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

C++ PROGRAMMING LAB

Prepared by:

Name of Student: Prem Thatikonda

Roll No: 06

Batch: 2023-27

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 an 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 create 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.

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Name of Student: Prem Thatikonda

Roll Number: 06

Experiment No: 1

Title: Program to find the roots of a quadratic equation

Theory:

- The program starts by taking user input for the coefficients a,b, and c.
- It calculates the discriminant using the formula b^2 - 4ac.
- Depending on the value of the discriminant, the program calculates and displays the roots.
    - If the discriminant is greater than zero, it calculates two distinct real roots.
    - If the discriminant is equal to zero, it calculates a single real root.
    - If the discriminant is less than zero, it calculates complex roots using the imaginary part.

Code:

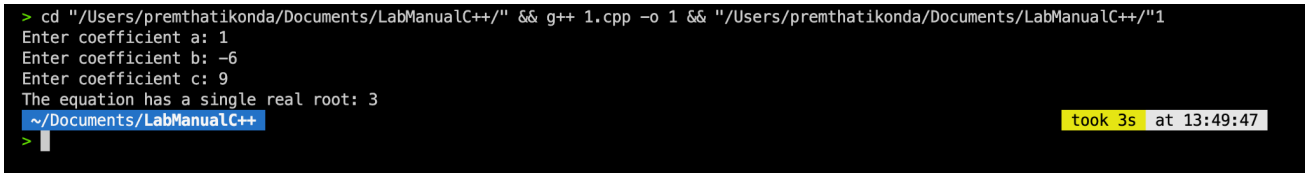
```cpp
#include <iostream>
#include <cmath>
using namespace std;


int main() {
   // Coefficients of the quadratic equation: ax^2 + bx + c = 0
   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;


// Calculating the discriminant
double discriminant = b * b - 4 * a * c;


// Check the value of the discriminant
if (discriminant > 0){
    // Calculate two distinct real roots
    double root1 = (-b + sqrt(discriminant)) / (2 * a);
    double root2 = (-b - sqrt(discriminant)) / (2 * a);


    cout << "Root 1: " << root1 << endl;
    cout << "Root 2: " << root2 << endl;
}


else if (discriminant == 0){
    // Calculating the single real root
    double root = -b / (2 * a);
    cout << "The equation has a single real root: " << root << endl;
}


else{
    // Calculating imaginary part
    double imaginaryPart = sqrt(abs(discriminant)) / (2 * a);
```

```
        cout << "Root 1: " << -b / (2 * a) << " + " << imaginaryPart << "i" << endl;
        cout << "Root 2: " << -b / (2 * a) << " - " << imaginaryPart << "i" << endl;
    }

    return 0;
}
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

Thus I have written the code to find square roots of a quadratic equation using the discriminant formula using test cases and if-else structures to determine which formula to use.

Experiment No : 2

Title: Program to calculate the power of a number using a loop

Theory:

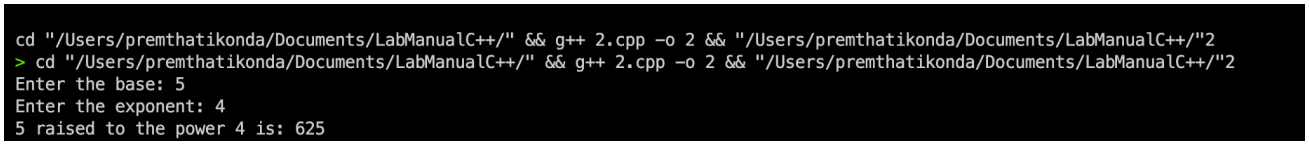
Using a for loop, iterate from 1 to the exponent entered by the user, and keep multiplying the result by the base that many times.

Code:

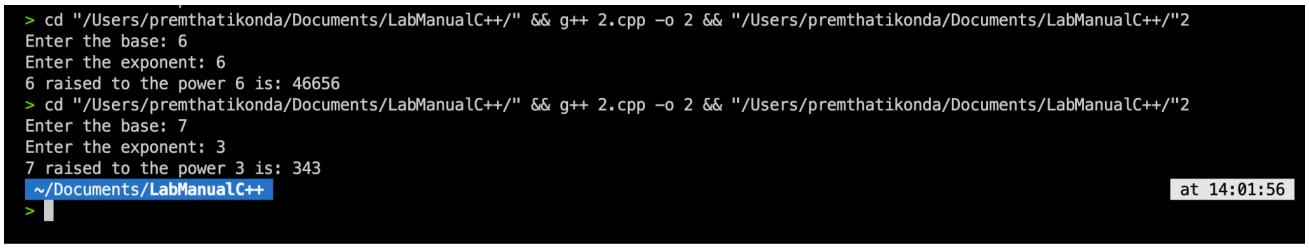
```cpp
#include <iostream>

using namespace std;



int main() {

   // Input base and exponent from the user

   double base, exponent;



   cout << "Enter the base: ";

   cin >> base;



   cout << "Enter the exponent: ";

   cin >> exponent;



   // Initialize result to 1, as any number to the power of 0 is 1

   long long int result = 1;
```

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

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

    return 0;
}
```

Output (screenshot):



Test Cases (any 2):



Conclusion:

Hence, using a for loop we can calculate the power of a number.

Experiment No : 3

Title: Check if a given string is a palindrome

Theory:

Take string input; create a function that iterates through the string from 1st character to last, and compares 1st and last character, 2nd and 2nd last character, and so on; if the characters are the same, continue. As soon as 2 characters don't match, return False. If the loop ends after all iterations, it means that all checked characters were the same, hence return true at the end of the function.

