INSTITUTE OF TECHNOLOGY AND MANAGEMENT
SKILLS UNIVERSITY,
KHARGHAR, NAVI MUMBAI

C++ PROGRAMMING LAB

Prepared by:

Name of Student: _____Yashika Pandurang Thakur_____

Roll No: _____23_____

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! + ... + 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	Create a C++ program to print the following pattern: ***** * * * * * * *****
15	Write a C++ program to display the following pattern: 1 232 34543 4567654 34543 232

16	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.

17	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.
18	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.

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: _____Yashika Thakur_____

Roll Number: _____23_____

Experiment No: 1

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

Theory:

A quadratic equation is a second-order polynomial equation in a single variable, typically represented as:

ax^2+bx+c=0

Here, a, b, and c are coefficients, and x is the variable. The solutions to this equation, known as the roots, can be found using the quadratic formula.

The quadratic formula provides the solutions to a quadratic equation in the form

x=-b+-root b^2-4ac/2a

b^2-4ac, is known as the discriminant.

Code:

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

int main() {
    double a, b, c;

    // Input coefficients from the user
    cout << "Enter coefficient a: ";
    cin >> a;

    cout << "Enter coefficient b: ";
    cin >> b;

    cout << "Enter coefficient c: ";
    cin >> c;

    // Calculate the discriminant (b^2 - 4ac)

    // Check the discriminant to determine the nature of roots
    if (discriminant > 0) {
        // Two distinct real 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 (double root)
        double root = -b / (2 * a);

        cout << "Roots are real and identical:" <<endl;
        cout << "Root = " << 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;
    }
```
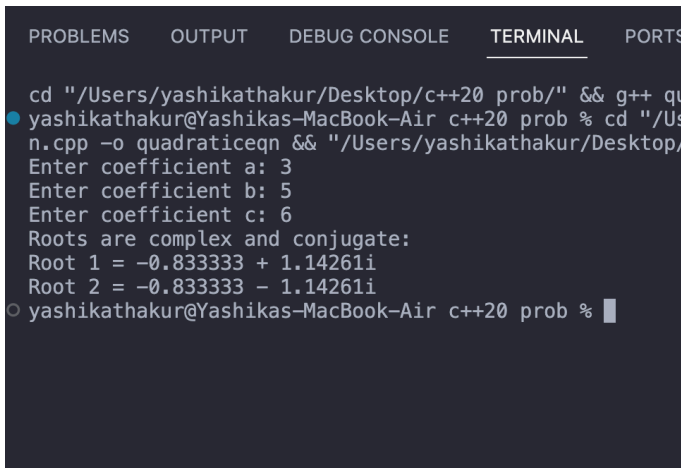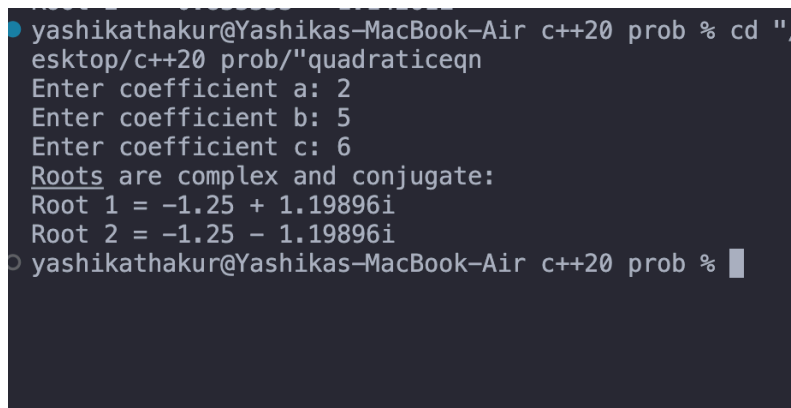
```
        return 0;
```
Output: (screenshot)

Test Case: Any two (screenshot)

Case1:

Case2:

Conclusion:

The program accurately calculates the roots of a quadratic equation, providing real, repeated, or imaginary solutions based on the discriminant.

Experiment No: 2

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

Theory:

Power calculation involves multiplying the base by itself for the specified exponent using a loop.

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

int main() {
    double base, power, result = 1;

    // Input base and exponent from the user
    cout << "Enter the base: ";
    cin >> base;

    cout << "Enter the power: ";
    cin >> power;

    // Calculate power using a loop
    for (int i = 0; i < power; i++) {
        result *= base;
    }

    // Display the result
    cout << base << " raised to the power " << power << " is: " << result << endl;

    return 0;
```

}

Output: (screenshot)

Test Case: Any two (screenshot)

Case1:

Case2:

Conclusion:

The program effectively computes the power of a number using a loop, providing the result by iteratively multiplying the base with itself according to the given exponent.

Experiment No: 3

Title:

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

Theory:

To determine if a given string is a palindrome, you can compare the characters from the beginning with the characters from the end. If the string reads the same in both directions, it is a palindrome.

Code:

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

bool isPalindrome(const string& str) {
    int start = 0;
    int end = str.length() - 1;

    while (start < end) {
        if (str[start] != str[end]) {
            return false; // Not a palindrome
        }
        start++;
        end--;
    }

    return true; // Palindrome
}

int main() {
    string inputString;
```
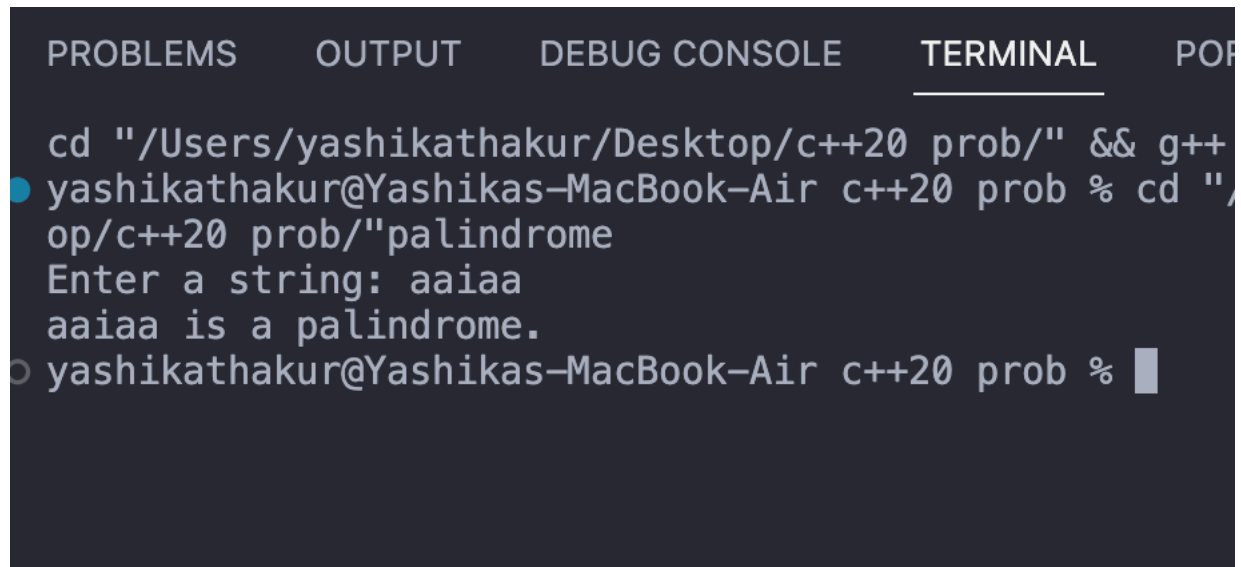
```
    cout << "Enter a string: ";
    getline(cin, inputString);

