp -o 14 && "/Users/s.d./Desktop/College/20cpp/"14
*****
*      *
*      *
*      *
*      *
*****

```

Test Case: Any two (screenshot)

```

int rows = 7;
int cols = 7;

```

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 14.cpp -o 14 && "/Users/s.d./Desktop/College/20cpp/"14
*****
*      *
*      *
*      *
*      *
*      *
*****
```

```
int rows = 8;
int cols = 8;
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 14.cpp -o 14 && "/Users/s.d./Desktop/College/20cpp/"14
*****
*      *
*      *
*      *
*      *
*      *
*      *
*****
```

Conclusion: This code effectively prints a hollow rectangle of stars with a specified number of rows and columns. It demonstrates the use of nested loops for pattern generation and conditional statements for decision-making. It could be modified to create different shapes or patterns with varying characters and dimensions.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 15

Title : Write a C++ program to display the following pattern:

1

232

34543

4567654

34543

232

1

Theory: Loops build a number triangle. Spaces indent each line for proper shape. Numbers go up on left side, down on right side. Makes a mountain-like shape with numbers.

Code :

```
#include <iostream>
```

```
using namespace std;
```

```
int main()
```

```
{
```

```
    int i, j, k, l;
```

```
    for (i = 1; i <= 4; i++)
```

```
    {
```

```
        for (j = 4; j >= i; j--)
```

```

        cout << " ";

    for (k = i; k <= 2 * i - 1; k++)

        cout << k;

    for (l = 2 * i - 2; l >= i; l--)

        cout << l;

    cout << endl;

}

for (i = 3; i >= 1; i--)

{

    for (j = 4; j >= i; j--)

        cout << " ";

    for (k = i; k <= 2 * i - 1; k++)

        cout << k;

    for (l = 2 * i - 2; l >= i; l--)

        cout << l;

    cout << endl;

}

return 0;

}

```

Output: (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 15.
cpp -o 15 && "/Users/s.d./Desktop/College/20cpp/"15
1
232
34543
4567654
34543
232
1
```

Test Case: Any two (screenshot) Change the value of I

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 15.
cpp -o 15 && "/Users/s.d./Desktop/College/20cpp/"15
1
232
34543
4567654
567898765
4567654
34543
232
1
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 15.  
cpp -o 15 && "/Users/s.d./Desktop/College/20cpp/"15  
1  
232  
34543  
232  
1
```

Conclusion: This code effectively prints a numerical pattern with a triangular shape. It demonstrates the use of nested loops for pattern generation and numerical manipulation. It could be modified to create different shapes, number ranges, or spacing patterns.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 16

Title :Write a program to creating an inventory management system for a small store. The system should use object-oriented principles in C++. Your program should have the following features:

- Create a **Product** class that represents a product in the inventory. Each **Product** object should have the following attributes:
 - Product ID (an integer)
 - Product Name (a string)
 - Price (a floating-point number)
 - Quantity in stock (an integer)
- Implement a parameterized constructor for the **Product** class to initialize the attributes when a new product is added to the inventory.

Theory: Builds a "Product" class to store details like ID, name, price, and quantity. Protects data from outside interference by making most things private. Lets you create and manage products like filling boxes with their information. You can also change things inside the box, if needed.

Code :

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
class Product {
```

```
    private:
```

```
        int productID;
```

```
string productName;
```

```
float price;
```

```
int quantityInStock;
```

```
public:
```

```
Product(int id,string name, float price, int quantity) {
```

```
    productID = id;
```

```
    productName = name;
```

```
    this->price = price;
```

```
    quantityInStock = quantity;
```

```
}
```

```
int getProductID() { return productID; }
```

```
string getProductName() { return productName; }
```

```
float getPrice() { return price; }
```

```
int getQuantityInStock() { return quantityInStock; }
```

```
void setProductID(int id) { productID = id; }
```

```
void setProductName(string name) { productName = name; }
```

```
void setPrice(float price) { this->price = price; }

void setQuantityInStock(int quantity) { quantityInStock = quantity; }

};
```

```
int main() {

    Product p1(1, "Product1", 100.0, 10);

    Product p2(2,"product 2",1000.,10);

    cout << "Product ID: " << p2.getProductID() <<endl;

    cout << "Product Name: " << p2.getProductName() << endl;