Code:

```
#include <iostream>

#include <string>


using namespace std;


bool isPalindrome(string str) {

   int length = str.length();


   for (int i = 0; i < length / 2; ++i) {

       if (str[i] != str[length - 1 - i]) {

           // Characters don't match, not a palindrome

           return false;

       }
```

```
    }

    return true;

}



int main() {

    string input;

    cout << "Enter a string: ";

    getline(cin, input);



    if (isPalindrome(input)) {

        cout << "The given string is a palindrome.\n";

    }

    else{

        cout << "The given string is not a palindrome.\n";

    }

    return 0;

}
```

Output (screenshot):

```
cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 3.cpp -o 3 && "/Users/premthatikonda/Documents/LabManualC++/"3
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 3.cpp -o 3 && "/Users/premthatikonda/Documents/LabManualC++/"3
Enter a string: racecar
The given string is a palindrome.
```

Test Cases (any 2):

```
 > cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 3.cpp -o 3 && "/Users/premthatikonda/Documents/LabManualC++/"3
 Enter a string: hello
 The given string is not a palindrome.
 > cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 3.cpp -o 3 && "/Users/premthatikonda/Documents/LabManualC++/"3
 Enter a string: yayay
 The given string is a palindrome.
  ~/Documents/LabManualC++                                    took 4s  at 14:08:56
 >
```

Conclusion: Hence created a program to check if a string is palindrome or not.

Experiment No : 4

Title: Program to simulate a simple ATM machine with basic functions to withdraw, check balance and deposit money.

Theory:

Using switch case, we can take user input on what they want to do and based on that, do the respective functions of either checking balance, withdrawing or depositing money, or exiting the program.

Code:

```
#include <iostream>

using namespace std;



void atmMenu() {

   // Initial balance

   double balance = 1000.0;



   while (true) {

       // Display menu

       cout << "\nATM Menu:\n";

       cout << "1. Check Balance\n";

       cout << "2. Deposit\n";

       cout << "3. Withdraw\n";

       cout << "4. Exit\n";

       cout << "Enter your choice (1-4): ";
```

```cpp
// User choice
int choice;
cin >> choice;


switch (choice) {
    case 1:
        // Check Balance
        cout << "Your balance: $" << balance << endl;
        break;


    case 2:
        // Deposit
        double depositAmount;
        cout << "Enter deposit amount: $";
        cin >> depositAmount;


        if (depositAmount > 0) {
            balance += depositAmount;
            cout << "Deposit successful. New balance: $" << balance << endl;
        } else {
            cout << "Invalid deposit amount.\n";
        }
        break;


    case 3:
        // Withdraw
        double withdrawAmount;
        cout << "Enter withdrawal amount: $";
        cin >> withdrawAmount;
```

```
            if (withdrawAmount > 0 && withdrawAmount <= balance) {
                balance -= withdrawAmount;
                cout << "Withdrawal successful. New balance: $" << balance << endl;
            } else {
                cout << "Invalid withdrawal amount or insufficient funds.\n";
            }
            break;

        case 4:
            // Exit
            cout << "Exiting ATM. Thank you!\n";
            return;

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

int main() {
    cout << "Welcome to the Simple ATM Machine!\n";
    atmMenu();

    return 0;
}
```

Output (screenshot):

```
cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 4.cpp -o 4 && "/Users/premthatikonda/Documents/LabManualC++/"4
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 4.cpp -o 4 && "/Users/premthatikonda/Documents/LabManualC++/"4
Welcome to the Simple ATM Machine!

ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice (1-4): 1
Your balance: $1000
```

Test Cases (any 2):

```
ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice (1-4): 2
Enter deposit amount: $5000
Deposit successful. New balance: $6000

ATM Menu:
1. Check Balance
2. Deposit
3. Withdraw
4. Exit
Enter your choice (1-4): 3
Enter withdrawal amount: $2000
Withdrawal successful. New balance: $4000
```

Conclusion:

Hence using switch-case, we can simulate a simple atm machine with basic functionalities.

Experiment No : 5

Title: Program to find out the greatest number among 3 numbers using nested if-else statements.

Theory:

Take 3 numbers input, use an if else statement to compare if 1st number is greater than 2nd, then nest it with a check if it's greater than the 3rd number too.

Similarly, check if the 2nd is greater than the 1st, then nest with check if it's greater than the 3rd too.

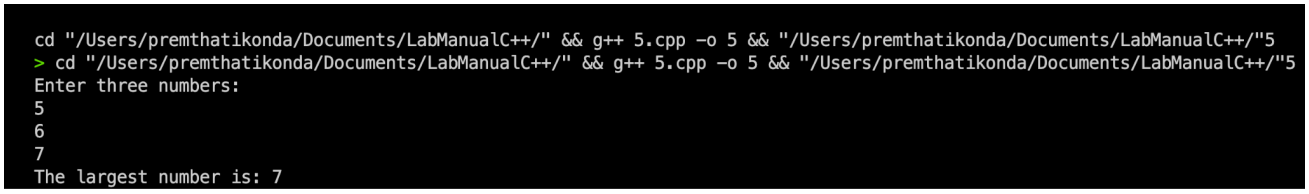
Else, declare 3rd number as the greatest.

Code:

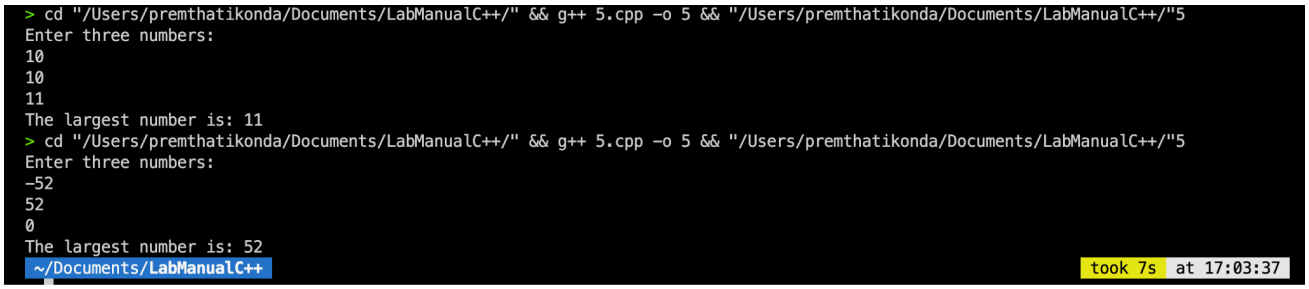
```
#include <iostream>

using namespace std;



int main() {

   double num1, num2, num3;



   cout << "Enter three numbers: \n";

   cin >> num1 >> num2 >> num3;



   // Check and find the largest number using nested if-else statements

   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 Cases (any 2):



Conclusion:

Hence, using nested if-else statements, we can find out the greatest number amongst 3 numbers.

Experiment No : 6

Title: Calculate a student's grade based on the marks obtained in 5 subjects using an if-else ladder.

Theory:

Take marks input for the 5 marks, calculate the average by dividing the sum of those 5 marks by 5 (no. of subjects). Now according to a certain range of marks, declare the grade obtained by the student.

Code:

```
#include <iostream>

using namespace std;



int main() {

   int subject1, subject2, subject3, subject4, subject5;



   cout << "Enter marks for 5 subjects (out of 100 each):\n";

   cout << "Subject 1: ";

   cin >> subject1;



   cout << "Subject 2: ";

   cin >> subject2;



   cout << "Subject 3: ";

   cin >> subject3;



   cout << "Subject 4: ";

   cin >> subject4;
```

```
    cout << "Subject 5: ";
    cin >> subject5;

    // Calculate average marks
    double average = (subject1 + subject2 + subject3 + subject4 + subject5) / 5.0;

    if (average >= 90){
        cout << "Grade: A\n";
    }
    else if (average >= 80) {
        cout << "Grade: B\n";
    }
    else if (average >= 70) {
        cout << "Grade: C\n";
    }
    else if (average >= 60) {
        cout << "Grade: D\n";
    }
    else {
        cout << "Grade: F (Fail)\n";
    }

    return 0;
}
```

Output (screenshot):

```
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 6.cpp -o 6 && "/Users/premthatikonda/Documents/LabManualC++/"6
Enter marks for 5 subjects (out of 100 each):
Subject 1: 80
Subject 2: 80
Subject 3: 80
Subject 4: 80
Subject 5: 80
Grade: B
```

Test Cases (any 2):

```
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 6.cpp -o 6 && "/Users/premthatikonda/Documents/LabManualC++/"6
Enter marks for 5 subjects (out of 100 each):
Subject 1: 100
Subject 2: 60
Subject 3: 70
Subject 4: 35
Subject 5: 95
Grade: C
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 6.cpp -o 6 && "/Users/premthatikonda/Documents/LabManualC++/"6
Enter marks for 5 subjects (out of 100 each):
Subject 1: 90
Subject 2: 90
Subject 3: 90
Subject 4: 90
Subject 5: 90
Grade: A
~/Documents/LabManualC++                                took 4s at 17:10:19
```

Conclusion:

Hence, a student's grade can be obtained by asking the user for the marks obtained in 5 subjects.

Experiment No : 7

Title: Sum of digits of a number till it becomes a single-digit number

Theory:

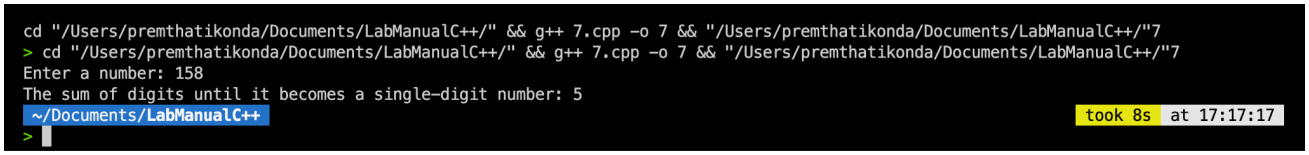
calculate the sum of digits of a user-input number until it becomes a single-digit number. The `sumOfDigits` function iterates through the digits using a `while` loop. The main function repeatedly calls this function until the entered number is less than 10. The final single-digit result is displayed to the user.

Code:

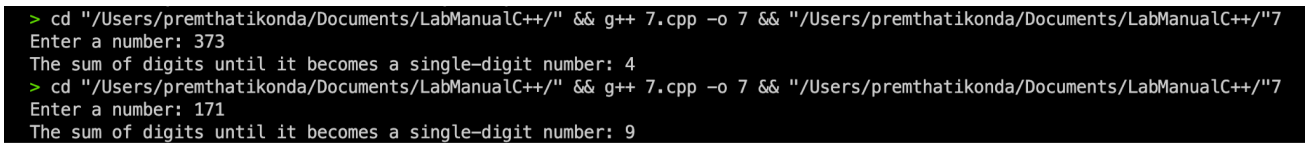
```
#include <iostream>

using namespace std;



int sumOfDigits(int num) {

   int sum = 0;



   while (num > 0) {

       sum += num % 10;

       num /= 10;

   }



   return sum;

}



int main() {

   int number;



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

```
    cin >> number;


    while (number >= 10) {
        number = sumOfDigits(number);
    }


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


    return 0;
}
```

Output (screenshot):



Test Cases (any 2):



Conclusion:

Hence using while loops, we can repeatedly add the digits till the sum is a single digit number.

Experiment No : 8

Title: Pascal's triangle using nested loops

Theory:

Take input for the number of rows to print. 2 nested loops to iterate through each row and each coefficient within a row. Inner loop calculates and returns the coefficient to be printed using a formula n! / k!(n-k)! .

Spaces are to be printed between each number for the triangular shape.

