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

C++ PROGRAMMING LAB

Prepared by:

Name of Student: PREM ANIL THAKARE

Roll No: 02

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++. 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 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: Prem Thakare

Roll Number: 02

Experiment No: 01

Title:

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

Theory:

The program is designed to find the roots of a quadratic equation in the form **ax^2 + bx + c = 0**. The roots are determined based on the discriminant (denoted by d). The discriminant is calculated using the formula d = b^2 - 4ac. The program then proceeds to check the value of the discriminant to determine the nature of the roots.

- If the discriminant is greater than 0, the roots are real and unequal.
- If the discriminant is equal to 0, the roots are real and equal.
- If the discriminant is less than 0, the roots are imaginary.

The roots are calculated using the quadratic formula:

(-b + sqrt(d))/(2a) and (-b - sqrt(d)) / (2a).

Code:

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

double discr(double a,double b,double c,double d){
    double x1,x2;
    d= b*b - 4 * a *c;
    cout<<"Discriminant:"<<d<<endl;
    if(d>1){
        x1= -b + sqrt(d);
        x2= -b - sqrt(d);
        cout<<"Root 1:"<<x1/2*a<<endl;
        cout<<"Root 2:"<<x2/2*a<<endl;
        cout<<"Unequal & Real Roots";
    }
    else if(d==1){
        x1= -b + sqrt(d);
        x2= -b - sqrt(d);
        cout<<"Root 1:"<<x1/2*a<<endl;
        cout<<"Root 2:"<<x2/2*a<<endl;
        cout<<"Unequal & Real Roots";
    }
    else{
        x1= -b + sqrt(abs(d));
        x2= -b - sqrt(abs(d));
        cout<<"Root 1:"<<x1/2*a<<endl;
        cout<<"Root 2:"<<x2/2*a<<endl;
        cout<<"Equal & Imaginary Roots";
    }
    return 0;
    }

int main(){
    double a,b,c,d;
    cout<<"Enter the Quadratic Equation in the Form:\n ax^2+bx+c=0"<<endl;
    cout<<"Enter the Value of a:";
    cin>>a;
    cout<<"Enter the Value of b:";
    cin>>b;
    cout<<"Enter the Value of c:";
    cin>>c;
    discr(a,b,c,d);
    return 0;
}
```

Output (screenshot):

Test Case: Any two (screenshot):

Conclusion:

- The program starts by taking input for the coefficients a, b, and c from the user.
- It then calls the discr function, which calculates the discriminant and determines the roots based on its value.
- The program prints the discriminant and the roots along with a message indicating whether the roots are real and unequal, real and equal, or imaginary.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 02

Title:

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

Theory:

The program is designed to calculate the power of a number using recursion. It defines a **power** function that takes two parameters - **base** and **exponent**. The function uses recursion to calculate the result of raising the **base** to the power of **exponent**. The logic in the function is as follows:

- If **exponent** is 0, the function returns 1.0 since any number raised to the power of 0 is 1.
- If **exponent** is negative, the function recursively calculates the reciprocal of the product of **base** and the power of **base** with the absolute value of **exponent - 1**.
- If **exponent** is positive, the function recursively multiplies **base** by the power of **base** with **exponent - 1**.

The program then takes input for the **base** and **exponent** from the user in the **main** function, calls the **power** function, and prints the result.

Code:

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

double power(double base, int exponent) {
    if (exponent == 0) {
        return 1.0;
    } else if (exponent < 0) {
        return 1.0 / (base * power(base, -exponent - 1));
    } else {
        return base * power(base, exponent - 1);
    }
}

int main() {
    double base;
    int exponent;

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

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

    double result = power(base, exponent);

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

    return 0;
}
```

Output (screenshot):

Test Case: Any two (screenshot):

Conclusion:

- The program uses recursion to calculate the power of a number, providing a clear demonstration of recursive function implementation.
- It handles cases where the exponent is 0 or negative, ensuring the correctness of the calculation.
- The user is prompted to input the base and exponent, making the program interactive and user friendly.
- The program correctly calculates and prints the result of the power operation.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 03

Title:

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

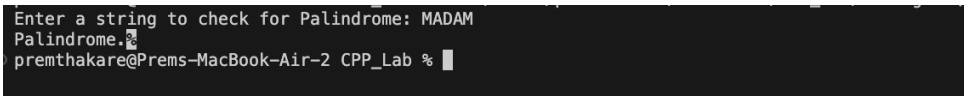
Theory:

The program checks whether a given string is a palindrome or not. A palindrome is a string that reads the same forward as backward. The program takes a string as input, iterates through half of its length, and compares the characters from the beginning and end of the string. If all pairs of corresponding characters match, the string is considered a palindrome.

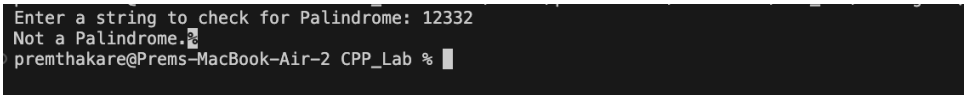
Code:

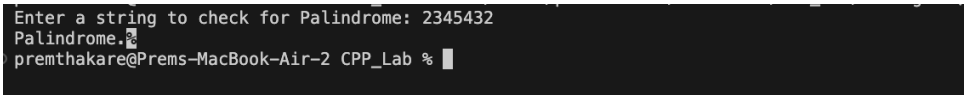
```cpp
#include <iostream>
#include <string>
using namespace std;


int main()
{
    string str;
    int a = 0,n;

    cout << "Enter a string to check for Palindrome: ";
    cin >> str;
    n=str.length();

    for(int i=0; i<n/2 ;i++)
    {
        if(str[i] == str [n - (i+1) ] )
        {
            a++;
        }
    }

    if(a==n/2){
        cout<<"Palindrome.";
    }
    else{
        cout<<"Not a Palindrome.";
        }
    return 0;
}
```

Output (screenshot):



Test Case: Any two (screenshot):





Conclusion:

- The program effectively checks whether the given string is a palindrome using a simple iterative approach.
- It correctly handles strings of both even and odd lengths by iterating through only the first half of the string.
- The use of the **string** class simplifies string handling, and the program is concise and easy to understand.
- The result is printed based on the comparison result, providing a clear indication of whether the input string is a palindrome or not.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 04

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:

The program simulates a simple ATM machine with basic functionalities, allowing users to check their account balance, deposit money, withdraw money, or exit the program. The program uses a **switch** statement to execute different actions based on the user's input.

Code:

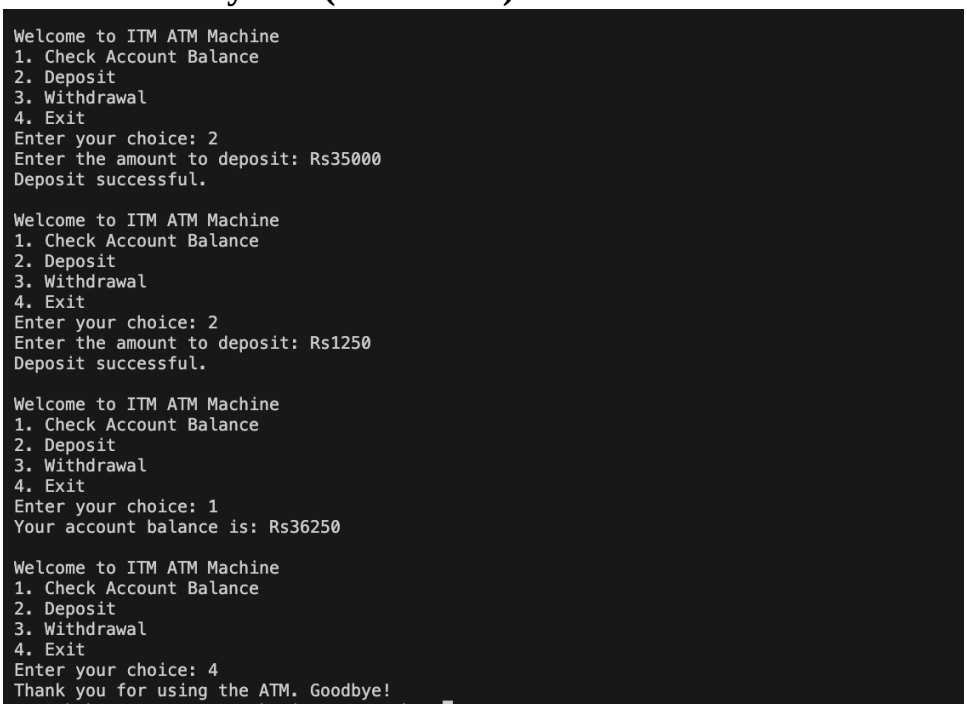
```cpp
#include <iostream>
using namespace std;

double accountBalance = 0;

void displayBalance() {
    cout << "Your account balance is: Rs" << accountBalance << endl;
}

void deposit() {
    double amount;
    cout << "Enter the amount to deposit: Rs";
    cin >> amount;
    if (amount > 0) {
        accountBalance += amount;
        cout << "Deposit successful.\n";
    } else {
        cout << "Invalid amount for deposit.\n";
    }
}

void withdraw() {
    double amount;
    cout << "Enter the amount to withdraw: Rs";
    cin >> amount;
    if (amount > 0 && amount <= accountBalance) {
        accountBalance -= amount;
        cout << "Withdrawal successful.\n";
    } else {
        cout << "Invalid amount for withdrawal or insufficient funds.\n";
    }
}

int main() {
    while (true) {
        cout << "\nWelcome to ITM ATM Machine\n";
        cout << "1. Check Account Balance\n";
        cout << "2. Deposit\n";
        cout << "3. Withdrawal\n";
        cout << "4. Exit\n";
        cout << "Enter your choice: ";

        int choice;
        cin >> choice;

        switch (choice) {
            case 1:
                displayBalance();
                break;
            case 2:
                deposit();
```

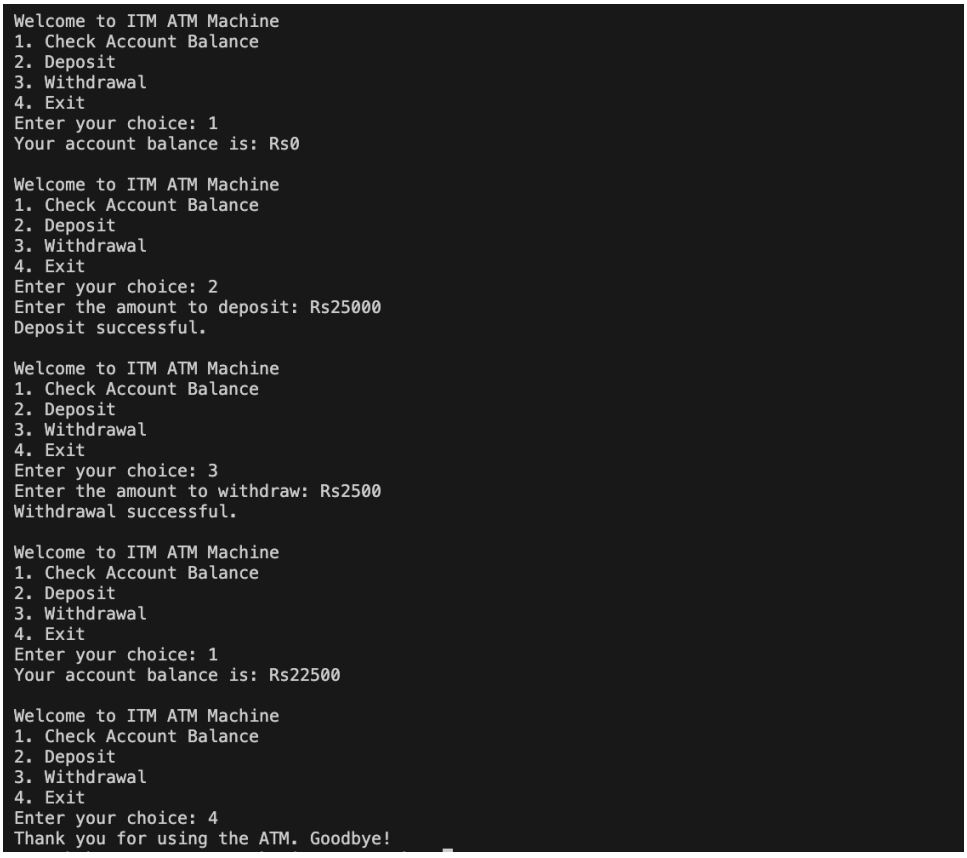
```
            case 2:
                deposit();
                break;
            case 3:
                withdraw();
                break;
            case 4:
                cout << "Thank you for using the ATM. Goodbye!\n";
                return 0;
            default:
                cout << "Invalid choice. Please try again.\n";
        }
    }

    return 0;
}
```

Output (screenshot):

```
Welcome to ITM ATM Machine
1. Check Account Balance
2. Deposit
3. Withdrawal
4. Exit
Enter your choice: 1
Your account balance is: Rs0

Welcome to ITM ATM Machine
1. Check Account Balance
2. Deposit
3. Withdrawal
4. Exit
Enter your choice: 2
Enter the amount to deposit: Rs25000
Deposit successful.

Welcome to ITM ATM Machine
1. Check Account Balance
2. Deposit
3. Withdrawal
4. Exit
Enter your choice: 3
Enter the amount to withdraw: Rs2500
Withdrawal successful.

Welcome to ITM ATM Machine
1. Check Account Balance
2. Deposit
3. Withdrawal
4. Exit
Enter your choice: 1
Your account balance is: Rs22500

Welcome to ITM ATM Machine
1. Check Account Balance
2. Deposit
3. Withdrawal
4. Exit
Enter your choice: 4
Thank you for using the ATM. Goodbye!
```

Test Case: Any two (screenshot):





Conclusion:

- The program provides a simple and interactive ATM interface for users to check their balance, deposit money, and withdraw money.

- It incorporates error handling by checking for invalid amounts during deposit and withdrawal operations.
- The use of functions (**displayBalance**, **deposit**, **withdraw**) enhances code modularity and readability.
- The program is structured using a loop and a **switch** statement, making it easy to navigate and understand.
- The user is continuously prompted with the menu until they choose to exit, ensuring the program remains interactive.
- The program could be further enhanced by incorporating additional features, such as password verification, transaction history, or account creation. Additionally, input validation (checking for non-numeric inputs) could be added for robustness.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 05

Title:

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

Theory:

The program is designed to find the largest among three numbers using nested if-else statements. It takes three numbers as input from the user and uses nested if-else statements to compare them and determine the largest one.