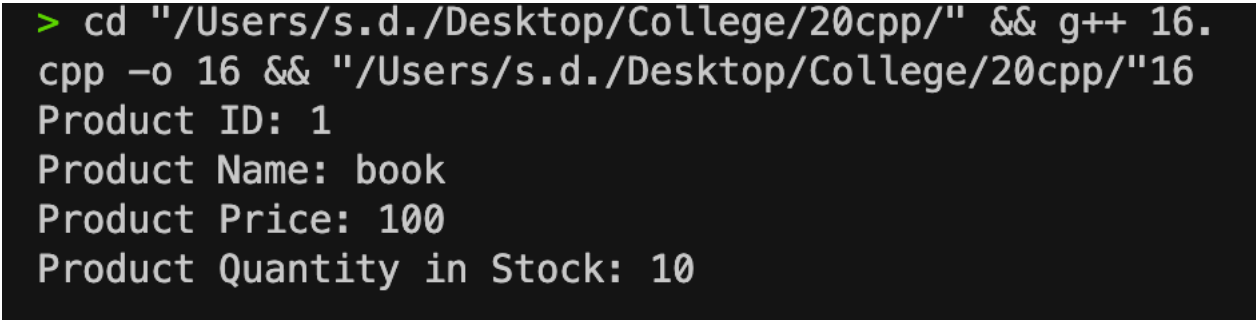
    cout << "Product Price: " << p2.getPrice() << endl;

    cout << "Product Quantity in Stock: " << p2.getQuantityInStock()
<<endl;

    return 0;

}
```

Output: (screenshot)



```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 16.
cpp -o 16 && "/Users/s.d./Desktop/College/20cpp/"16
Product ID: 1
Product Name: book
Product Price: 100
Product Quantity in Stock: 10
```

Test Case: Any two (screenshot) Change the value of Products

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 16.
cpp -o 16 && "/Users/s.d./Desktop/College/20cpp/"16
Product ID: 3
Product Name: Macbook
Product Price: 123000
Product Quantity in Stock: 46
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 16.
cpp -o 16 && "/Users/s.d./Desktop/College/20cpp/"16
Product ID: 2
Product Name: Electric pen
Product Price: 1000
Product Quantity in Stock: 10
```

Conclusion: This code models a Product with attributes and methods using a class structure. It demonstrates encapsulation (protecting data with private access). It highlights the use of constructors for initialization and getters/setters for data access. It provides a foundation for building more complex product-related applications.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 17

Title :Write a program to manage student records. Create a class Student with attributes such as name, roll number, and marks. Implement methods for displaying student details, adding new students, and calculating the average marks of all students in the record system.

Theory: Creates a blueprint for students with name, roll number, and marks. Organizes students into a class record like a virtual classroom. Allows adding, viewing, and calculating average marks for students.

Code :

```
#include <iostream>
```

```
#include<vector>
```

```
#include<algorithm>
```

```
using namespace std;
```

```
class Student {
```

```
    private:
```

```
        string name;
```

```
        int roll_number;
```

```
        float marks;
```

```
    public:
```

```
        Student(string n, int r, float m) {
```

```
    name = n;  
  
    roll_number = r;  
  
    marks = m;  
  
}
```

```
void displayStudentDetails() {  
  
    cout << "Name: " << name << ", Roll Number: " << roll_number <<  
    ", Marks: " << marks << std::endl;  
  
}
```

```
string getName() {  
  
    return name;  
  
}
```

```
float getMarks() {  
  
    return marks;  
  
}
```