Code:

```
#include <iostream>

using namespace std;


int main() {

   int numRows;


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

   cin >> numRows;


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

       int coefficient = 1;


       // Print spaces to center the triangle

       for (int j = 0; j < numRows - i; ++j) {

           cout << "  ";

       }
```

```
        // Print coefficients in each row
        for (int j = 0; j <= i; ++j) {
            cout << coefficient << "    ";

            // Calculate the next coefficient
            coefficient = coefficient * (i - j) / (j + 1);
        }

        cout << endl;
    }

    return 0;
}
```

Output (screenshot):

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

Test Cases (any 2):

```
cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 8.cpp -o 8 && "/Users/premthatikonda/Documents/LabManualC++/"8
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 8.cpp -o 8 && "/Users/premthatikonda/Documents/LabManualC++/"8
Enter the number of rows for Pascal's Triangle: 5
         1
        1   1
      1   2   1
    1   3   3   1
  1   4   6   4   1
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 8.cpp -o 8 && "/Users/premthatikonda/Documents/LabManualC++/"8
Enter the number of rows for Pascal's Triangle: 7
            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
 ~/Documents/LabManualC++                                    at 17:29:23
>
```

Conclusion:

Using nested loops and logic, pascal's triangle can be printed for the desired number of rows.

Experiment No : 9

Title: Sum of series : 1/1! + 2/2! + 3/3! + ... + N/N! using nested loops.

Theory: Take series limit from user.

WIthin a for loop, calculate numerator by multiplying iterator 'i' with 1, and denominator by calculating the factorial of that 'i' number. Then divide the numerator and denominator, and add terms to the sum.

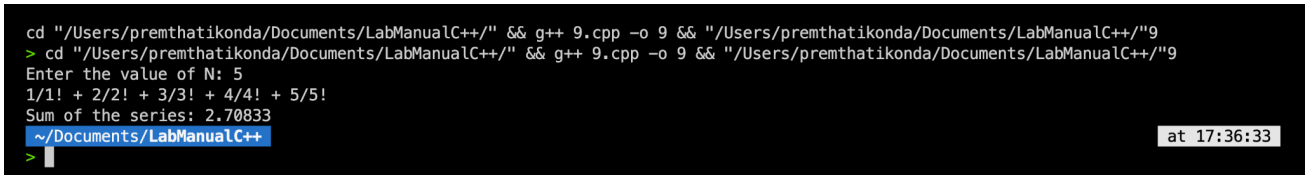
Code:

```
#include <iostream>



using namespace std;



// Function to calculate the factorial of a number

int factorial(int n) {

   if (n == 0 || n == 1) {
```

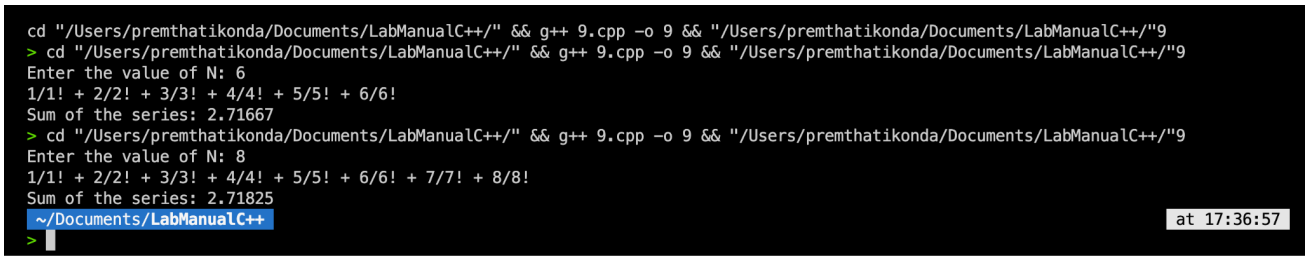
```
        return 1;
    }
    else{
        return n * factorial(n - 1);
    }
}

int main() {
    int N;
    cout << "Enter the value of N: ";
    cin >> N;

    double sum = 0;

    // Calculate and sum the series using nested loops
    for (int i = 1; i <= N; ++i) {
        double term = 1.0 * i / factorial(i);

        // Add the current term to the sum
        sum += term;

        // Display the current term
        cout << i << "/" << i << "! ";
        if (i < N) {
            cout << "+ ";
        }
    }

    // Display the overall sum
    cout << "\nSum of the series: " << sum << endl;
```

```
    return 0;
}
```

Output (screenshot):



Test Cases (any 2):



Conclusion:

Hence, by simply using a loop and logically dividing numerators and denominators by calculating factorial, we can find the sum of the series.

Experiment No : 11

Title: check if an array is in ascending order

Theory:

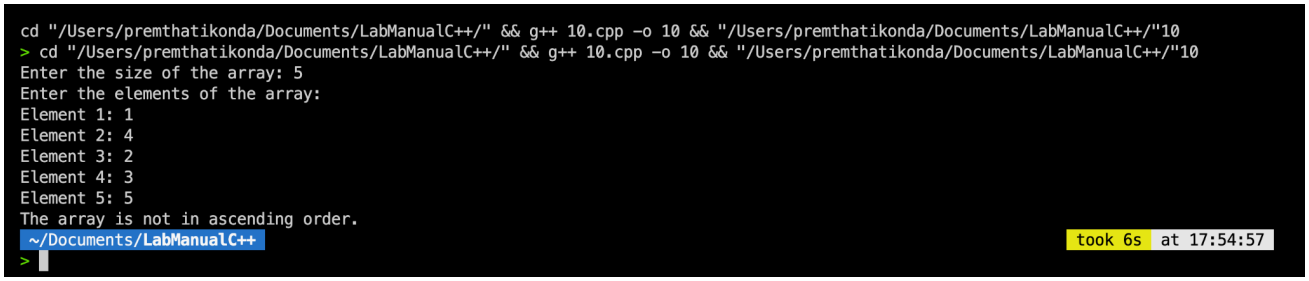
iterate through the array comparing values and as soon as a condition is satisfied, print that the array isn't in ascending order; else print after the entire loop that it is in ascending order.