Code:

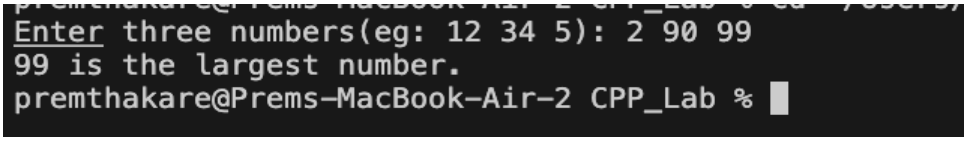
```cpp
#include <iostream>
using namespace std;

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

    cout << "Enter three numbers(eg: 12 34 5): ";
    cin >> num1 >> num2 >> num3;

    if (num1 >= num2) {
        if (num1 >= num3) {
            cout << num1 << " is the largest number." <<endl;
        } else {
            cout << num3 << " is the largest number." <<endl;
        }
    } else {
        if (num2 >= num3) {
            cout << num2 << " is the largest number." <<endl;
        } else {
            cout << num3 << " is the largest number." <<endl;
        }
    }
    return 0;
}
```

Output (screenshot):

```
Enter three numbers(eg: 12 34 5): 34 11 68
68 is the largest number.
```

Test Case: Any two (screenshot):

```
Enter three numbers(eg: 12 34 5): 456 123 23
456 is the largest number.
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```



Conclusion:

- The program effectively determines the largest among three numbers using nested if-else statements.
- The use of nested if-else statements is appropriate for comparing three numbers and handling multiple cases.
- The program provides clear and concise output, indicating the largest number.
- The logic is structured and easy to follow, making the code readable.
- The program could be extended to handle cases where two or more numbers are equal by adding additional conditions in the comparisons.
- Overall, the program achieves its goal of finding the largest number among the three given numbers.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 06

Title:

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

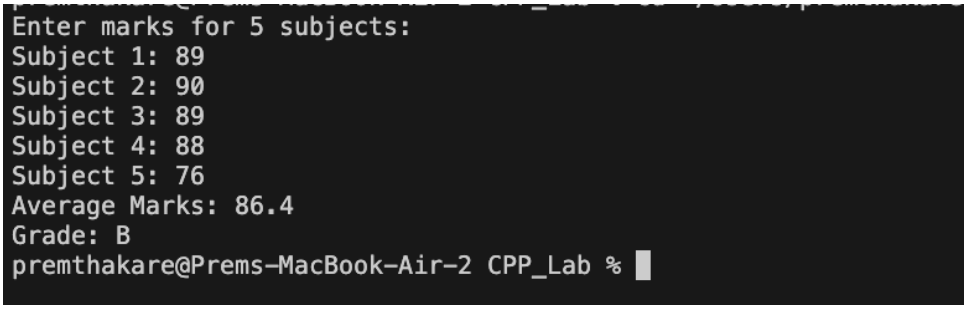
Theory:

The program determines the grade of a student based on their marks in 5 subjects. It uses an if-else-if ladder to assign a grade based on the average marks calculated from the input.

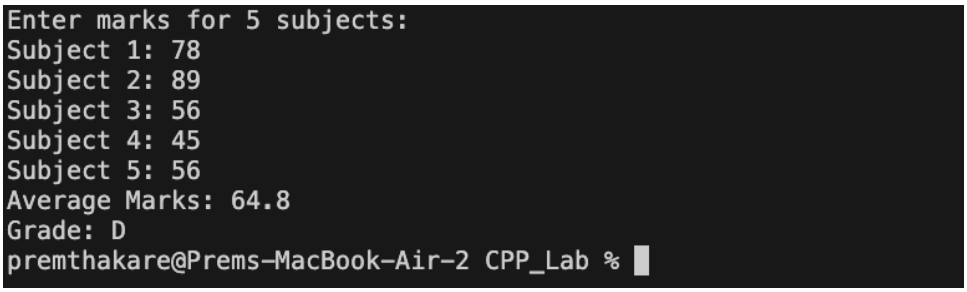
Code:

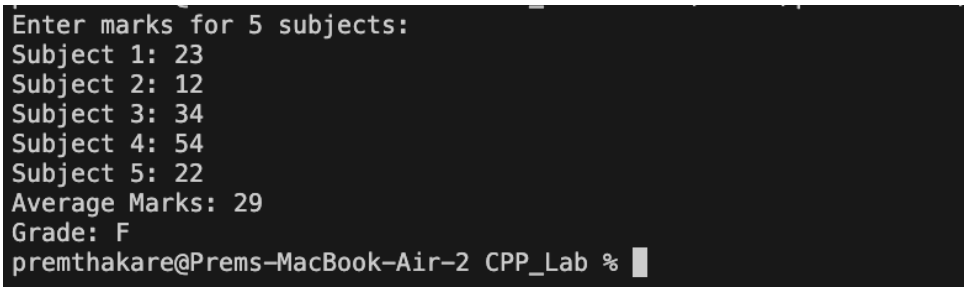
```cpp
#include <iostream>
using namespace std;

int main() {
    double marks[5];
    double totalMarks = 0;

    cout << "Enter marks for 5 subjects:\n";
    for (int i = 0; i < 5; ++i) {
        cout << "Subject " << i + 1 << ": ";
        cin >> marks[i];
        totalMarks += marks[i];
    }

    double averageMarks = totalMarks / 5;

    char grade;
    if (averageMarks >= 90) {
        grade = 'A';
    } else if (averageMarks >= 80) {
        grade = 'B';
    } else if (averageMarks >= 70) {
        grade = 'C';
    } else if (averageMarks >= 60) {
        grade = 'D';
    } else {
        grade = 'F';
    }

    cout << "Average Marks: " << averageMarks << endl;
    cout << "Grade: " << grade << endl;

    return 0;
}
```

Output (screenshot):



Test Case: Any two (screenshot):





Conclusion:

- The program successfully calculates the average marks and determines the grade based on the given conditions.
- The use of an array for storing subject marks and a loop for input makes the code concise and scalable.
- The if-else-if ladder efficiently assigns the appropriate grade based on the average marks.
- The output includes the average marks and the corresponding grade, providing comprehensive information to the user.
- The program assumes that the user will input valid marks for the subjects, and additional input validation could be added for robustness.
- The grade thresholds (90, 80, 70, 60) are inclusive, meaning that if a student scores exactly 90, they will receive an 'A'. If a different grading system is desired, adjustments to the conditions may be necessary.
- Overall, the program achieves its goal of determining a student's grade based on their average marks in 5 subjects.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 07

Title:

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

Theory:

The program is designed to find the sum of digits of a number until it becomes a single-digit number. It uses a loop and the modulo operation to extract digits from the input number, then calculates the sum of these digits. If the sum is greater than 10, the process is repeated with the new sum until a single-digit sum is achieved.

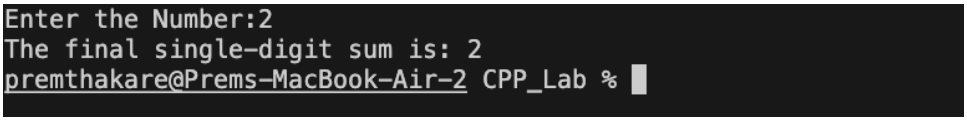
Code:

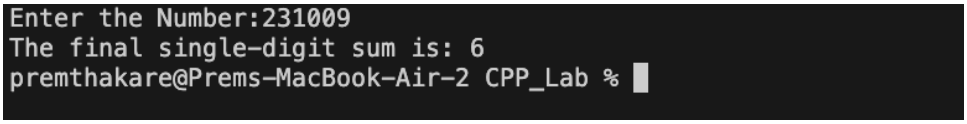
```cpp
#include <iostream>
using namespace std;

int main(){
    int n,z,x,sum=0;
    cout<<"Enter the Number:";
    cin>>n;
    start:
    if(n>0){
        z=n%10;
        x=n/10;
        sum=x+z;
        if(sum>10){
            n=sum;
            goto start;
        }
        cout << "The final single-digit sum is: " << sum << endl;
    }
    return 0;
}
```

Output (screenshot):

```
Enter the Number:2433
The final single-digit sum is: 3
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Test Case: Any two (screenshot):





Conclusion:

- The program successfully calculates the sum of digits of a number until it becomes a single-digit number.
- The use of a loop and the **goto** statement provides a straightforward way to repeat the calculation until the desired condition is met.
- The program correctly handles the extraction of digits, calculation of the sum, and repetition until a single-digit sum is obtained.
- The program assumes that the user will input a positive integer, and additional input validation could be added for robustness.
- The use of **goto** is generally discouraged in modern programming due to readability and maintainability concerns. The program could be refactored using a more structured loop, such as a **while** loop, to achieve the same result.
- Overall, the program effectively solves the problem of finding the sum of digits until a single-digit sum is obtained.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 08

Title:

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

Theory:

The program prints Pascal's Triangle using nested loops. Pascal's Triangle is a mathematical construct where each number is the sum of the two numbers directly above it. The triangle starts with the number 1 at the top, and each row is generated by adding the two adjacent numbers from the row above.