    if (isPalindrome(inputString)) {
        cout <<   inputString <<" " <<"is a palindrome." << endl;
    } else {
        cout << inputString <<" "<<" is not a palindrome." << endl;
    }

    return 0;
}
```
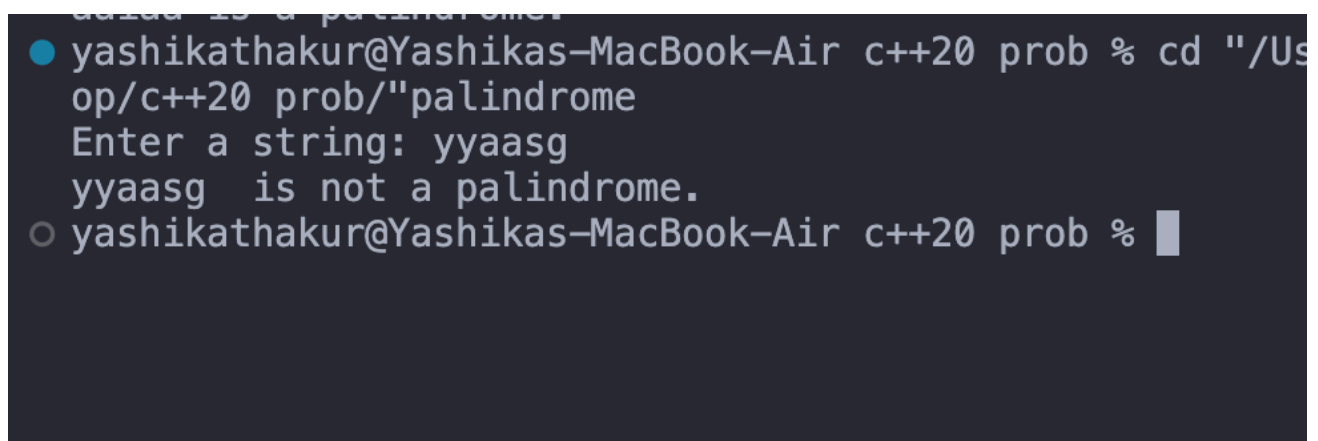
Output: (screenshot)

Test Case: Any two (screenshot)

Case1:

Case2:


```
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd "
  op/c++20 prob/"palindrome
  Enter a string: ssdss
  ssdss is a palindrome.
○ yashikathakur@Yashikas-MacBook-Air c++20 prob % ▌
```

Conclusion:

This program defines a function `isPalindrome` that cleans the input string by removing spaces and converting it to lowercase. It then checks if the cleaned string is a palindrome by comparing characters from the beginning and end. The `main` function takes user input and calls this function to determine and display whether the input string is a palindrome or not.

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:

A simple ATM simulation program allows users to perform basic operations like checking balance, depositing, and withdrawing money. This can be achieved using a switch statement in C++.

Code:

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

// Function to display the menu
void displayMenu()
```

```cpp
{
    cout << "ATM Menu:";
    cout << "1. Check Balance";
    cout << "2. Deposit";
    cout << "3. Withdraw";
    cout << "4. Exit";
}

int main()
{
    double balance = 1000.0; // Initial balance
    int choice;
    double amount;

    do
    {
        displayMenu(); // Display the menu

        // Get user choice
        cout << "Enter your choice (1-4): ";
        cin >> choice;

        // Process user choice using a switch statement
        switch (choice)
        {
            case 1:
                // Check Balance
                cout << "Your balance is: "  << balance <<endl;
                break;

            case 2:
                // Deposit
                cout << "Enter the amount to deposit: ";
                cin >> amount;
                if (amount > 0)
                {
                    balance += amount;
                    cout << "Deposit successful. Your new balance
is: "  << balance << endl;
                } else {
                    std::cout << "Invalid amount. Please enter a
positive value.";
                }
                break;

            case 3:
                // Withdraw
                cout << "Enter the amount to withdraw: ";
```

```cpp
                cin >> amount;
                if (amount > 0 && amount <= balance)
                {
                    balance -= amount;
                    cout << "Withdrawal successful. Your new
balance is: " << balance <<endl;
                } else
                {
                    cout << "Invalid amount or insufficient
funds.";
                }
                break;

            case 4:
                // Exit
                cout << "Exiting the ATM. Thank you!";
                break;

            default:
                cout << "Invalid choice. Please enter a number
between 1 and 4.";
        }

    } while (choice != 4);

    return 0;
}
```

Output: (screenshot)

Test Case: Any two (screenshot)

Case1:

```
cd "/Users/yashikathakur/Desktop/c++20 prob/" && g++ atm.cpp -o atm && "/Users/ya
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd "/Users/yashikathakur/Desktop/
"atm
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 2
Enter the amount to deposit: 7000
Deposit successful. Your new balance is: 8000
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 1
Your balance is: 8000
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 3
Enter the amount to withdraw: 5005
Withdrawal successful. Your new balance is: 2995
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 4
Exiting the ATM. Thank you!%
○ yashikathakur@Yashikas-MacBook-Air c++20 prob % █
```

Case2:

```
cd "/Users/yashikathakur/Desktop/c++20 prob/" && g++ atm.cpp -o atm && "/Users/yas
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd "/Users/yashikathakur/Desktop/d
"atm
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 1
Your balance is: 1000
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 3
Enter the amount to withdraw: 100
Withdrawal successful. Your new balance is: 900
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 2
Enter the amount to deposit: 9092
Deposit successful. Your new balance is: 9992
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 4
Exiting the ATM. Thank you!%
○ yashikathakur@Yashikas-MacBook-Air c++20 prob % █
```

Conclusion:

This program defines an ATM class with methods for checking balance, depositing, and withdrawing money. The main function uses a do-while loop to display a menu, take user input, and execute the selected operation using a switch statement. The loop continues until the user chooses to exit.

Experiment No: 5

Title:

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

Theory:

- The program defines a function `findLargest` that takes three integers and returns the largest among them using nested if-else statements.
- The main function prompts the user to input three numbers and calls the `findLargest` function to determine and display the largest number.

Code:

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

int main()
{
    double num1, num2, num3;

    // Input three numbers from the user
    cout << "Enter three numbers: ";
    cin >> num1 >> num2 >> num3;

    // Nested if-else statements to find the largest number
    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)

Test Case: Any two (screenshot)

Case1:

Case2:

Conclusion:

The nested if-else statements provide a structured way to compare three numbers and find the largest. This approach helps handle various cases of input and ensures the correct determination of the maximum value. The program can be easily extended to handle more numbers by adding additional conditions.

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:

The program defines a function `calculateGrade` that takes the average marks of 5 subjects and returns the corresponding grade using an if-else-if ladder.   The main function prompts the user to input marks for 5 subjects, calculates the average, and calls the `calculateGrade` function to determine and display the grade.