Code:

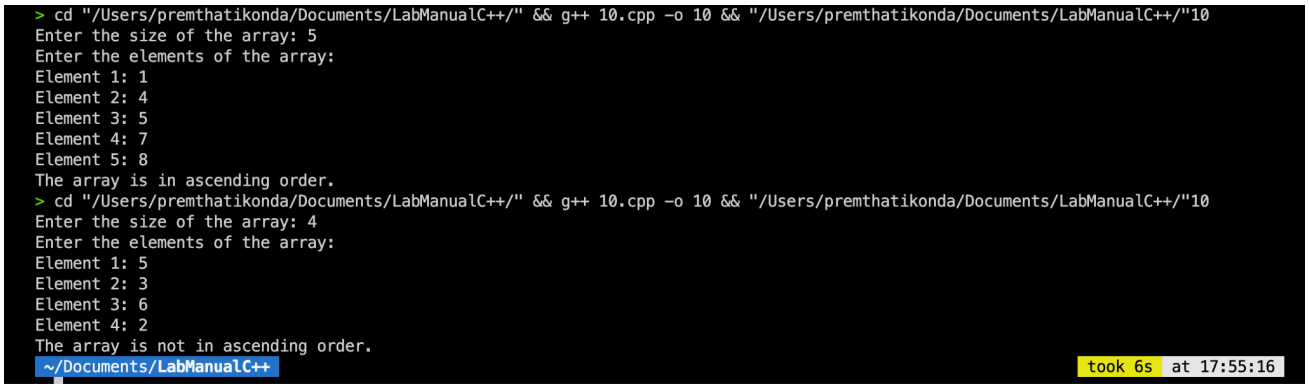
```
#include <iostream>

using namespace std;



bool isAscending(const int arr[], int size) {

   for (int i = 0; i < size - 1; ++i) {

       if (arr[i] > arr[i + 1]) {

           return false;

           }

   }

   return true;

}



int main() {

   int size;

   cout << "Enter the size of the array: ";

   cin >> size;



   int arr[size];

   cout << "Enter the elements of the array:\n";
```

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

        cout << "Element " << i + 1 << ": ";

        cin >> arr[i];

    }



    if (isAscending(arr, size)) {

        cout << "The array is in ascending order.\n";

    } else {

        cout << "The array is not in ascending order.\n";

    }



    return 0;

}
```

Output (screenshot):



Test Cases (any 2):



Conclusion:

Hence by simply iterating through the array and comparing 2 values, we can find out if an array is in ascending order or not.

Experiment No : 10

Title: Check an array of strings and return it in alphabetical order

Theory:

Using the bubble sort algorithm, we can compare 2 strings, based on their ASCII values, and swap if they need to be swapped, otherwise skip to the next iteration and check.

Code:

```
#include <iostream>

#include <string>

using namespace std;



void bubbleSort(string arr[], int size) {
```

```
    for (int i = 0; i < size - 1; ++i) {
        for (int j = 0; j < size - i - 1; ++j) {
            if (arr[j] > arr[j + 1]) {
                // Swap elements if they are in the wrong order
                string temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    int size;

    // Input the size of the array
    cout << "Enter the size of the array: ";
    cin >> size;

    // Input the elements of the array of strings
    string arr[size];
    cout << "Enter the elements of the array of strings:\n";
    for (int i = 0; i < size; ++i) {
        cout << "Element " << i + 1 << ": ";
        cin >> arr[i];
    }

    bubbleSort(arr, size);

    cout << "Sorted array in alphabetical order: ";
```

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

        cout << arr[i] << " ";

    }

    cout << endl;



    return 0;

}
```

Output (screenshot):

```
cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 11.cpp -o 11 && "/Users/premthatikonda/Documents/LabManualC++/"11
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 11.cpp -o 11 && "/Users/premthatikonda/Documents/LabManualC++/"11
Enter the size of the array: 5
Enter the elements of the array of strings:
Element 1: apple
Element 2: cat
Element 3: dog
Element 4: egg
Element 5: ball
Sorted array in alphabetical order: apple ball cat dog egg
```

Test Cases (any 2):

```
cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 11.cpp -o 11 && "/Users/premthatikonda/Documents/LabManualC++/"11
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 11.cpp -o 11 && "/Users/premthatikonda/Documents/LabManualC++/"11
Enter the size of the array: 3
Enter the elements of the array of strings:
Element 1: apple
Element 2: ball
Element 3: cat
Sorted array in alphabetical order: apple ball cat
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 11.cpp -o 11 && "/Users/premthatikonda/Documents/LabManualC++/"11
Enter the size of the array: 5
Enter the elements of the array of strings:
Element 1: pen
Element 2: pencil
Element 3: paper
Element 4: post
Element 5: prick
Sorted array in alphabetical order: paper pen pencil post prick
```

Conclusion:

Hence by accessing ASCII values of characters, we can simply compare 2 strings too and sort an array of strings in ascending order.

Experiment No : 12

Title: Calculate the sum of elements in each row of a matrix

Theory:

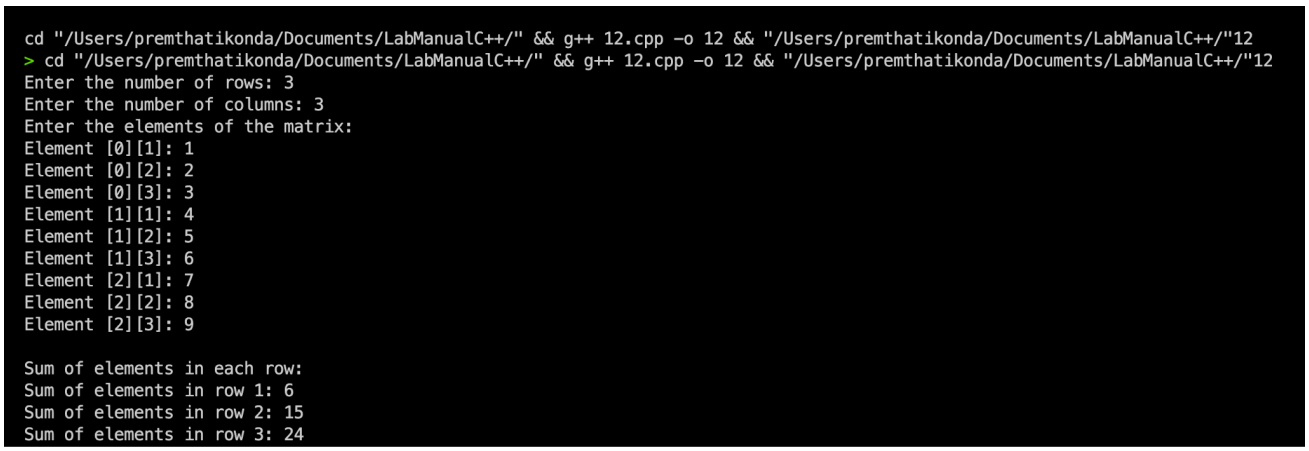
Iterate through each row of a matrix individually based on the number of columns, and assign the greatest number in the row to a variable and print it.