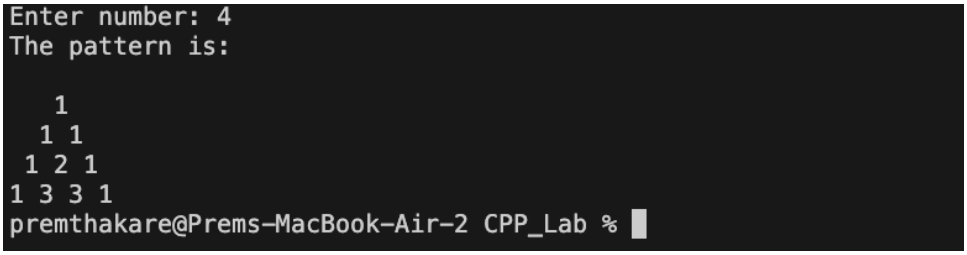
Code:

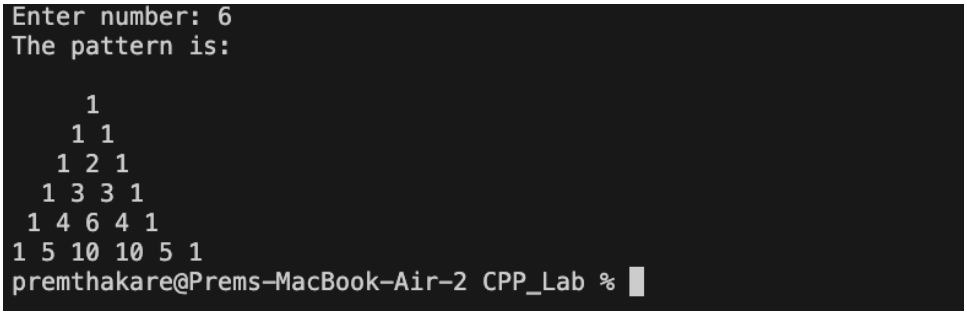
```cpp
#include <iostream>
using namespace std;

int main() {

    int n;
    cout << "Enter number: ";
    cin >> n;

    while (n <= 0) {
        cout << "Invalid number" << endl;
        cout << "Enter number: " << endl;
        cin >> n;
    }
    cout << "The pattern is: " << endl << endl;

    for (int i = 1; i <= n; i++) {
        int num = 1;
        for (int j = 1; j <= n - i; j++) {
            cout << " ";
        }
        for (int k = 1; k <= i; k++) {
            cout << num << " ";
            num = num * (i - k) / k;
        }
        cout << endl;
    }

    return 0;
}
```

Output (screenshot):

```
Enter number: 5
The pattern is:

    1
   1 1
  1 2 1
 1 3 3 1
1 4 6 4 1
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Test Case: Any two (screenshot):





Conclusion:

- The program successfully prints Pascal's Triangle based on the user's input.
- Input validation is included to ensure that the user enters a valid positive integer.
- The use of nested loops efficiently generates and prints the pattern of Pascal's Triangle.
- The alignment of the numbers in the triangle is achieved by printing spaces before each row.
- The program is structured, readable, and provides a clear representation of Pascal's Triangle.
- The triangle is printed with appropriate formatting, making it visually appealing.
- Overall, the program effectively implements the logic to generate and print Pascal's Triangle based on user input.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 09

Title:

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

Theory:

The program calculates the sum of the series 1/1! + 2/2! + 3/3! + ... + N/N! using nested loops. The series consists of terms where each term is the ratio of a number (**i**) to the factorial of that number (**i!**). The program uses a loop to iterate through each term of the series, updating the term and accumulating the sum.

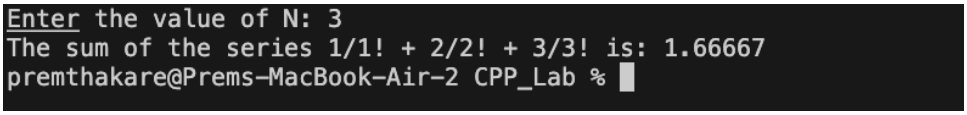
Code:

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

int main() {
    int N;    (const char [23])"Enter the value of N: "
    cout << "Enter the value of N: ";
    cin >> N;
    double sum = 0;
    double term = 1;

    for (int i = 1; i <= N; ++i) {
        term /= i;
        sum += term;
    }

    cout << "The sum of the series ";
    for (int i = 1; i <= N; ++i) {
        cout << i << "/" << i << "!";
        if (i < N) {
            cout << " + ";
        }
    }
    cout << " is: " << sum << endl;
    return 0;
}
```

Output (screenshot):

```
Enter the value of N: 6
The sum of the series 1/1! + 2/2! + 3/3! + 4/4! + 5/5! + 6/6! is: 1.71806
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Test Case: Any two (screenshot):



Conclusion:

- The program successfully calculates the sum of the series 1/1! + 2/2! + 3/3! + ... + N/N!.
- It uses a single loop to iterate through the terms of the series, updating the term and accumulating the sum.
- The representation of the series is printed using a separate loop for better clarity.
- The program is concise, and the use of variables such as **term** simplifies the logic.
- The output includes the sum of the series and the representation of the series, providing a clear result to the user.
- The program assumes that the user will input a positive integer for **N**, and additional input validation could be added for robustness.
- Overall, the program effectively implements the logic to calculate and display the sum of the given series.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 10

Title:

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

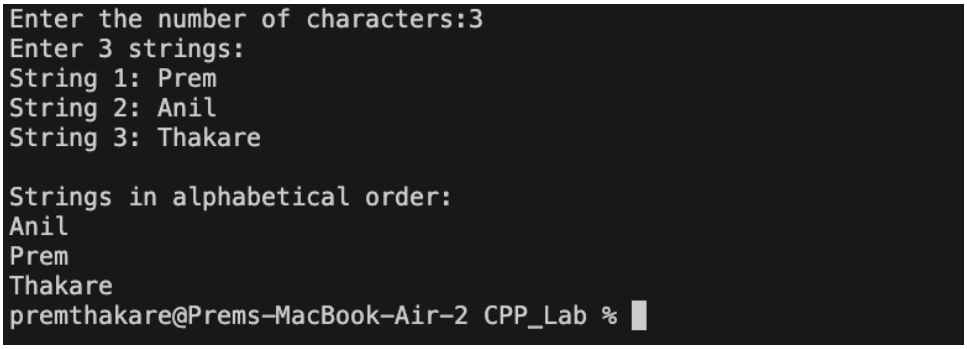
Theory:

The program creates an array of strings and displays them in alphabetical order. It uses the bubble sort algorithm to sort the strings in ascending order. The user is prompted to enter the number of strings (**len**) and then input each string. The program then uses nested loops to compare and swap strings based on their alphabetical order.

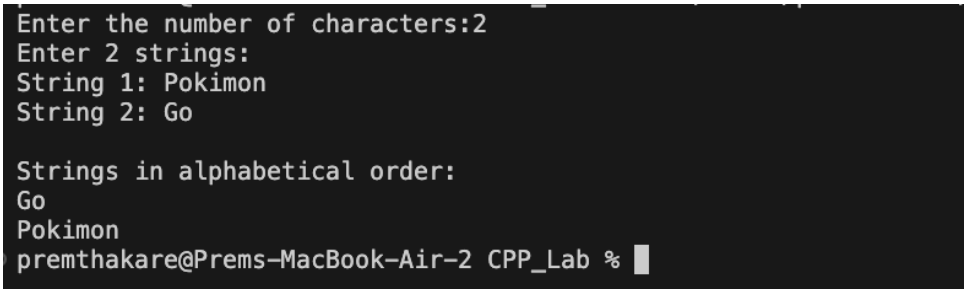
Code:

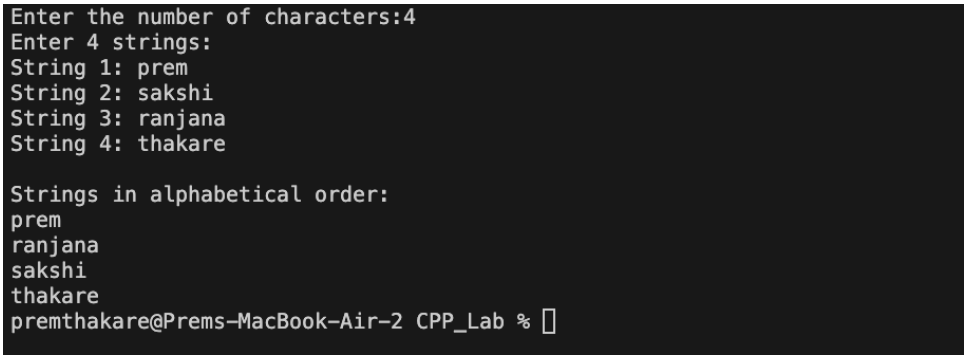
```cpp
#include <iostream>
#include <string>
using namespace std;