Code:

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

int main() {
    string name;
    // Input marks for 5 subjects
    double sub1, sub2, sub3, sub4, sub5;


    cout<<"Please enter Your Name:";
    cin>>name;
    cout << "Enter marks for 5 subjects:";
    cin >> sub1 >> sub2 >> sub3 >> sub4 >> sub5;

    // Calculate the average marks
    double averageMarks = (sub1 + sub2 + sub3 + sub4 + sub5) / 5;

    // Determine the grade based on the average marks using if-
else-if ladder

    if (averageMarks >= 90)
    {
        cout<<name<<" "<< "You have grade: A";
    }
    else if (averageMarks >= 80)
    {
        cout<<name<<" "<< "You have grade: B";
    }
    else if (averageMarks >= 70)
    {
        cout<<name<<" "<< "You have grade: C";
    }
    else if (averageMarks >= 60)
    {
        cout<<name<<" "<< "You have grade: D";
    }
    else
    {
        cout<<name<<" "<< "You have grade: F";
    }
```

```
    return 0;
}
```

Output: (screenshot)


```
cd "/Users/yashikathakur/Desktop/c++20 prob/" && g++
e
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd "
hikathakur/Desktop/c++20 prob/"tempCodeRunnerFile
Please enter Your Name:Yashika
Enter marks for 5 subjects:56
78
90
88
78
Yashika You have grade: C%
○ yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Test Case: Any two (screenshot)

Case1:


```
e
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd
hikathakur/Desktop/c++20 prob/"tempCodeRunnerFile
Please enter Your Name:sakshi
Enter marks for 5 subjects:67
90
90
89
88
sakshi You have grade: B%
○ yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Case2:

Conclusion:

The if-else-if ladder provides a systematic way to evaluate the average marks and assign a grade based on predefined ranges. This structure allows for easy modification or extension of grade criteria. The program ensures clarity and efficiency in determining a student's grade.

Experiment No: 7

Title:

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

Theory:

- The program defines a function `sumOfDigits` that takes an integer and iteratively adds its digits until a single-digit sum is obtained.

- The main function prompts the user to input a number, calls the sumOfDigits function, and displays the final sum.

Code:

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

int main()
{
    int number;

    // Input a number from the user
    cout << "Enter a number: ";
    cin >> number;

    // Keep finding the sum of digits until it becomes a single-digit number
    while (number >= 10) {
        int sum = 0;

        // Calculate the sum of digits
        while (number > 0) {
            sum += number % 10;
            number /= 10;
        }

        number = sum;
    }

    // Display the final result
    cout << "The sum of digits until it becomes a single-digit number: " << number <<endl;

    return 0;
}
```

Output: (screenshot)

```
cd "/Users/yashikathakur/Desktop/c++20 prob/" && g++ tempCodeR
e
yashikathakur@Yashikas-MacBook-Air c++20 prob % cd "/Users/yas
hikathakur/Desktop/c++20 prob/"tempCodeRunnerFile
Enter a number: 678
The sum of digits until it becomes a single-digit number: 3
yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Test Case: Any two (screenshot)

Case1:


```
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd "/Users/yashi
  hikathakur/Desktop/c++20 prob/"tempCodeRunnerFile
  Enter a number: 578
  The sum of digits until it becomes a single-digit number: 2
○ yashikathakur@Yashikas-MacBook-Air c++20 prob % ▊
```

Case2:


```
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd "/Users/yas
  hikathakur/Desktop/c++20 prob/"tempCodeRunnerFile
  Enter a number: 789
  The sum of digits until it becomes a single-digit number: 6
○ yashikathakur@Yashikas-MacBook-Air c++20 prob % ▊
```

Conclusion:

By using a loop, this program efficiently calculates the sum of digits until a single-digit number is obtained. The approach ensures clarity and flexibility for handling various input numbers. It provides a concise solution for repetitive digit sum calculations.

Experiment No: 8

Title:

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

Theory:

- The program defines a function `calculateCoefficient` to calculate binomial coefficients using recursion.
- Another function `printPascalsTriangle` uses nested loops to generate and display the values of each row in Pascal's Triangle.
- The main function prompts the user to input the number of rows, calls the `printPascalsTriangle` function, and displays the triangle.

Code:

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

int main() {
    int numRows;

    // Input: Number of rows in Pascal's Triangle
    cout << "Enter the number of rows for Pascal's Triangle: ";
    cin >> numRows;

    // Printing Pascal's Triangle
    for (int i = 0; i < numRows; i++) {
        // Adding spaces to center the triangle
        for (int j = 0; j < numRows - i - 1; j++) {
            cout << " ";
        }

        int coeff = 1;
        for (int j = 0; j <= i; j++) {
            // Calculate and print the binomial coefficient for each position
            cout << coeff << " ";

            // Update the binomial coefficient for the next position
            coeff = coeff * (i - j) / (j + 1);
        }

        cout << endl;
    }

    return 0;
}
```

Output: (screenshot)

```
cd "/Users/yashikathakur/Desktop/c++20 prob/" && g++
yashikathakur@Yashikas-MacBook-Air c++20 prob % cd "/
ktop/c++20 prob/"pascals_tri
Enter the number of rows for Pascal's Triangle: 3
  1
 1 1
1 2 1
yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Test Case: Any two (screenshot)

Case1:

```
cd "/Users/yashikathakur/Desktop/c++20 prob/" && g+
yashikathakur@Yashikas-MacBook-Air c++20 prob % cd
ktop/c++20 prob/"pascals_tri
Enter the number of rows for Pascal's Triangle: 8
       1
      1 1
     1 2 1
    1 3 3 1
   1 4 6 4 1
  1 5 10 10 5 1
 1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Case2:

```
cd "/Users/yashikathakur/Desktop/c++20 prob/" && g-
yashikathakur@Yashikas-MacBook-Air c++20 prob % cd
ktop/c++20 prob/"pascals_tri
Enter the number of rows for Pascal's Triangle: 6
     1
    1 1
   1 2 1
  1 3 3 1
 1 4 6 4 1
1 5 10 10 5 1
yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Conclusion:

By utilizing nested loops and a recursive function for binomial coefficients, this program efficiently generates and prints Pascal's Triangle. The approach ensures a clear and concise solution for visualizing the triangular array of coefficients. The flexibility of the functions allows the user to easily customize the number of rows to be displayed.

Experiment No: 9

Title:

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

Theory:

- The program defines a function `calculateFactorial` to calculate the factorial of a given number using a loop.
- Another function `calculateSeriesSum` uses nested loops to calculate the sum of the series up to a specified number of terms.
- The main function prompts the user to input the number of terms, calls the `calculateSeriesSum` function, and displays the sum.

Code:

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

int main() {
    int N;

    // Input: Number of terms in the series
    cout << "Enter the value of N: ";
    cin >> N;

    double sum = 0;
    double term = 1;

    // Calculating the sum of the series
```

```cpp
    for (int i = 1; i <= N; i++) {
        term *= i;   // Calculate factorial
        sum += i / term;
    }

    // Displaying the result
    cout << "Sum of the series: " << sum << endl;

    return 0;
}
```

Output: (screenshot)

```
 cd "/Users/yashikathakur/Desktop/c++20 prob/" && g
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd
 /Desktop/c++20 prob/"sum_of_series
 Enter the value of N: 7
 Sum of the series: 2.71806
○ yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Test Case: Any two (screenshot)

Case1:

```
● yashikathakur@Yashikas-MacBook-Air c++20 prob % c
 /Desktop/c++20 prob/"sum_of_series
 Enter the value of N: 6
 Sum of the series: 2.71667
○ yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Case2

```
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd
 /Desktop/c++20 prob/"sum_of_series
 Enter the value of N: 4
 Sum of the series: 2.66667
○ yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Conclusion:

This program efficiently calculates the sum of the series by employing nested loops for both factorial calculation and series summation. The iterative approach provides a clear and straightforward solution for handling series involving factorials. The user can easily customize the number of terms in the series for calculation.

Experiment No: 10

Title:

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

Theory:

- The program utilizes the C++ standard library's `<algorithm>` header for the `std::sort` function, which is used to sort the array of strings in alphabetical order.
- The array of strings is input by the user, and the `std::sort` function is applied to rearrange them alphabetically.
- The sorted array is then displayed.

Code:

```cpp
#include <iostream>
#include <algorithm>   // For using the sort function
using namespace std;

int main()
{
    const int size = 5;   // Change the size based on the number of
strings you want to input
    string strings[size];

    // Input: Taking strings from the user
    cout << "Enter " << size << " strings:\n";
    for (int i = 0; i < size;i++)
    {
        cout << "String " << i + 1 << ": ";
        cin >> strings[i];
    }
```
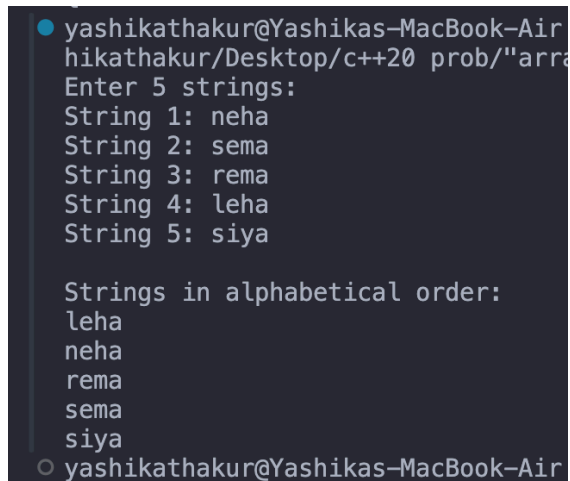
```
    // Sorting the array of strings in alphabetical order
    sort(strings, strings + size);

    // Displaying the strings in alphabetical order
    cout << "\nStrings in alphabetical order:\n";
    for (int i = 0; i < size; i++)
    {
        cout << strings[i] << endl;
    }

    return 0;
}
```

Output: (screenshot)

Test Case: Any two (screenshot)

Case1:

Case2:

Conclusion:

By leveraging the `std::sort` function from the C++ standard library, this program efficiently sorts an array of strings in alphabetical order. The approach ensures a concise and straightforward solution for arranging and displaying strings. Users can easily customize the array size and input values for different scenarios.

Experiment No: 11

Title:

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

Theory:

- The program defines an array and takes user input for its elements.
- It then iterates through the array to check if each element is greater than or equal to its predecessor, indicating ascending order.
- If the loop completes without breaking, the program concludes that the array is sorted in ascending order.

Code:
```cpp
#include <iostream>

using namespace std;

int main() {
```

```cpp
    const int size = 5;   // Change the size based on the size of
your array
    int array[size];

    // Input: Taking array elements from the user
    cout << "Enter " << size << " integers:\n";
    for (int i = 0; i < size; ++i) {
        cout << "Element " << i + 1 << ": ";
        cin >> array[i];
    }

    // Checking if the array is sorted in ascending order
    int i;
    for (i = 1; i < size; ++i) {
        if (array[i] < array[i - 1]) {
            cout << "The array is not sorted in ascending order.
\n";
            break;
        }
    }

    // If the loop completed without breaking, the array is sorted
    if (i == size) {
        cout << "The array is sorted in ascending order.\n";
    }

    return 0;
}
```

Output: (screenshot)

```
cd "/Users/yashikathakur/Desktop/c++20 prob/" && g
  "array_ascending
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd
  ing && "/Users/yashikathakur/Desktop/c++20 prob/"a
  Enter 5 integers:
  Element 1: 8
  Element 2: 6
  Element 3: 7
  Element 4: 4
  Element 5: 9
  The array is not sorted in ascending order.
○ yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Test Case: Any two (screenshot)

Case1:


```
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd
  ing && "/Users/yashikathakur/Desktop/c++20 prob/"ar
  Enter 5 integers:
  Element 1: 2
  Element 2: 3
  Element 3: 4
  Element 4: 5
  Element 5: 6
  The array is sorted in ascending order.
○ yashikathakur@Yashikas-MacBook-Air c++20 prob % ▊
```

Case2:


```
        array_ascending
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd ",
  ing && "/Users/yashikathakur/Desktop/c++20 prob/"arr
  Enter 5 integers:
  Element 1: 6
  Element 2: 8
  Element 3: 10
  Element 4: 12
  Element 5: 14
  The array is sorted in ascending order.
○ yashikathakur@Yashikas-MacBook-Air c++20 prob % ▊
```

Conclusion:

This program efficiently determines if an array is sorted in ascending order by comparing consecutive elements. The simplicity of the approach allows for quick identification of sorted arrays. Users can easily adjust the array size and input values for different scenarios.

Experiment No: 12

Title:

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

Theory:

- The program defines a 2D matrix and takes user input for its elements, row-wise.
- It then uses nested loops to calculate the sum of elements in each row, storing the result in a separate variable for each row.
- The program displays the sum of elements in each row.

Code:

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

int main() {
    const int rows = 3;   // Change the number of rows based on your matrix
    const int cols = 3;   // Change the number of columns based on your matrix
    int matrix[rows][cols];

    // Input: Taking matrix elements from the user
    cout << "Enter matrix elements row-wise:\n";
    for (int i = 0; i < rows; i++) {
        for (int j = 0; j < cols; j++) {
            cout << "Element at position (" << i + 1 << ", " << j + 1 << "): ";
            cin >> matrix[i][j];
        }
    }

    // Calculating the sum of elements in each row
    cout << "\nSum of elements in each row:\n";
    for (int i = 0; i < rows; i++) {
        int rowSum = 0;
        for (int j = 0; j < cols; j++) {
            rowSum += matrix[i][j];
        }
        cout << "Row " << i + 1 << ": " << rowSum << endl;
    }

    return 0;
}
```

Output: (screenshot)


```
e_rowofmatrix
yashikathakur@Yashikas-MacBook-Air c++20
f_ele_rowofmatrix && "/Users/yashikathak
Enter matrix elements row-wise:
Element at position (1, 1): 2
Element at position (1, 2): 4
Element at position (1, 3): 6
Element at position (2, 1): 7
Element at position (2, 2): 8
Element at position (2, 3): 9
Element at position (3, 1): 3
Element at position (3, 2): 2
Element at position (3, 3): 4

Sum of elements in each row:
Row 1: 12
Row 2: 24
Row 3: 9
yashikathakur@Yashikas-MacBook-Air c++20
```

Test Case: Any two (screenshot)

Case1:


```
e_rowofmatrix
yashikathakur@Yashikas-MacBook-Air c++20 pr
sers/yashikathakur/Desktop/c++20 prob/"sum_
Enter matrix elements row-wise:
Element at position (1, 1): 7
Element at position (1, 2): 3
Element at position (1, 3): 4
Element at position (2, 1): 2
Element at position (2, 2):
5
Element at position (2, 3): 7
Element at position (3, 1): 5
Element at position (3, 2): 4
Element at position (3, 3): 7

Sum of elements in each row:
Row 1: 14
Row 2: 14
Row 3: 16
yashikathakur@Yashikas-MacBook-Air c++20 pr
```

Case2:


```
e_rowofmatrix
yashikathakur@Yashikas-MacBook-Air
sers/yashikathakur/Desktop/c++20 pr
Enter matrix elements row-wise:
Element at position (1, 1): 4
Element at position (1, 2): 5
Element at position (1, 3): 3
Element at position (2, 1): 2
Element at position (2, 2): 5
Element at position (2, 3): 6
Element at position (3, 1): 3
Element at position (3, 2): 6
Element at position (3, 3): 7