Code:

```cpp
#include <iostream>

using namespace std;



int main() {

   int rows, cols;



   // Input the number of rows and columns for the matrix

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

   cin >> rows;



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

   cin >> cols;



   int matrix[rows][cols];

   cout << "Enter the elements of the matrix:\n";

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

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

           cout << "Element [" << i << "][" << j << "]: ";

           cin >> matrix[i][j];
```

```
        }

    }



    cout << "\nSum of elements in each row:\n";

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

        int sum = 0;

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

            sum += matrix[i][j];

        }

        cout << "Sum of elements in row " << i + 1 << ": " << sum << endl;

    }



    return 0;

}
```

Output (screenshot):



Test Cases (any 2):

```
cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 12.cpp -o 12 && "/Users/premthatikonda/Documents/LabManualC++/"12
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 12.cpp -o 12 && "/Users/premthatikonda/Documents/LabManualC++/"12
Enter the number of rows: 3
Enter the number of columns: 3
Enter the elements of the matrix:
Element [0][1]: 1
Element [0][2]: 2
Element [0][3]: 3
Element [1][1]: 4
Element [1][2]: 5
Element [1][3]: 6
Element [2][1]: 7
Element [2][2]: 8
Element [2][3]: 9

Sum of elements in each row:
Sum of elements in row 1: 6
Sum of elements in row 2: 15
Sum of elements in row 3: 24
```

```
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 12.cpp -o 12 && "/Users/premthatikonda/Documents/LabManualC++/"12
Enter the number of rows: 2
Enter the number of columns: 4
Enter the elements of the matrix:
Element [0][0]: 5
Element [0][1]: 5
Element [0][2]: 5
Element [0][3]: 5
Element [1][0]: 10
Element [1][1]: 10
Element [1][2]: 10
Element [1][3]: 10

Sum of elements in each row:
Sum of elements in row 1: 20
Sum of elements in row 2: 40
```

Conclusion:

Hence by simply iterating through each row and adding the values to 1 variable, we can get the sum of each row's elements.

Experiment No. : 13

Title: Generate all possible permutations of a string

Theory: Take string input; Convert the string into an array of characters; Pass it into a function that swaps two consecutive values; this swapping must happen recursively.

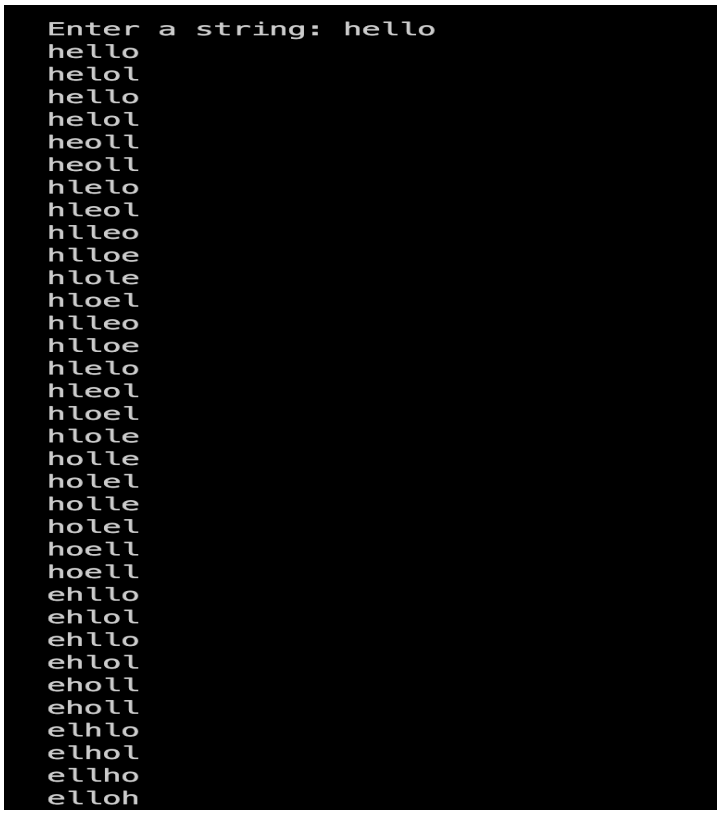
Code:

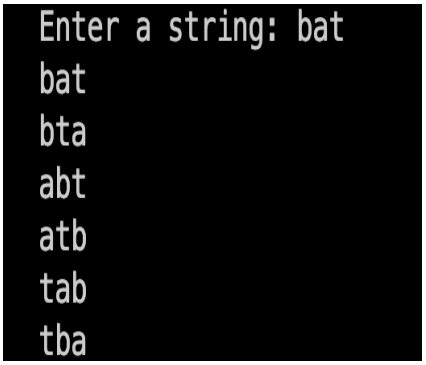
```cpp
#include <iostream>


void swap(char& a, char& b) {
   char temp = a;
   a = b;
   b = temp;
}


void generatePermutations(char str[], int start, int end) {
   if (start == end) {
       std::cout << str << '\n';
       return;
   }

   for (int i = start; i <= end; ++i) {
       swap(str[start], str[i]);
       generatePermutations(str, start + 1, end);
       swap(str[start], str[i]); // Backtrack
   }
```

```
}

int main() {

    std::string input;



    // Input the string

    std::cout << "Enter a string: ";

    std::cin >> input;



    // Convert string to a character array

    char str[input.length() + 1];

    strcpy(str, input.c_str());



    // Generate and print permutations

    generatePermutations(str, 0, input.length() - 1);



    return 0;

}
```

Output(Screenshot):

Test Cases(any 2 screenshot):





Conclusion:

Hence by mainly using recursion, we can generate all possible permutations of a string.