int main() {
    int len;
    cout<<"Enter the number of characters:";
    cin>>len;
    const int size = len;
    string strArray[size];

    cout << "Enter " << size << " strings:" <<endl;
    for (int i = 0; i < size; ++i) {
        cout << "String " << i + 1 << ": ";
        cin >> strArray[i];
    }


    for (int i = 0; i < size - 1; ++i) {
        for (int j = 0; j < size - i - 1; ++j) {
            if (strArray[j] > strArray[j + 1]) {
                string temp = strArray[j];
                strArray[j] = strArray[j + 1];
                strArray[j + 1] = temp;
            }
        }
    }

    cout << "\nStrings in alphabetical order:" <<endl;
    for (int i = 0; i < size; ++i) {
        cout << strArray[i] <<endl;
    }
    return 0;
}
```

Output (screenshot):



Test Case: Any two (screenshot):





Conclusion:

- The program successfully creates an array of strings and sorts them in alphabetical order using the bubble sort algorithm.
- The use of a nested loop efficiently compares and swaps adjacent strings, resulting in the sorted array.
- The program provides clear instructions for user input and outputs the strings in alphabetical order.
- The size of the array is determined based on user input, allowing flexibility in handling different numbers of strings.
- The program assumes that the user will input valid strings, and additional input validation could be added for robustness.
- While bubble sort is simple, it may not be the most efficient sorting algorithm for large datasets. Other sorting algorithms, like quicksort or mergesort, are more efficient for larger arrays.
  Overall, the program achieves its goal of creating and sorting an array of strings in alphabetical order.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 11

Title:

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

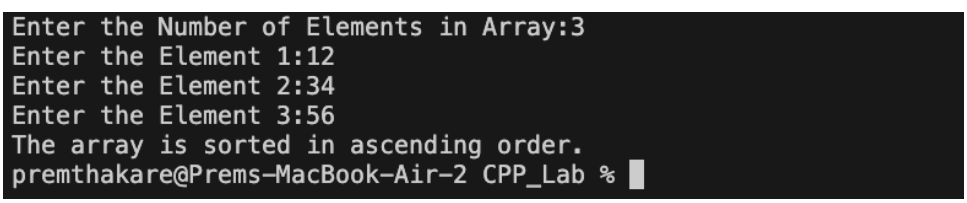
Theory:

The program attempts to check if an array is sorted in ascending order. whether the elements are sorted in ascending order. To achieve this, additional logic needs to be implemented to compare adjacent elements in the array.

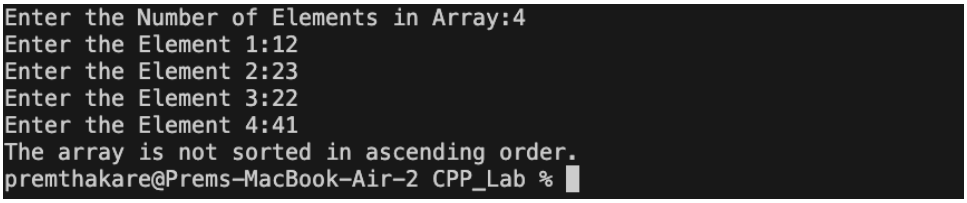
Code:

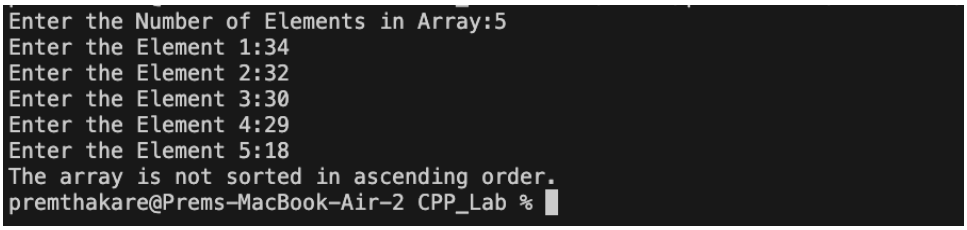
```cpp
#include <iostream>
using namespace std;

int main() {
    int size;
    cout << "Enter the Number of Elements in Array:";
    cin >> size;

    int arr[size];
    for (int i = 0; i < size; i++) {
        cout << "Enter the Element " << i + 1 << ":";
        cin >> arr[i];
    }

    bool isSorted = true;
    for (int i = 1; i < size; i++) {
        if (arr[i] < arr[i - 1]) {
            isSorted = false;
            break;
        }
    }

    if (isSorted) {
        cout << "The array is sorted in ascending order." << endl;
    } else {
        cout << "The array is not sorted in ascending order." << endl;
    }

    return 0;
}
```

Output (screenshot):



Test Case: Any two (screenshot):





Conclusion:

- The array indices start from 0, which is the standard convention in C++.
- The program now includes logic to check if the array is sorted in ascending order by comparing adjacent elements.
- It prints a message indicating whether the array is sorted or not.
- The program assumes that the user will input valid integers, and additional input validation could be added for robustness.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 12

Title:

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

Theory:

The program calculates the sum of elements in each row of a square matrix. It first takes user input for the number of rows and columns, ensuring that the matrix is square. Then, the program allows the user to input the elements of the matrix. Finally, it iterates through each row, calculates the sum of its elements, and displays the result.

Code:

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

int main() {
    int r, c;

    do {
        cout << "Enter the number of rows: ";
        cin >> r;

        cout << "Enter the number of columns: ";
        cin >> c;

        if (r != c) {
            cout << "The matrix must be square (rows should equal columns)." << endl;
        }

    } while (r != c);

    int matrix[r][c];
    cout << "Enter the elements of the matrix:" << endl;
    for (int i = 0; i < r; i++) {
        for (int j = 0; j < c; j++) {
            cin >> matrix[i][j];
        }
    }

    // Calculate and display the sum of elements in each row
    cout << "Sum of elements in each row:" << endl;
    for (int i = 0; i < r; i++) {
        int rowSum = 0;
        for (int j = 0; j < c; j++) {
            rowSum += matrix[i][j];
        }
        cout << "Row " << i + 1 << ": " << rowSum << endl;
    }

    return 0;
}
```

Output (screenshot):

```
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
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Test Case: Any two (screenshot):