Sum of elements in each row:
Row 1: 12
Row 2: 13
Row 3: 16
yashikathakur@Yashikas-MacBook-Air
```

Conclusion:

This program efficiently computes the sum of elements in each row of a matrix using nested loops. The approach ensures clarity and flexibility for handling various matrix sizes. Users can easily customize the number of rows and columns, making it adaptable to different scenarios.

Experiment No: 13

Title:

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

Theory:

- The program takes a string input from the user.
- It uses the `std::next_permutation` function to obtain permutations of the sorted string.
- The sorted string is displayed as the initial permutation, and subsequent permutations are generated and printed until all possible permutations are covered.

Code:

```cpp
#include <iostream>
#include <algorithm>   // For using std::next_permutation
using namespace std;

int main() {
    string str;

    // Input: Taking a string from the user
    cout << "Enter a string: ";
    cin >> str;

    // Sorting the string to get the initial permutation
    sort(str.begin(), str.end());

    // Displaying the initial permutation
    cout << "\nAll permutations of the string:\n";
    do
    {
        cout << str << endl;
    } while (next_permutation(str.begin(), str.end()));
```

```
    return 0;
}
```

Output: (screenshot)

```
All permutations of the string:
achmt
achtm
acmht
acmth
acthm
actmh
ahcmt
ahctm
ahmct
ahmtc
ahtcm
ahtmc
amcht
amcth
amhct
```

Test Case: Any two (screenshot)

Case1:

```
All permutations of the string:
ginr
girn
gnir
gnri
grin
grni
ignr
igrn
ingr
inrg
irgn
irng
ngir
ngri
nigr
```

Case2:

```
++20 prob/ generate_permutation
yashikathakur@Yashikas-MacBook-A
e_permutation && "/Users/yashika
Enter a string: sin

All permutations of the string:
ins
isn
nis
nsi
sin
sni
yashikathakur@Yashikas-MacBook-A
```

Conclusion:

This program effectively demonstrates the generation of all permutations of a string using the C++ standard library's `std::next_permutation` function. The sorted string ensures an organized start for generating permutations. Users can easily input different strings to observe the permutations generated by the program.

Experiment No: 14

Title:

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

```
*****
*   *
*   *
*   *
*****
```

Theory:

- The program uses nested loops to traverse through rows and columns.
- '*' characters are printed for the first and last rows and the first and last columns, creating a hollow square pattern.
- Spaces are printed for other positions within the square.

Code:

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

int main() {
    const int size = 5;   // Change the size based on the desired pattern size

    // Printing the pattern
    for (int i = 1; i <= size; ++i) {
        for (int j = 1; j <= size; ++j) {
            // Printing '*' for the first and last rows, and the first and last columns
            if (i == 1 || i == size || j == 1 || j == size) {
```

```
                cout << "*";
            } else {
                // Printing space for other positions
                cout << "";
            }
        }
        cout << endl;
    }

    return 0;
}
```

Output: (screenshot)

```
● yashikathakur@
  pp -o pattern1
  *****
  **
  **
  **
  *****
○ yashikathakur@
```

Conclusion:

The program efficiently prints a hollow square pattern using nested loops, demonstrating a simple approach for generating such patterns. Users can easily modify the `size` variable to control the dimensions of the square. The program provides a clear and concise solution for printing hollow squares of varying sizes.

Experiment No: 15

Title:

Write a C++ program to display the following pattern:
1
232
34543
4567654
34543
232

Theory:

- The program uses two nested loops to traverse through rows and columns.
- It determines whether to print increasing or decreasing numbers based on the current row position.
- Numbers are printed in a pyramid pattern with appropriate spacing.

Code:

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

int main()
{
    const int size = 4;    // Change the size based on the desired pattern size

    // Printing the pattern
    for (int i = 1; i <= size * 2 - 1; ++i)
    {
        int num;

        if (i <= size)
        {
            // Printing increasing numbers for the first half
            num = i;
        } else
        {
            // Printing decreasing numbers for the second half
            num = 2 * size - i;
        }

        // Printing the numbers with appropriate spacing
        for (int j = 1; j <= num; ++j)
        {
            cout << j;
        }

        for (int j = num - 1; j >= 1; --j)
        {
            cout << j;
        }

        cout << endl;
    }

    return 0;
```
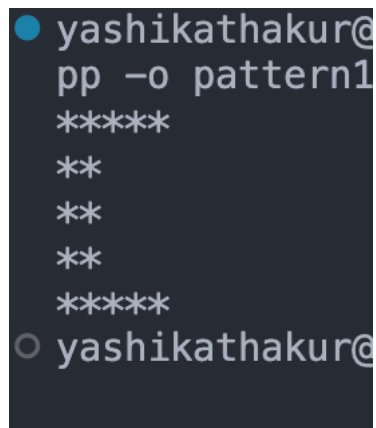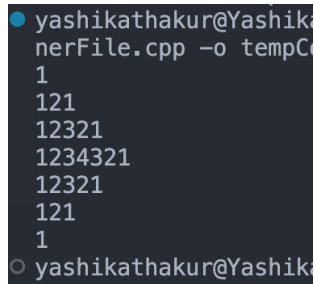
Output: (screenshot)