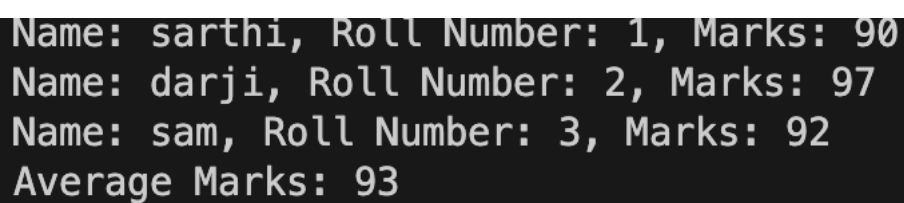
```
};
```

```
class StudentRecord {  
  
    private:  
  
        vector<Student> students;  
  
  
    public:  
  
        void addStudent(Student student) {  
  
            students.push_back(student);  
  
        }  
  
  
        void displayStudentDetails() {  
  
            for (auto &student : students) {  
  
                student.displayStudentDetails();  
  
            }  
  
        }  
  
  
        float calculateAverageMarks() {  
  
            float total_marks = 0;  
  
            for (auto &student : students) {  
  
                total_marks += student.getMarks();  
  
            }  
  
        }  
  
};
```

```
    }  
  
    return total_marks / students.size();  
  
}  
  
};
```

```
int main() {  
  
    StudentRecord sr;  
  
    sr.addStudent(Student("sarathi", 1, 99));  
  
    sr.addStudent(Student("darji", 2, 97));  
  
    sr.addStudent(Student("sam", 3, 95));  
  
  
    sr.displayStudentDetails();  
  
    cout << "Average Marks: " << sr.calculateAverageMarks() << endl;  
  
  
    return 0;  
  
}
```

Output: (screenshot)

A screenshot of a terminal window showing the output of the program. The text is white on a black background. It lists three students: sarathi with roll number 1 and marks 90, darji with roll number 2 and marks 97, and sam with roll number 3 and marks 92. At the bottom, it shows the calculated average marks as 93.

```
Name: sarathi, Roll Number: 1, Marks: 90  
Name: darji, Roll Number: 2, Marks: 97  
Name: sam, Roll Number: 3, Marks: 92  
Average Marks: 93
```


Test Case: Any two (screenshot) Change the Marks

```
Name: sarthi, Roll Number: 1, Marks: 75  
Name: darji, Roll Number: 2, Marks: 87  
Name: sam, Roll Number: 3, Marks: 82  
Average Marks: 81.3333
```

```
Name: sarthi, Roll Number: 1, Marks: 99  
Name: darji, Roll Number: 2, Marks: 97  
Name: sam, Roll Number: 3, Marks: 95  
Average Marks: 97
```

Conclusion : This code demonstrates or representing student data. It uses classes for modeling real-world entities and managing collections of objects. It highlights the use of constructors, getters/setters, and methods for object interaction. It provides a foundation for building more complex student management systems.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 18

Title :Write a program that implements a basic calculator. Use a class Calculator with methods to perform addition, subtraction, multiplication, and division of two numbers. The program should allow the user to input two numbers and select an operation to perform.

Theory: Creates a "Calculator" toolbox with functions for addition, subtraction, multiplication, and division. Guides the user through entering numbers and choosing an operation. Prevents crashes from division by zero by throwing an error message.

Code :

```
#include <iostream>
```

```
using namespace std;
```

```
class Calculator {
```

```
public:
```

```
    static double add(double a, double b) {
```

```
        return a + b;
```

```
    }
```

```
    static double subtract(double a, double b) {
```

```
        return a - b;
```

```
    }
```

```
    static double multiply(double a, double b) {
```

```
    return a * b;  
}
```

```
static double divide(double a, double b) {  
    if (b == 0) {  
        throw invalid_argument("Division by zero is not allowed.");  
    }  
    return a / b;  
}  
};
```

```
int main() {  
    double num1, num2;  
    char operation;  
  
    cout << "Enter the first number: ";  
    cin >> num1;  
  
    cout << "Enter the second number: ";  
    cin >> num2;
```

```
cout << "Enter the operation (+, -, *, /): ";
```

```
cin >> operation;
```

```
switch (operation) {
```

```
    case '+':
```

```
        cout << "The sum is " << Calculator::add(num1, num2) << "." << endl;
```

```
        break;
```

```
    case '-':
```

```
        cout << "The difference is " << Calculator::subtract(num1, num2) << "." << endl;
```

```
        break;
```

```
    case '*':
```

```
        cout << "The product is " << Calculator::multiply(num1, num2) << "." << endl;
```

```
        break;
```

```
    case '/':
```

```
        try {
```

```
            cout << "The quotient is " << Calculator::divide(num1, num2) << "." << endl;
```

```

    } catch (const invalid_argument& e) {

        cerr << e.what() << endl;

    }

    break;

default:

    cerr << "Invalid operation." << endl;

    break;

}

return 0;

}

```

Output: (screenshot)

```

> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 18.
cpp -o 18 && "/Users/s.d./Desktop/College/20cpp/"18
Enter the first number: 12
Enter the second number: 23
Enter the operation (+, -, *, /): +
The sum is 35.