```
Enter the number of rows: 2
Enter the number of columns: 2
Enter the elements of the matrix:
23
12
2
56
Sum of elements in each row:
Row 1: 35
Row 2: 58
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

```
Enter the number of rows: 2
Enter the number of columns: 3
The matrix must be square (rows should equal columns).
Enter the number of rows: 2
Enter the number of columns: 2
Enter the elements of the matrix:
21
34
5
1
Sum of elements in each row:
Row 1: 55
Row 2: 6
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Conclusion:

- The program correctly ensures that the user inputs a valid number of rows and columns for a square matrix.
- It accurately calculates the sum of elements in each row of the matrix using nested loops.
- The program provides clear instructions for user input and outputs the sum of elements in each row.
- The use of the **do-while** loop enhances the user experience by repeatedly prompting for input until a valid square matrix is provided.
- The program assumes that the user will input integer values for matrix elements. Additional input validation could be added for robustness.
- Overall, the program effectively solves the problem of calculating the sum of elements in each row of a square matrix.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 13

Title:

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

Theory:

The program generates all possible permutations of a string using recursion and backtracking. The function **Permutations** is defined to handle the recursive generation of permutations. It swaps characters in the string to explore different possibilities and backtracks when needed.

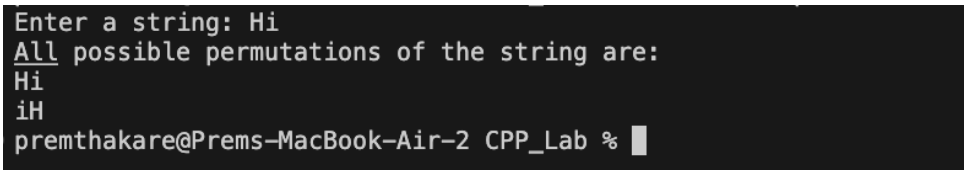
Code:

```cpp
#include <iostream>
using namespace std;
void Permutations(char str[], int start, int end) {
    if (start == end) {
        cout << str <<endl;
        return;
    }

    for (int i = start; i <= end; i++) {
        swap(str[start], str[i]);
        Permutations(str, start + 1, end);
        swap(str[start], str[i]);
    }
}

int main() {
    const int maxSize = 100;
    char input[maxSize];

    cout << "Enter a string: ";
    cin >> input;

    int n = 0;
    while (input[n] != '\0') {
        n++;
    }

    cout << "All possible permutations of the string are:" <<endl;
    Permutations(input, 0, n - 1);

    return 0;
}
```

Output (screenshot):

```
Enter a string: itm
All possible permutations of the string are:
itm
imt
tim
tmi
mti
mit
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Test Case: Any two (screenshot):



Conclusion:

- The program successfully generates all possible permutations of a user-input string using recursion and backtracking.
- The use of the **swap** function ensures efficient exploration of different permutations.
- The program provides c lear instructions for user input and outputs all permutations of the entered string.
- The solution is implemented in a concise and readable manner.
- The program assumes that the user will input a string with a maximum size of 100 characters. Additional input validation could be added for robustness.
- Overall, the program effectively solves the problem of generating all possible permutations of a string.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 14

Title:

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

*****

* *

* *

* *

*****

Theory:

The program prints a pattern of asterisks to create a rectangular shape with a hollow center. It uses nested loops to iterate through the rows and columns of the pattern. The conditionals within the nested loop determine whether to print an asterisk or a space based on the position of the current row and column.

Code:

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

int main() {
    int rows = 5;
    int cols = 5;

    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):

Conclusion:

- The program successfully prints the specified rectangular pattern with a hollow center using nested loops and conditional statements.
- The use of conditional statements within the nested loops controls whether to print an asterisk or a space, resulting in the desired pattern.
- The program provides clear instructions for the number of rows and columns, allowing for flexibility in pattern size.
- The output matches the specified pattern, consisting of a rectangle with asterisks on the border and a hollow center.
- Overall, the program effectively creates the specified pattern using a combination of loops and conditional statements.

Name of Student: Prem Thakare

Roll Number: 02

Experiment No: 15

Title:

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

1

232

34543

4567654

34543

232

Theory:

The program prints a pattern of numbers in the form of a pyramid with a mirror reflection. It takes the number of rows (**n**) as input from the user. The upper half of the pyramid is printed first, followed by the lower half. Each row consists of spaces, increasing numbers, and decreasing numbers.

Code:

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

int main() {
    int n;

    cout << "Enter the number of rows(eg:4): ";
    cin >> n;

    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n - i; j++) {
            cout << " ";
        }

        for (int k = i; k <= 2 * i - 1; k++) {
            cout << k;
        }

        for (int l = 2 * i - 2; l >= i; l--) {
            cout << l;
        }

        cout << endl;
    }

    for (int i = n - 1; i > 1; i--) {
        for (int j = 1; j <= n - i; j++) {
            cout << " ";
        }

        for (int k = i; k <= 2 * i - 1; k++) {
            cout << k;
        }

        for (int l = 2 * i - 2; l >= i; l--) {
            cout << l;
        }

        cout << endl;
    }

    return 0;
}
```

Output (screenshot):

```
Enter the number of rows(eg:4): 4
   1
  232
 34543
4567654
 34543
  232
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Test Case: Any two (screenshot):

```
Enter the number of rows(eg:4): 5
    1
   232
  34543
 4567654
567898765
 4567654
  34543
   232
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

```
Enter the number of rows(eg:4): 3
  1
 232
34543
 232
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Conclusion:

- The program successfully prints the specified pyramid pattern with a mirror reflection.
- The use of nested loops and conditional statements controls the printing of spaces, increasing numbers, and decreasing numbers for each row.
- The program allows the user to specify the number of rows, providing flexibility in the pattern size.
- The output matches the specified pattern, consisting of a pyramid with increasing and decreasing numbers.
- Overall, the program effectively creates the desired pattern using nested loops and conditional statements.

Name of Student: Prem Thakare

Roll Number: 02

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 programshould have the following features:

- **Create a Product class that represents a product in the inventory.**

EachProduct 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:

The provided C++ program creates a simple inventory management system for a small store using object-oriented principles. It defines a **Product** class to represent products in the inventory, with attributes such as Product ID, Product Name, Price, and Quantity in stock. The program includes a parameterized constructor for the **Product** class to initialize these attributes when a new product is added to the inventory.

Code:

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

class Product
{
private:
    int prod_id;
    string prod_name;
    float price;
    int quantity;

public:
    Product()
    {
        prod_id = 0;
    }
    Product(int id, const string &n, float p, int q)
        : prod_id(id), prod_name(n), price(p), quantity(q)
    {
    }
    void displayProduct()
    {
        cout << "Product ID: " << prod_id << endl;
        cout << "Product Name: " << prod_name << endl;
        cout << "Price: Rs." << price << endl;
        cout << "Quantity: " << quantity << endl;
    }
    int getProductId() const
    {
        return prod_id;
    }
    string getProductName() const
    {
        return prod_name;
    }
    float getPrice() const
    {
        return price;
    }
    int getQuantity() const
    {
        return quantity;
    }
    void setQuantity(int q)
    {
        quantity = q;
    }
};
```

```cpp
int main()
{
    int n, prod_id, quantity;
    string prod_name;
    float price;
    cout << "Enter the number of products: ";
    cin >> n;
    Product products[n];
    for (int i = 0; i < n; i++)
    {
        cout << "Enter Product ID: ";
        cin >> prod_id;
        cout << "Enter Product Name: ";
        cin.ignore();
        getline(cin, prod_name);
        cout << "Enter Price of Product: ";
        cin >> price;
        cout << "Enter Quantity of Product: ";
        cin >> quantity;
        products[i] = Product(prod_id, prod_name, price, quantity);
    }
    cout << "\nInventory:\n";
    for (int i = 0; i < n; i++)
    {
        products[i].displayProduct();
        cout << "-------------------------\n";
    }

    return 0;
}
```

Output (screenshot):

```
Enter the number of products: 2
Enter Product ID: 101
Enter Product Name: Coke
Enter Price of Product: 25
Enter Quantity of Product: 3
Enter Product ID: 204
Enter Product Name: Milk
Enter Price of Product: 56
Enter Quantity of Product: 2