```
yashikathakur@Yashika
nerFile.cpp -o tempC
1
121
12321
1234321
12321
121
1
yashikathakur@Yashika
```

Conclusion:

The program efficiently prints a pyramid pattern with numbers using nested loops, showcasing an approach to generate visually appealing patterns. Users can easily modify the size variable to control the dimensions of the pyramid. The program provides a clear and concise solution for printing pyramid patterns with varying sizes.

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++. Yourprogram 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 toinitialize the attributes when a new product is added to the inventory.

Theory:

- The program defines a Product class with private attributes (productId, productName, price, quantityInStock) and public methods to display information, get quantity, and update quantity.

- Two `Product` objects are created using the parameterized constructor, and their initial information is displayed.
- The program simulates a sale by updating the quantity of one product and displays the updated information.

Code:

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

class Product {
private:
    int productId;
    string productName;
    float price;
    int quantityInStock;

public:
    // Parameterized constructor to initialize attributes
    Product(int id, const string& name, float pr, int qty)
        : productId(id), productName(name), price(pr),
quantityInStock(qty)
        {}

    // Display product information
    void displayProductInfo() const
    {
        cout << "Product ID: " << productId << "\n"
             << "Product Name: " << productName << "\n"
             << "Price: $" << price << "\n"
             << "Quantity in Stock: " << quantityInStock <<
"\n\n";
    }

    // Getter for product quantity
    int getQuantity() const
    {
        return quantityInStock;
    }

    // Update product quantity after a sale or restocking
    void updateQuantity(int newQuantity)
    {
        quantityInStock = newQuantity;
    }
};
```

```cpp
int main()
{
    // Creating Product objects using the parameterized constructor
    Product product1(1, "Laptop", 899.99, 10);
    Product product2(2, "Smartphone", 499.99, 15);

    // Displaying initial product information
    cout << "Initial Product Information:\n";
    product1.displayProductInfo();
    product2.displayProductInfo();

    // Simulating a sale and updating quantity
    int soldQuantity = 3;
    int newQuantity = product1.getQuantity() - soldQuantity;
    product1.updateQuantity(newQuantity);

    // Displaying updated product information
    cout << "Product Information after Sale:\n";
    product1.displayProductInfo();

    return 0;
}
```
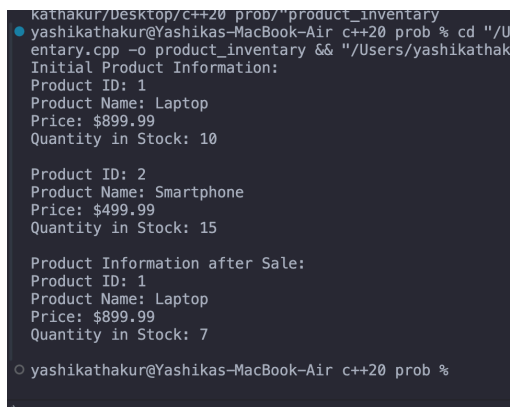
Output: (screenshot)


```
kathakur/Desktop/c++20 prob/"product_inventory
● yashikathakur@Yashikas-MacBook-Air c++20 prob % cd "/U
entary.cpp -o product_inventory && "/Users/yashikathak
Initial Product Information:
Product ID: 1
Product Name: Laptop
Price: $899.99
Quantity in Stock: 10

Product ID: 2
Product Name: Smartphone
Price: $499.99
Quantity in Stock: 15

Product Information after Sale:
Product ID: 1
Product Name: Laptop
Price: $899.99
Quantity in Stock: 7

○ yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Conclusion:

This program illustrates the basic principles of object-oriented programming by encapsulating product information and operations within a class. It provides a foundation for more complex inventory management systems. Users can extend this model by adding additional features and methods for a comprehensive inventory management solution.

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:

- The Student class has private attributes (name, rollNumber, marks) and methods to display details, get/set name, roll number, and marks.
- The StudentRecordSystem class contains a vector of Student objects and methods to add students, display all students, and calculate/display the average marks.
- In the main function, an instance of StudentRecordSystem is created, and sample students are added. Details and average marks are displayed.

Code:

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

class Student {
private:
    string name;
    int rollNumber;
    float marks;

public:

    Student(const string& studentName, int studentRollNumber, float studentMarks)
        : name(studentName), rollNumber(studentRollNumber), marks(studentMarks) {}

    void displayStudentDetails() const {
        cout << "Name: " << name << endl;
        cout << "Roll Number: " << rollNumber << endl;
        cout << "Marks: " << marks << endl;
        cout << "-------------------------\n";
```

```cpp
    }

    const string& getName() const { return name; }
    void setName(const string& studentName) { name = studentName; }

    int getRollNumber() const { return rollNumber; }
    void setRollNumber(int studentRollNumber) { rollNumber =
studentRollNumber; }

    float getMarks() const { return marks; }
    void setMarks(float studentMarks) { marks = studentMarks; }
};

class StudentRecordSystem {
private:
    vector<Student> studentRecords;

public:

    void addStudent() {
        string name;
        int rollNumber;
        float marks;

        cout << "Enter student details:\n";
        cout << "Name: ";
        cin.ignore();
        getline(cin, name);

        cout << "Roll Number: ";
        cin >> rollNumber;

        cout << "Marks: ";
        cin >> marks;

        studentRecords.push_back(Student(name, rollNumber, marks));
        cout << "Student added successfully!\n";
    }

    void displayAllStudents() const {
        if (studentRecords.empty()) {
            cout << "No student records available.\n";
        } else {
            cout << "Student Details:\n";
            for (const auto& student : studentRecords) {
                student.displayStudentDetails();
            }
        }
    }
```

```cpp
    void calculateAndDisplayAverageMarks() const {
        if (studentRecords.empty()) {
            cout << "No student records available.\n";
        } else {
            float totalMarks = 0;
            for (const auto& student : studentRecords) {
                totalMarks += student.getMarks();
            }

            float averageMarks = totalMarks /
studentRecords.size();
            cout << "Average Marks of All Students: " <<
averageMarks << std::endl;
        }
    }
};

int main() {

    StudentRecordSystem recordSystem;

    recordSystem.addStudent();
    recordSystem.addStudent();
    recordSystem.addStudent();

    recordSystem.displayAllStudents();
    recordSystem.calculateAndDisplayAverageMarks();

    return 0;
}
```

Output: (screenshot)

```
Enter student details:
Name: yashika
Roll Number: 13
Marks: 54
Student added successfully!
Enter student details:
Name: sakshi
Roll Number: 78
Marks: 67
Student added successfully!
Enter student details:
Name: arpita
Roll Number: 45
Marks: 90
Student added successfully!
Student Details:
Name: ashika
Roll Number: 13
Marks: 54
------------------------
Name: sakshi
Roll Number: 78
Marks: 67
------------------------
Name: arpita
Roll Number: 45
Marks: 90
------------------------
Average Marks of All Students: 70.3333
○ yashikathakur@Yashikas-MacBook-Air c++20 prob %
master*+ ↺   ⊗ 0 ⚠ 0   🕊 0
```

Conclusion:

This program demonstrates a basic Student Record System using object-oriented principles. It showcases encapsulation, data abstraction, and vector usage for managing student records. Users can expand this system by adding more features such as searching, sorting, or additional student details. The code provides a foundation for building more advanced student management systems.

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:

- The `Calculator` class encapsulates basic arithmetic operations (addition, subtraction, multiplication, division).
- In the `main` function, an instance of the `Calculator` class is created.
- The user inputs two numbers and selects an operation from the menu.
- The selected operation is performed using the appropriate method from the `Calculator` class.

Code:

```cpp
#include <iostream>

class Calculator {
public:
    // Method to perform addition
    float add(float num1, float num2) {
        return num1 + num2;
    }

    // Method to perform subtraction
    float subtract(float num1, float num2) {
        return num1 - num2;
    }

    // Method to perform multiplication
```

```cpp
    float multiply(float num1, float num2) {
        return num1 * num2;
    }

    // Method to perform division
    float divide(float num1, float num2) {
        if (num2 != 0) {
            return num1 / num2;
        } else {
            std::cerr << "Error: Division by zero is not allowed.\n";
            return 0;
        }
    }
};

int main() {
    // Create an instance of the Calculator class
    Calculator calculator;

    // Input two numbers from the user
    float num1, num2;
    std::cout << "Enter the first number: ";
    std::cin >> num1;
    std::cout << "Enter the second number: ";
    std::cin >> num2;

    // Display menu for operation selection
    std::cout << "Select operation:\n";
    std::cout << "1. Addition\n";
    std::cout << "2. Subtraction\n";
    std::cout << "3. Multiplication\n";
    std::cout << "4. Division\n";

    int choice;
    std::cout << "Enter your choice (1-4): ";
    std::cin >> choice;

    // Perform the selected operation
    switch (choice) {
        case 1:
            std::cout << "Result: " << calculator.add(num1, num2) << "\n";
            break;
        case 2:
            std::cout << "Result: " << calculator.subtract(num1, num2) << "\n";
            break;
        case 3:
```

```cpp
            std::cout << "Result: " << calculator.multiply(num1,
num2) << "\n";
            break;
        case 4:
            std::cout << "Result: " << calculator.divide(num1,
num2) << "\n";
            break;
        default:
            std::cerr << "Error: Invalid choice.\n";
            break;
    }

    return 0;
}
```
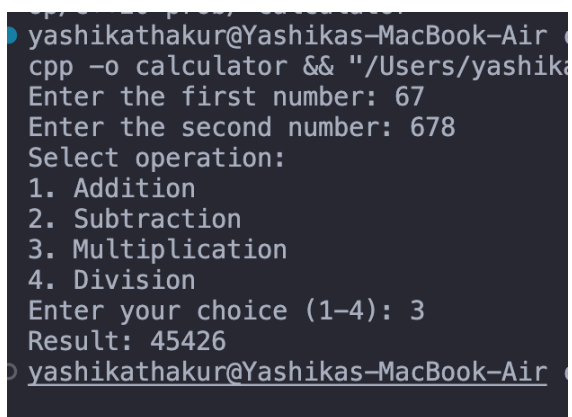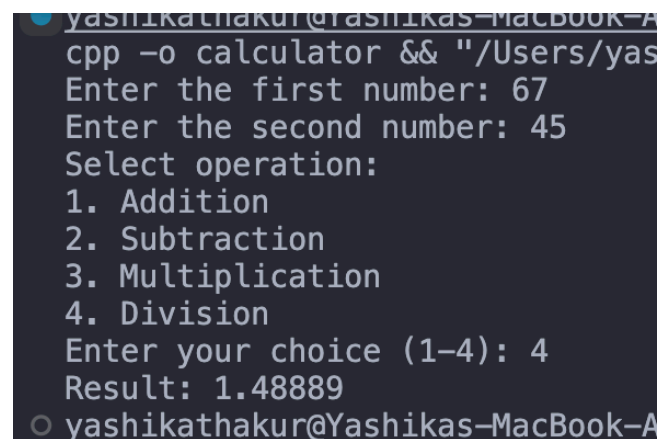
Output: (screenshot)