```

Test Case: Any two (screenshot)

```

> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 18.
cpp -o 18 && "/Users/s.d./Desktop/College/20cpp/"18
Enter the first number: 20
Enter the second number: 10
Enter the operation (+, -, *, /): /
The quotient is 2.

```

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 18.  
cpp -o 18 && "/Users/s.d./Desktop/College/20cpp/"18  
Enter the first number: 2  
Enter the second number: 45  
Enter the operation (+, -, *, /): *  
The product is 90.
```

Conclusion : This code demonstrates a simple calculator using a class to organize and execute arithmetic operations. It highlights the use of static methods, user input, switch statements, and exception handling. It provides a basic foundation for building more complex calculators or mathematical applications.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 19

Title : Write a program to simulate a simple online shop. Create a class Product with attributes like name, price, and quantity in stock. Implement methods for adding products to the shopping cart, calculating the total cost, and displaying the contents of the cart.

Theory: Creates "product" objects like book, pen, and notebook. Lets you show all available products or pick one to add to the virtual cart. Keeps track of how many of each product are in the cart. Shows you what's in your cart after you add things. Repeats until you stop shopping.

Code :

```
#include <iostream>
```

```
#include <string>
```

```
using namespace std;
```

```
class Product {
```

```
public:
```

```
    string name;
```

```
    double price;
```

```
    int quantity;
```

```
    // Simulate adding a product to the cart using a counter
```

```
void addProduct() {  
  
    quantity++;  
  
    cout << "Added " << name << " to cart." << endl;  
  
}
```

// Display product details

```
void displayProduct() {  
  
    cout << "Product: " << name << endl;  
  
    cout << "Price: " << price<<"₹"<< endl;  
  
    cout << "Quantity in cart: " << quantity << endl;  
  
}
```

```
};
```

```
int main() {  
  
    // Individual products (no vector)  
  
    Product book = {"Book", 1599, 0};  
  
    Product pen = {"Redbull", 250, 0};  
  
    Product notebook = {"Notebook", 799, 0};
```



```
int choice;
```

```
char addMore;
```

```
do {
```

```
    cout << "\nOnline Shop:" << endl;
```

```
    cout << "1. Display Products" << endl;
```

```
    cout << "2. Add Product to Cart" << endl;
```

```
    cout << "3. View Cart" << endl;
```

```
    cout << "Enter your choice: ";
```

```
    cin >> choice;
```

```
    switch (choice) {
```

```
        case 1:
```

```
            // Display available products
```

```
            cout << "1. Book" << endl;
```

```
            cout << "2. Pen" << endl;
```

```
            cout << "3. Notebook" << endl;
```

```
            break;
```

```
        case 2:
```

```
int productIndex;

cout << "Enter product number to add: ";

cin >> productIndex;

if (productIndex >= 1 && productIndex <= 3) {

    switch (productIndex) {

        case 1:

            book.addProduct();

            break;

        case 2:

            pen.addProduct();

            break;

        case 3:

            notebook.addProduct();

            break;

    }

} else {

    cout << "Invalid product number." << endl;

}

break;
```

case 3:

```
// Display cart contents
```

```
cout << "\nCart Items:" << endl;
```

```
if (book.quantity > 0) {
```

```
    book.displayProduct();
```

```
}
```

```
if (pen.quantity > 0) {
```

```
    pen.displayProduct();
```

```
}
```

```
if (notebook.quantity > 0) {
```

```
    notebook.displayProduct();
```

```
}
```

```
break;
```

default:

```
cout << "Invalid choice." << endl;
```

```
}
```

```
cout << "\nAdd more products? (Y/N): ";
```

```
cin >> addMore;
```

```
} while (addMore == 'y' || addMore == 'Y');
```

```
return 0;
```

```
}
```

Output: (screenshot)

```
Online Shop:
1. Display Products
2. Add Product to Cart
3. View Cart
Enter your choice: 3
```

```
Cart Items:
Product: Book
Price: 1599₹
Quantity in cart: 1
Product: Redbull
Price: 250₹
Quantity in cart: 1
```