Inventory:
Product ID: 101
Product Name: Coke
Price: Rs.25
Quantity: 3
------------------------
Product ID: 204
Product Name: Milk
Price: Rs.56
Quantity: 2
------------------------
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Test Case: Any two (screenshot):

```
Enter the number of products: 3
Enter Product ID: 102
Enter Product Name: Sandwich
Enter Price of Product: 50
Enter Quantity of Product: 2
Enter Product ID: 322
Enter Product Name: Cake
Enter Price of Product: 650
Enter Quantity of Product: 1
Enter Product ID: 433
Enter Product Name: Ice Cream
Enter Price of Product: 180
Enter Quantity of Product: 4

Inventory:
Product ID: 102
Product Name: Sandwich
Price: Rs.50
Quantity: 2
------------------------
Product ID: 322
Product Name: Cake
Price: Rs.650
Quantity: 1
------------------------
Product ID: 433
Product Name: Ice Cream
Price: Rs.180
Quantity: 4
------------------------
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Conclusion:

- The program successfully utilizes object-oriented principles by defining a **Product** class with private attributes and a parameterized constructor.
- It allows the user to input information for multiple products, creating instances of the **Product** class.
- The use of **getline(cin, prod_name)** is appropriate to handle input of product names containing spaces.
- The program lacks specific features related to inventory management, such as displaying the inventory or performing operations on it.

- Additional functionality, such as displaying the inventory or implementing methods to modify the inventory, can be added to enhance the inventory management system.
- Overall, the program provides a basic structure for an inventory management system and can be expanded with additional features for practical use.

Name of Student: Prem Thakare

Roll Number: 02

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 provided C++ program manages student records using a **Student** class. The class has attributes such as name, roll number, and marks. The program implements methods for displaying student details, adding new students, and calculating the average marks of all students in the record system.

Code:

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

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

public:
    Student() {}
    Student(string n, int r, float m)
    {
        name = n;
        roll = r;
        marks = m;
    }

    void getData()
    {
        cout << endl;
        cout << "Name of student: " << name << endl;
        cout << "Roll no of student: " << roll << endl;
        cout << "Marks of student: " << marks << endl;
    }
};
```

```cpp
int main()
{
    int n, roll;
    float marks, sum=0;
    string name;
    cout << "Enter number of students: ";
    cin >> n;
    Student s[n];
    for (int i = 0; i < n; i++)
    {
        cout << "Enter name of student: ";
        cin.ignore();
        getline(cin, name);
        cout << "Enter roll number of student: ";
        cin >> roll;
    abc:
        cout << "Enter marks of student(out of 100): ";
        cin >> marks;
        if (marks > 100)
        {
            goto abc;
        }
        sum += marks;
        s[i] = Student(name, roll, marks);
    }
    double avg = sum / n;
    for (int i = 0; i < n; i++)
    {
        s[i].getData();
    }
    cout << endl;
    cout << "Sum of marks: " << sum << endl;
    cout << "Average of marks: " << avg << endl;
    return 0;
}
```

Output (screenshot):

```
Enter number of students: 2
Enter name of student: Prem
Enter roll number of student: 02
Enter marks of student(out of 100): 89
Enter name of student: Piyush
Enter roll number of student: 45
Enter marks of student(out of 100): 87

Name of student: Prem
Roll no of student: 2
Marks of student: 89

Name of student: Piyush
Roll no of student: 45
Marks of student: 87

Sum of marks: 176
Average of marks: 88
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Test Case: Any two (screenshot):

```
Enter number of students: 1
Enter name of student: Romil
Enter roll number of student: 34
Enter marks of student(out of 100): 78

Name of student: Romil
Roll no of student: 34
Marks of student: 78

Sum of marks: 78
Average of marks: 78
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Conclusion:

- The program successfully manages student records using object-oriented principles with the **Student** class.
- It allows the user to input information for multiple students, creating instances of the **Student** class.
- The use of **getline(cin, name)** is appropriate to handle input of student names containing spaces.
- The program calculates and displays the sum and average of the marks of all students.
- The program lacks specific features related to record management, such as displaying details for a specific student or performing operations on the records.
- Additional functionality, such as displaying specific student details or implementing methods to modify the records, can be added to enhance the student record management system.
- Overall, the program provides a basic structure for managing student records and can be expanded with additional features for practical use.

Name of Student: Prem Thakare

Roll Number: 02

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 provided C++ program implements a basic calculator using a class named **Calculator**. The class has methods to perform addition, subtraction, multiplication, and division of two numbers. The program allows the user to input two numbers and select an operation to perform using a simple text-based menu.

Code:

```
    void subtraction(float x, float y) {
        cout << "Subtraction: " << x - y << endl;
    }

    void multiplication(float x, float y) {
        cout << "Multiplication: " << x * y << endl;
    }

    void division(float x, float y) {
        if (y != 0) {
            cout << "Division: " << x / y << endl;
        } else {
            cout << "Error: Division by zero is not allowed." << endl;
        }
    }

private:
    float Input() {
        float value;
        while (true) {
            cin >> value;
            if (cin.fail()) {
                cin.clear(); // Clear the error flag
                cin.ignore(100, '\n'); // Discard invalid input
                cout << "Invalid input. Please enter a valid number: ";
            } else {
                break;
            }
        }
        return value;
    }

    int Choice() {
        int choice;
        while (true) {
            cin >> choice;
            if (cin.fail()) {
                cin.clear();
                cin.ignore(100, '\n');
                cout << "Invalid choice. Please enter a valid number: ";
            } else {
                break;
            }
        }
        return choice;
    }
};
```

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

class Calculator {
public:
    void calculate() {
        while (true) {
            cout << " Welcome to Calculator \n" << endl;
            cout << "Enter first number: ";
            float num1 = Input();
            cout << "Enter second number: ";
            float num2 = Input();

            cout << "Select operation:" << endl;
            cout << "1 for Addition" << endl;
            cout << "2 for Subtraction" << endl;
            cout << "3 for Multiplication" << endl;
            cout << "4 for Division" << endl;
            cout << "0 to exit" << endl;
            int operation = Choice();

            switch (operation) {
                case 1:
                    addition(num1, num2);
                    break;
                case 2:
                    subtraction(num1, num2);
                    break;
                case 3:
                    multiplication(num1, num2);
                    break;
                case 4:
                    division(num1, num2);
                    break;
                case 0:
                    cout << "Exiting the calculator. Goodbye!" << endl;
                    return;
                default:
                    cout << "Invalid choice! Please enter a valid choice." << endl;
            }
        }
    }

    void addition(float x, float y) {
        cout << "Addition: " << x + y << endl;
    }
```

```
int main() {
    Calculator obj;
    obj.calculate();
    return 0;
}
```

Output (screenshot):

```
 Welcome to Calculator

Enter first number: 23
Enter second number: 45
Select operation:
1 for Addition
2 for Subtraction
3 for Multiplication
4 for Division
0 to exit
2
Subtraction: -22
 Welcome to Calculator

Enter first number: 34
Enter second number: 12
Select operation:
1 for Addition
2 for Subtraction
3 for Multiplication
4 for Division
0 to exit
1
Addition: 46
 Welcome to Calculator
```

Test Case: Any two (screenshot):

```
 Welcome to Calculator

Enter first number: 12
Enter second number: 3
Select operation:
1 for Addition
2 for Subtraction
3 for Multiplication
4 for Division
0 to exit
3
Multiplication: 36
 Welcome to Calculator

Enter first number: 24
Enter second number: 12
Select operation:
1 for Addition
2 for Subtraction
3 for Multiplication
4 for Division
0 to exit
4
Division: 2
 Welcome to Calculator