Test Case: Any two (screenshot)

Case1:

Case2:

```
yashikathakur@yashikas-MacBook-Air
cpp -o calculator && "/Users/yashik
Enter the first number: 56
Enter the second number: 45
Select operation:
1. Addition
2. Subtraction
3. Multiplication
4. Division
Enter your choice (1-4): 2
Result: 11
yashikathakur@Yashikas-MacBook-Air
```

Conclusion:

This program showcases the use of a class (Calculator) to structure and encapsulate functionality related to arithmetic operations. It provides a simple command-line interface for users to perform basic calculations. The switch statement efficiently handles the user's choice for the operation. Users can easily extend the Calculator class to include more operations or features.

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:

The provided C++ code represents a simple console-based food ordering system. The system is designed to handle three categories of food items: Desserts, Appetizers, and Main Courses. Each category is represented by a corresponding class (Dessert, Appetizer, and MainCourse), which inherits from the base class FoodItem.

The FoodItem class contains attributes such as name, price, and description, while the derived classes (Dessert, Appetizer, and MainCourse) include methods to display the menu and add items to the cart.

The foodMenu() function serves as the main driver for the food ordering process. It continuously prompts the user to choose a category, select items from the menu, and adds them to the cart. The user can continue adding items or choose to proceed to checkout. The program calculates the total bill, including a tax of 18%, and prompts the user to select a payment method (Online or Cash).

Code:

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

class FoodItem {
public:
    string name;
    int price;
    string description;
};

class Dessert : public FoodItem {
public:
    int showMenu() {
        int choice;
        while (true) {
            try {
                cout << "\n1. Vanilla Ice Cream (INR 80/pc)\n2. Choco lava cake (INR 120/pc)\n3. Ice cream Shake (INR 60/pc)\n";
                cin >> choice;
                if (1 <= choice && choice <= 3) {
                    return choice;
                } else {
                    cout << "Wrong input. Please try again.\n";
                }
            } catch (...) {
                cout << "Invalid input. Please enter a number.\n";
            }
        }
    }

    void cart(int choice) {
        if (choice == 1) {
            name = "Vanilla Ice Cream";
```

```cpp
            price = 80;
            description = "Made with Milk and essence of vanilla";
        } else if (choice == 2) {
            name = "Choco Lava Cake";
            price = 120;
            description = "A Chocolate based cake with melted
chocolate filling";
        } else if (choice == 3) {
            name = "Ice Cream Shake";
            price = 60;
            description = "Milk Shake with ice cream topped on it,
a mouthwatering edible shake";
        }
    }
};

class Appetizer : public FoodItem {
public:
    int showMenu() {
        int choice;
        while (true) {
            try {
                cout << "\n1. Potato Wedges (INR 120/plate)\n2.
French Fries (INR 100/plate)\n3. Paneer Chilly (INR 180/plate)\n";
                cin >> choice;
                if (1 <= choice && choice <= 3) {
                    return choice;
                } else {
                    cout << "Wrong input. Please try again.\n";
                }
            } catch (...) {
                cout << "Invalid input. Please enter a number.\n";
            }
        }
    }
    void cart(int choice) {
        if (choice == 1) {
            name = "Potato Wedges";
            price = 120;
            description = "Fried Potato with wrinkled shape and
seasoned with salt and pepper";
        } else if (choice == 2) {
            name = "French Fries";
            price = 100;
            description = "Deep Fried Potato sticks with Peri Peri
seasonings";
        } else if (choice == 3) {
            name = "Paneer Chilly";
            price = 180;
```

```cpp
                description = "Paneer Seasoned with veggies and sauce
and Saute with golden crisp";
        }
    }
};

class MainCourse : public FoodItem {
public:
    int showMenu() {
        int choice;
        while (true) {
            try {
                cout << "\n1. Veg Plate Thali (INR 230/Thali)\n2.
Non Veg Plate Thali (INR 260/Thali)\n3. Special Veg Thali (INR 280/
Thali)\n4. Special Non Veg Thali (INR 300/Thali)\n";
                cin >> choice;
                if (1 <= choice && choice <= 4) {
                    return choice;
                } else {
                    cout << "Wrong input. Please try again.\n";
                }
            } catch (...) {
                cout << "Invalid input. Please enter a number.\n";
            }
        }
    }
}

    void cart(int choice) {
        if (choice == 1) {
            name = "Veg Plate Thali";
            price = 230;
            description = "A Full meal Thali Served with 4 Rotis
and Jeera Rice alongside With salad, paneer, Dal, papad";
        } else if (choice == 2) {
            name = "Non Veg Plate Thali";
            price = 260;
            description = "A Full meal Thali Served with 4 Rotis
and Jeera Rice alongside With salad, Chicken, Dal, papad";
        } else if (choice == 3) {
            name = "Special Veg Thali";
            price = 280;
            description = "A Full meal Thali Served with 4 Rotis
and Jeera Rice alongside With salad, 2 paneer sabzi, veg mix, Dal,
papad";
        } else if (choice == 4) {
            name = "Special Non Veg Thali";
            price = 300;
```

```cpp
            description = "A Full meal Thali Served with 4 Rotis
and Jeera Rice alongside With salad, 2 Chicken sabzis, Mutton, Dal,
papad";
        }
    }
};

void foodMenu() {
    int price = 0;
    string pay, id, y;
    Dessert des;
    Appetizer app;
    MainCourse mncr;
    int i = 1;
    vector<string> l;
    while (true) {
        cout << "\nWELCOME TO FOOD ORDERING SECTION\n\nWHAT IS IT
THAT YOU'D LIKE TO ORDER?";
        cout << "\n1. Dessert\n2. Appetizer\n3. Main Course\n";
        int choice;
        cin >> choice;

        if (choice > 3 || choice < 1) {
            continue;
        }
        if (choice == 1) {
            cout << "\n                    DESSERT                    \n";
            int dc = des.showMenu();
            des.cart(dc);
        } else if (choice == 2) {
            cout << "\n                    APPETIZER
\n";
            int ac = app.showMenu();
            app.cart(ac);
        } else if (choice == 3) {
            cout << "\n                    MAIN MENU
\n";
            int mc = mncr.showMenu();
            mncr.cart(mc);
        }

        if (choice == 1) {
            cout << "\n\nYour Item : ";
            cout << "\n                    " << des.name << "   for INR " <<
des.price << "\n                    " << des.description << endl;
            l.push_back(des.name + "   for INR " +
to_string(des.price) + "                    " + des.description);
            price += des.price;
        } else if (choice == 2) {
```