```
Add more products? (Y/N): █
```

Test Case: Any two (screenshot)

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 19.
cpp -o 19 && "/Users/s.d./Desktop/College/20cpp/"19
```

```
Online Shop:
1. Display Products
2. Add Product to Cart
3. View Cart
Enter your choice: 2
Enter product number to add: 1
Added Book to cart.
```

```
Add more products? (Y/N): █
```

```
> cd "/Users/s.d./Desktop/College/20cpp/" && g++ 19.  
cpp -o 19 && "/Users/s.d./Desktop/College/20cpp/"19
```

Online Shop:

1. Display Products
2. Add Product to Cart
3. View Cart

Enter your choice: 1

1. Book
2. Pen
3. Notebook

Conclusion : This code simulates a basic online shopping experience with a limited product catalog. It highlights user interaction through menus and choices. It's a foundation for more complex shopping systems, needing improvements. Store products in a vector or array for better management. Calculate total cart value and implement checkout functionality. Consider database integration for persistent product storage.

Name of Student: Sarthi Sanjaybhai Darji

Roll Number: 12

Experiment No: 20

Title : Write a program to manage student grades for a classroom. Create a class Student with attributes for student name and an array to store grades. Implement methods for adding grades, calculating the average grade, and displaying the student's name and grades. Use constructors and destructors to initialize and release resources.

Theory: Builds a "Student" blueprint with name and a list of grades. Lets you create student objects, add grades, and calculate averages. Displays student info in a clear format.

Code :

```
// Header file for Student class
```

```
#include <iostream>
```

```
#include <string>
```

```
#include <vector>
```

```
using namespace std;
```

```
// Definition of Student class
```

```
class Student {
```

```
private:
```

```
    string name; // Name of the student
```

```
    vector<int> grades; // Grades of the student
```

```
public:
```

// Constructor of Student class that takes a string argument for the name of the student

```
Student(string n) : name(n) {}
```

// Method to add a grade to the student's grade list

```
void addGrade(int grade) {  
    grades.push_back(grade);  
}
```

// Method to calculate the average grade of the student

```
double calculateAverage() {  
    double sum = 0;  
    for (int grade : grades) {  
        sum += grade;  
    }  
    return grades.empty() ? 0 : sum / grades.size();  
}
```

// Method to display the information of the student

```
void displayInfo() {  
  
    cout << "\nStudent: " << name << endl;  
  
    cout << "Grades: ";  
  
    for (int grade : grades) {  
  
        cout << grade << " ";  
  
    }  
  
    cout << endl;  
  
    cout << "Average Grade: " << calculateAverage() << endl;  
  
}
```

```
// Default constructor of Student class
```

```
Student() {}
```

```
};
```

```
// Main function to demonstrate the functionality of the Student class
```

```
int main() {
```

```
    // Creating two student objects and adding grades to their grade lists
```

```
    Student s1("Sarathi Darji");
```

```
    s1.addGrade(90);
```



```
s1.addGrade(80);
```

```
s1.addGrade(70);
```

```
// Displaying the information of the first student
```

```
s1.displayInfo();
```

```
// Creating another student object and adding grades to their grade list
```

```
Student s2("Sparsh Sharma");
```

```
s2.addGrade(80);
```

```
s2.addGrade(70);
```

```
s2.addGrade(60);
```

```
// Displaying the information of the second student
```

```
s2.displayInfo();
```

```
return 0;
```

```
}
```

Output: (screenshot)

```
Student: Sarthi Darji  
Grades: 80 85 90  
Average Grade: 85  
  
Student: Sparsh Sharma  
Grades: 70 75 80  
Average Grade: 75
```

Test Case: Any two (screenshot)

```
Student: Sarthi Darji  
Grades: 80 80 80  
Average Grade: 80  
  
Student: Sparsh Sharma  
Grades: 70 70 70  
Average Grade: 70
```

```
Student: Sarthi Darji  
Grades: 90 80 70  
Average Grade: 80  
  
Student: Sparsh Sharma  
Grades: 80 70 60  
Average Grade: 70
```

Conclusion : This code effectively creates a student object model with basic grade management capabilities. It highlights the use of constructors, methods, and data storage within classes. It provides a foundation for building more complex student management systems or educational applications.

THANK YOU!!