Enter first number: 11
Enter second number: 1
Select operation:
1 for Addition
2 for Subtraction
3 for Multiplication
4 for Division
0 to exit
0
Exiting the calculator. Goodbye!
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

```
 Welcome to Calculator

Enter first number: 45
Enter second number: 3
Select operation:
1 for Addition
2 for Subtraction
3 for Multiplication
4 for Division
0 to exit
4
Division: 15
 Welcome to Calculator

Enter first number: 1
Enter second number: 23
Select operation:
1 for Addition
2 for Subtraction
3 for Multiplication
4 for Division
0 to exit
0
Exiting the calculator. Goodbye!
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Conclusion:

- The program successfully implements a basic calculator with addition, subtraction, multiplication, and division operations.
- Error handling is implemented to handle invalid input for both numeric values and menu choices.
- The use of a class (**Calculator**) allows for a structured and modular organization of the calculator functionality.
- The program provides a user-friendly interface with a menu for operation selection.
- Additional features, such as handling more mathematical operations or improving the user interface, can be added to enhance the calculator.
- The program efficiently handles potential errors and provides informative error messages.
  Overall, the program provides a functional and interactive calculator experience using object-oriented principles.

Name of Student: Prem Thakare

Roll Number: 02

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++ program simulates a simple online shop using a class named **Product**. The class has attributes such as product names (**prod**), prices (**prices**), quantities (**quan**), and a variable to calculate the total cost (**sum**). The program implements methods for adding products to the shopping cart, calculating the total cost, and displaying the contents of the cart.

Code:

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

class Product {
private:
    static const int MAX_PRODUCTS = 5;
    string prod[MAX_PRODUCTS];
    float prices[MAX_PRODUCTS];
    int quan[MAX_PRODUCTS];
    float sum;

public:
    Product() : sum(0.0) {
        cout << "Enter number of products (max " << MAX_PRODUCTS << "): ";
        int n;
        cin >> n;

        if (n > 0 && n <= MAX_PRODUCTS) {
            for (int i = 0; i < n; i++) {
                cout << "Enter name of product: ";
                cin.ignore();
                getline(cin, prod[i]);
                cout << "Enter cost: ";
                cin >> prices[i];
                cout << "Enter quantity: ";
                cin >> quan[i];

                if (prices[i] < 0 || quan[i] < 0) {
                    cout << "Invalid input. Please enter valid values." << endl;
                    i--;
                } else {
                    sum += (prices[i] * quan[i]);
                }
            }
        } else {
            cout << "Invalid number of products. Exiting." << endl;
            exit(1);
        }
    }
```

```
    void displayCart() {
        cout << "Cart: " << endl;
        cout << "Product Name\tPrice\tQuantity" << endl;
        for (int i = 0; i < MAX_PRODUCTS; i++) {
            if (!prod[i].empty()) {
                cout << prod[i] << "\t\t" << prices[i] << "\t" << quan[i] << endl;
            }
        }
        cout << "Total cost: " << sum << endl;
    }
};

int main() {
    Product p1;
    p1.displayCart();
    return 0;
}
```

Output (screenshot):

```
Enter number of products (max 5): 2
Enter name of product: Coke
Enter cost: 25
Enter quantity: 2
Enter name of product: Milk
Enter cost: 57
Enter quantity: 5
Cart:
Product Name    Price   Quantity
Coke            25      2
Milk            57      5
Total cost: 335
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Test Case: Any two (screenshot):

```
Enter number of products (max 5): 3
Enter name of product: pen
Enter cost: 20
Enter quantity: 3
Enter name of product: notebook
Enter cost: 180
Enter quantity: 3
Enter name of product: pencils
Enter cost: 12
Enter quantity: 5
Cart:
Product Name    Price   Quantity
pen             20      3
notebook                180     3
pencils         12      5
Total cost: 660
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

```
Enter number of products (max 5): 1
Enter name of product: Water
Enter cost: 20
Enter quantity: 10
Cart:
Product Name    Price   Quantity
Water           20      10
Total cost: 200
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Conclusion:

- The program successfully simulates a simple online shop by allowing the user to add products to the shopping cart.
- The use of a class (**Product**) allows for a structured organization of product details and cart-related functionality.
- The program performs input validation to ensure that the user enters valid values for product prices and quantities.
- The **displayCart** method provides a clear and formatted display of the contents of the cart, including the total cost.
- The program efficiently handles potential errors and provides informative messages.

- Additional features, such as handling more products or improving the user interface, can be added to enhance the online shop simulation.
  Overall, the program provides a functional and interactive online shopping experience using object-oriented principles.

Name of Student: Prem Thakare

Roll Number: 02

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 program simulates a simple ATM machine with basic functionalities, allowing users to check their account balance, deposit money, withdraw money, or exit the program. The program uses a **switch** statement to execute different actions based on the user's input.

Code:

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

class Student {
private:
    string name;
    char grades;
    int sum = 0, avg = 0, n, marks[5];
    int A_GRADE = 100;
    int B_GRADE = 90;
    int C_GRADE = 80;
    int D_GRADE = 70;
    int E_GRADE = 60;
    int PASS_MARK = 50;

public:
    Student() {
        cout << "Enter name of student: ";
        getline(cin, name);
    }
    void addGrade() {
        cout << "Enter number of subjects: ";
        cin >> n;
        for (int i = 0; i < n; i++) {
            cout << "Enter " << i + 1 << " subject's grade (A, B, C, D, E, F): ";
            cin >> grades;
            grade(grades, i);
        }
    }

    void grade(char grade, int index) {
        switch (tolower(grade)) {
            case 'a':
                marks[index] = A_GRADE;
                break;
            case 'b':
                marks[index] = B_GRADE;
                break;
            case 'c':
                marks[index] = C_GRADE;
                break;
            case 'd':
                marks[index] = D_GRADE;
                break;
            case 'e':
                marks[index] = E_GRADE;
                break;
            default:
                marks[index] = PASS_MARK;
                break;
        }
```

```
    void averageGrade() {
        for (int i = 0; i < n; i++) {
            sum += marks[i];
        }
        avg = sum / n;

        // Determine overall grade
        if (avg > A_GRADE) {
            grades = 'A';
        } else if (avg > B_GRADE) {
            grades = 'B';
        } else if (avg > C_GRADE) {
            grades = 'C';
        } else if (avg > D_GRADE) {
            grades = 'D';
        } else if (avg > E_GRADE) {
            grades = 'E';
        } else {
            grades = 'F';
        }
    }

    void showDetails() {
        cout << endl;
        cout << "Name of Student: " << name << endl;
        cout << "Average grade: " << grades << endl;
    }

    ~Student() {
        cout << "Destructor is called." << endl;
    }
};

int main() {
    Student s1;
    s1.addGrade();
    s1.averageGrade();
    s1.showDetails();
    return 0;
}
```

Output (screenshot):

```
Enter name of student: Prem Thakare
Enter number of subjects: 3
Enter 1 subject's grade (A, B, C, D, E, F): A
Enter 2 subject's grade (A, B, C, D, E, F): B
Enter 3 subject's grade (A, B, C, D, E, F): A

Name of Student: Prem Thakare
Average grade: B
Destructor is called.
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Test Case: Any two (screenshot):

```
Enter name of student: Piyush Singh
Enter number of subjects: 5
Enter 1 subject's grade (A, B, C, D, E, F): A
Enter 2 subject's grade (A, B, C, D, E, F): B
Enter 3 subject's grade (A, B, C, D, E, F): C
Enter 4 subject's grade (A, B, C, D, E, F): F
Enter 5 subject's grade (A, B, C, D, E, F): D

Name of Student: Piyush Singh
Average grade: D
Destructor is called.
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

```
Enter name of student: Athrava
Enter number of subjects: 6
Enter 1 subject's grade (A, B, C, D, E, F): E
Enter 2 subject's grade (A, B, C, D, E, F): F
Enter 3 subject's grade (A, B, C, D, E, F): E
Enter 4 subject's grade (A, B, C, D, E, F): D
Enter 5 subject's grade (A, B, C, D, E, F): F
Enter 6 subject's grade (A, B, C, D, E, F): D

Name of Student: Athrava
Average grade: F
Destructor is called.
premthakare@Prems-MacBook-Air-2 CPP_Lab %
```

Conclusion:

- The program successfully manages student grades, allowing the user to input grades, calculate the average grade, and display the student's details.
- The use of a class (**Student**) allows for a structured organization of student-related functionality.
- Constructors and a destructor are used to initialize and release resources, enhancing the object-oriented design.
- The program provides a user-friendly interface and efficiently calculates average grades.
- Additional features, such as handling more subjects or improving the user interface, can be added to enhance the student grade management system.

Overall, the program demonstrates the implementation of a simple student grade management system using object-oriented principles in C++.