```cpp
                cout << "Your Item : ";
                cout << "\n              " << app.name << "   for INR " <<
app.price << "\n              " << app.description << endl;
                l.push_back(app.name + "   for INR " +
to_string(app.price) + "              " + app.description);
                price += app.price;
            } else if (choice == 3) {
                cout << "Your Item : \n\n             " << mncr.name << "
for INR " << mncr.price << "\n            " << mncr.description <<
endl;
                l.push_back(mncr.name + "   for INR " +
to_string(mncr.price) + "             " + mncr.description);
                price += mncr.price;
            }

            cout << "\n\nWant to order any other food? (y/n)";
            cin >> y;

            if (y[0] != 'Y' && y[0] != 'y') {
                cout << "\n\nEAT WELL :)\nYOUR BILL : \n\n\n";
                int a = 1;
                for (const auto &item : l) {
                    cout << a << " : " << item << endl;
                    a++;
                }

                cout << "\nTotal Bill = INR " << price + (price * 0.18)
<< " + (TAX) " << price * 0.18 << endl;
                cout << "\nGRAND TOTAL  = INR " << price + (price *
0.18) << endl;
                cout << "\nHow would you like to pay? (Online/Cash)";
                cin >> pay;
                if (pay == "ONLINE" || pay == "CASH") {
                    if (pay == "ONLINE") {
                        cout << "\nEnter Upi id : ";
                        cin >> id;
                        cout << "\nSent the request to pay " << price +
(price * 0.18) << " to your upi id " << id << endl;
                    } else {
                        cout << "\nPlease pay " << price + (price *
0.18) << " with the cash With Change......Thank you :)" << endl;
                    }
                }
                cout << "\nTHANK YOU FOR CHOOSING OUR TREAT. HAVE A
GREAT DINE :)" << endl;
                break;
            }
        }
}
```

```cpp
int main() {
    foodMenu();
    return 0;
}
```

Output: (screenshot)

```
WELCOME TO FOOD ORDERING SECTION

WHAT IS IT THAT YOU'D LIKE TO ORDER?
1. Dessert
2. Appetizer
3. Main Course
3

            MAIN MENU

1. Veg Plate Thali (INR 230/Thali)
2. Non Veg Plate Thali (INR 260/Thali)
3. Special Veg Thali (INR 280/Thali)
4. Special Non Veg Thali (INR 300/Thali)
4
Your Item :

        Special Non Veg Thali  for INR 300
        A Full meal Thali Served with 4 Rotis and Jeera Rice alongside With salad, 2 Chicken sabzis, Mutton, Dal, papad


Want to order any other food? (y/n)n


EAT WELL :)
YOUR BILL :


1 : Special Non Veg Thali  for INR 300           A Full meal Thali Served with 4 Rotis and Jeera Rice alongside With salad, 2 Chicken sabzis, Mutton, Dal, papad

Total Bill = INR 354 + (TAX) 54

GRAND TOTAL  = INR 354

How would you like to pay? (Online/Cash)Cash

THANK YOU FOR CHOOSING OUR TREAT. HAVE A GREAT DINE :)
○ yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Test Case: Any two (screenshot)

Case1:

```
WELCOME TO FOOD ORDERING SECTION

WHAT IS IT THAT YOU'D LIKE TO ORDER?
1. Dessert
2. Appetizer
3. Main Course
1

            DESSERT

1. Vanilla Ice Cream (INR 80/pc)
2. Choco lava cake (INR 120/pc)
3. Ice cream Shake (INR 60/pc)
1


Your Item :
        Vanilla Ice Cream  for INR 80
        Made with Milk and essence of vanilla


Want to order any other food? (y/n)y

WELCOME TO FOOD ORDERING SECTION

WHAT IS IT THAT YOU'D LIKE TO ORDER?
1. Dessert
2. Appetizer
3. Main Course
2

            APPETIZER

1. Potato Wedges (INR 120/plate)
2. French Fries (INR 100/plate)
3. Paneer Chilly (INR 180/plate)
2
Your Item :
        French Fries  for INR 100
```

```
WELCOME TO FOOD ORDERING SECTION

WHAT IS IT THAT YOU'D LIKE TO ORDER?
1. Dessert
2. Appetizer
3. Main Course
2

            APPETIZER

1. Potato Wedges (INR 120/plate)
2. French Fries (INR 100/plate)
3. Paneer Chilly (INR 180/plate)
2
Your Item :
        French Fries  for INR 100
        Deep Fried Potato sticks with Peri Peri seasonings


Want to order any other food? (y/n)n


EAT WELL :)
YOUR BILL :


1 : Vanilla Ice Cream  for INR 80        Made with Milk and essence of vanilla
2 : French Fries  for INR 100         Deep Fried Potato sticks with Peri Peri seasonings

Total Bill = INR 212.4 + (TAX) 32.4

GRAND TOTAL  = INR 212.4

How would you like to pay? (Online/Cash)Cash

THANK YOU FOR CHOOSING OUR TREAT. HAVE A GREAT DINE :)
○ yashikathakur@Yashikas-MacBook-Air c++20 prob %
```

Conclusion:

The code successfully implements a basic food ordering system, demonstrating object-oriented programming concepts such as class inheritance and encapsulation. It provides a structured approach to handle different food categories and allows users to interactively select and order items. The code lacks some error handling and input validation, but overall, it serves as a good starting point for a simple console-based food ordering application. Improvements could be made by refining the input validation, error handling, and adding more features to enhance the overall user experience.

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:

- The Student class has private attributes (name, grades, numGrades) and public methods for adding grades, calculating average grade, and displaying student details.
- The constructor allocates dynamic memory for the grades array, and the destructor deallocates the memory.
- The main function creates a Student object, adds grades, and displays student details.

Code:

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

class Student {
private:
    string name;
    int* grades;
    int numGrades;

public:
    // Constructor
    Student(string studentName, int maxGrades) : name(studentName),
numGrades(0) {
        grades = new int[maxGrades];
    }

    // Destructor
    ~Student() {
        delete[] grades;
    }

    // Method to add grades
    void addGrade(int grade) {
        if (numGrades < 10) { // Assuming a maximum of 10 grades
            grades[numGrades++] = grade;
        } else {
            cout << "Cannot add more than 10 grades.\n";
        }
    }

    // Method to calculate average grade
    float calculateAverageGrade() const {
        if (numGrades == 0) {
            return 0.0;
        }
```

```cpp
        int total = 0;
        for (int i = 0; i < numGrades; ++i) {
            total += grades[i];
        }

        return static_cast<float>(total) / numGrades;
    }

    // Method to display student details and grades
    void displayStudentDetails() const {
        cout << "Student Name: " << name << "\n";
        cout << "Grades: ";
        for (int i = 0; i < numGrades; ++i) {
            cout << grades[i] << " ";
        }
        cout << "\n";
        cout << "Average Grade: " << calculateAverageGrade() <<
"\n";
    }
};

int main() {
    // Create a student object
    Student student1("John Doe", 10);

    // Add grades for the student
    student1.addGrade(85);
    student1.addGrade(90);
    student1.addGrade(78);

    // Display student details
    student1.displayStudentDetails();

    return 0;
}
```

Output: (screenshot)

```
cd /Users/yashikathakur/Desktop/c++20
● yashikathakur@Yashikas-MacBook-Air c++
  ur/Desktop/c++20 prob/"20
  Student Name: John Doe
  Grades: 85 90 78
  Average Grade: 84.3333
○ yashikathakur@Yashikas-MacBook-Air c++
```

Conclusion:

This program demonstrates proper memory management using dynamic memory allocation and deallocation in a class. The constructor initializes the dynamic array, and the destructor ensures the memory is released when the object goes out of scope. Users can extend the program to include additional features such as input validation, more detailed grading information, or handling a larger number of grades.