Experiment No : 14

Title:

Print pattern:

*****

* *

* *

* *

*****

Theory:

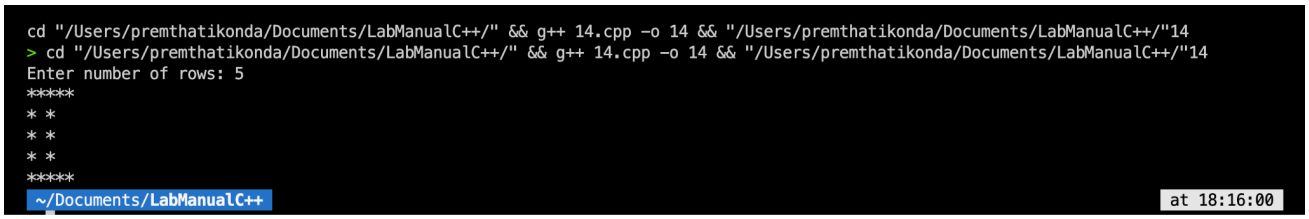
Use nested loops and a condition to check if ‘i’ is either 0 or the number of rows, or if ‘j’ is 0 or 2, then print “*” else print “ ”.

Code:

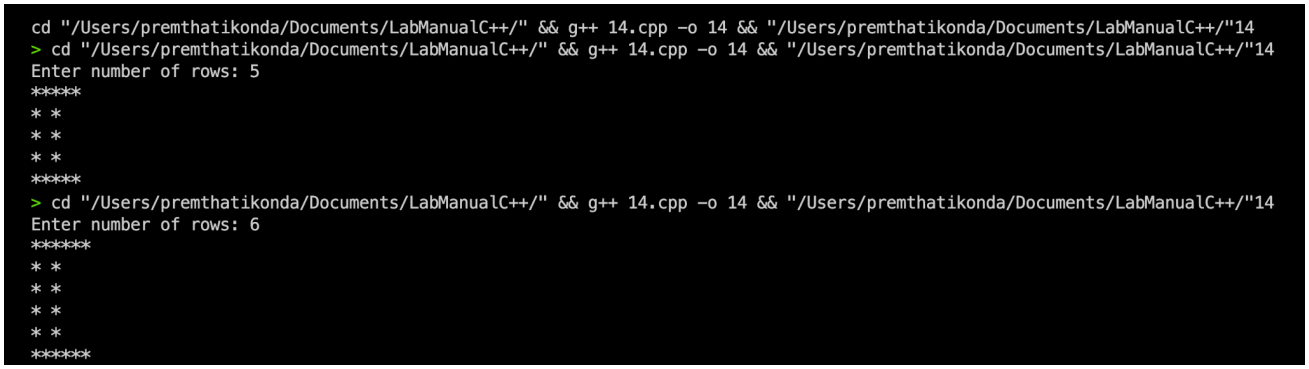
```
#include <iostream>

using namespace std;



int main(){

   int n;

   cout << "Enter number of rows: ";

   cin >> n;



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

       for(int j = 0; j < n; j++){

           if (i==0 || i == n-1 || j== 0 || j == 2){

               cout << "*";
```

```
            }
            else{
                cout << " ";
            }
        }
        cout << endl;
    }

    return 0;}
```

Output (screenshot):



Test Cases (any 2):



Conclusion:

Hence pattern printed using nested for loops and a conditional statement determining whether to print * or space(“ ”).

Experiment No : 15

Title:

Print pattern:

1

232

34543

4567654

34543

232

1

Theory:

Use nested loops to print the first half of the pattern first, in which the first loop prints ascending numbers, the second loop for descending.

Then another nested loop to print the bottom part of the pattern in which, first loop to print ascending part, next loop to print descending.

Code:

```
#include <iostream>

using namespace std;



int main(){

   int n1=5;



   for (int i = 1; i <= n1; i++){

       for (int j = i; j <= (2*i)-1; j++){
```

```
            cout << j;

        }

        for (int j = (2*i)-2; j >=i; j--){

            cout << j;

        }

        cout << endl;

    }


    for (int i = n1 - 1; i >= 1; i--){

        for (int j = i; j <= 2*i - 1; j++){

            cout << j;

        }

        for (int j = (2*i)-2; j >=i; j--){

            cout << j;

        }

        cout << endl;

    }

    return 0;

}
```

Output (screenshot):

```
cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 15.cpp -o 15 && "/Users/premthatikonda/Documents/LabManualC++/"15
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 15.cpp -o 15 && "/Users/premthatikonda/Documents/LabManualC++/"15
1
232
34543
4567654
34543
232
1
 ~/Documents/LabManualC++                                                    at 18:34:51
```

Test Cases (any 2):

```
cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 15.cpp -o 15 && "/Users/premthatikonda/Documents/LabManualC++/"15
> cd "/Users/premthatikonda/Documents/LabManualC++/" && g++ 15.cpp -o 15 && "/Users/premthatikonda/Documents/LabManualC++/"15
1
232
34543
4567654
567898765
4567654
34543
232
1
```

Conclusion:

Using nested loops, the above pattern has been printed.

Experiment No : 16

Title: Inventory Management system using class and objects

Theory:

Add products into the inventory by creating objects of them of the class Product, by passing in the parameters id, name, price and quantity during object creation.

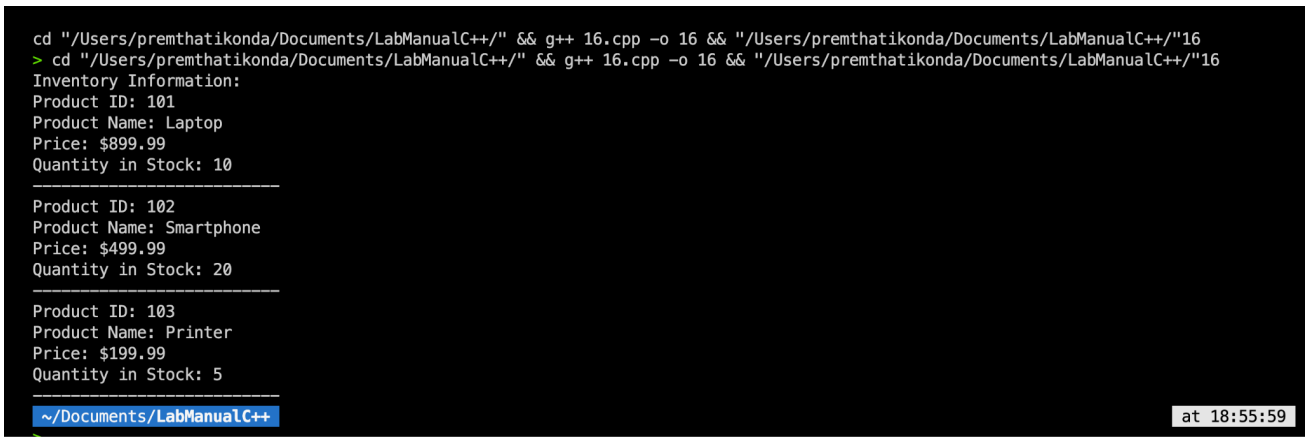
Code:

```cpp
#include <iostream>

#include <string>

using namespace std;
```

```cpp
class Product {
private:
    int productId;
    string productName;
    double price;
    int quantityInStock;

public:
    // Parameterized constructor
    Product(int id, const string& name, double prc, int qty) {
        productId = id;
        productName = name;
        price = prc;
        quantityInStock = qty;
    }

    void displayProductInfo() const {
        cout << "Product ID: " << productId << endl
             << "Product Name: " << productName << endl
             << "Price: $" << price << endl
             << "Quantity in Stock: " << quantityInStock << endl
             << "--------------------------" << endl;
    }
};

int main() {
    // Creating and displaying products
    Product product1(101, "Laptop", 899.99, 10);
    Product product2(102, "Smartphone", 499.99, 20);
```

```cpp
    Product product3(103, "Printer", 199.99, 5);



    cout << "Inventory Information:\n";

    product1.displayProductInfo();

    product2.displayProductInfo();

    product3.displayProductInfo();



    return 0;

}
```

Output (screenshot):



Conclusion:

Hence, using class and objects, we've made a simple inventory management system in c++.

Experiment No : 17

Title: Manage student records

Theory:

Create a class with variables name, rollNo, marks and set them to private.

Create a public constructor to add students, display info of students, and to add students.

Code:

```
#include <iostream>

using namespace std;



int MAX_STUDENTS = 3;

int numStudents = 3;

// Student class representing individual student records

class Student {

private:

   string name;

   int rollNumber;

   float marks;



public:

   Student(){}
```

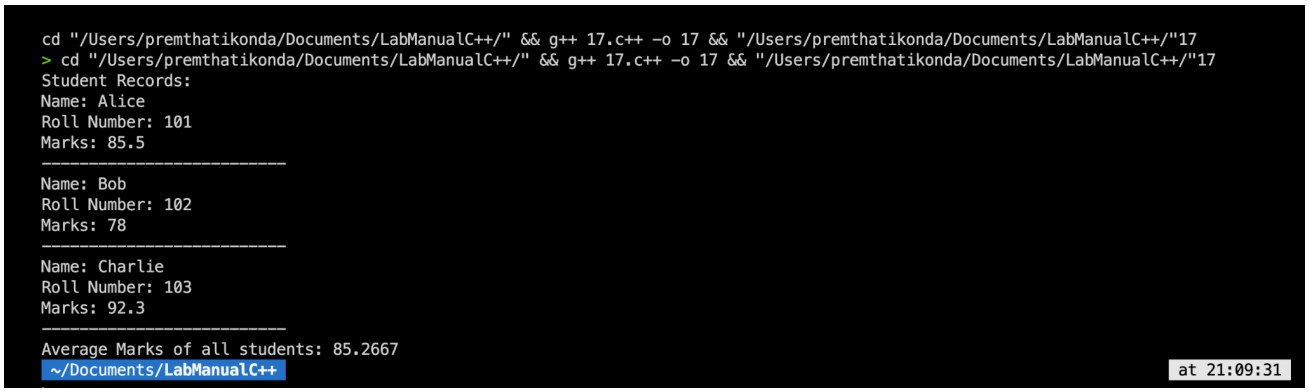
```
    Student(string n, int roll, float m)
        : name(n), rollNumber(roll), marks(m) {}

    // Method to display student details
    void displayDetails() const {
        cout << "Name: " << name << endl;
        cout << "Roll Number: " << rollNumber << endl;
        cout << "Marks: " << marks << endl;
        cout << "--------------------------" << endl;
    }

    // Getters to access private attributes
    string getName() const { return name; }
    int getRollNumber() const { return rollNumber; }
    float getMarks() const { return marks; }
};

// Function to calculate the average marks of all students
float calculateAverageMarks(const Student students[], int numStudents) {
    float totalMarks = 0;
    for (int i = 0; i < numStudents; ++i) {
        totalMarks += students[i].getMarks();
    }
    return numStudents > 0 ? totalMarks / numStudents : 0;
}

void addStudent(Student students[], int& numStudents, const string& name, int roll, float
marks) {
    MAX_STUDENTS++;
    // ++numStudents;
```

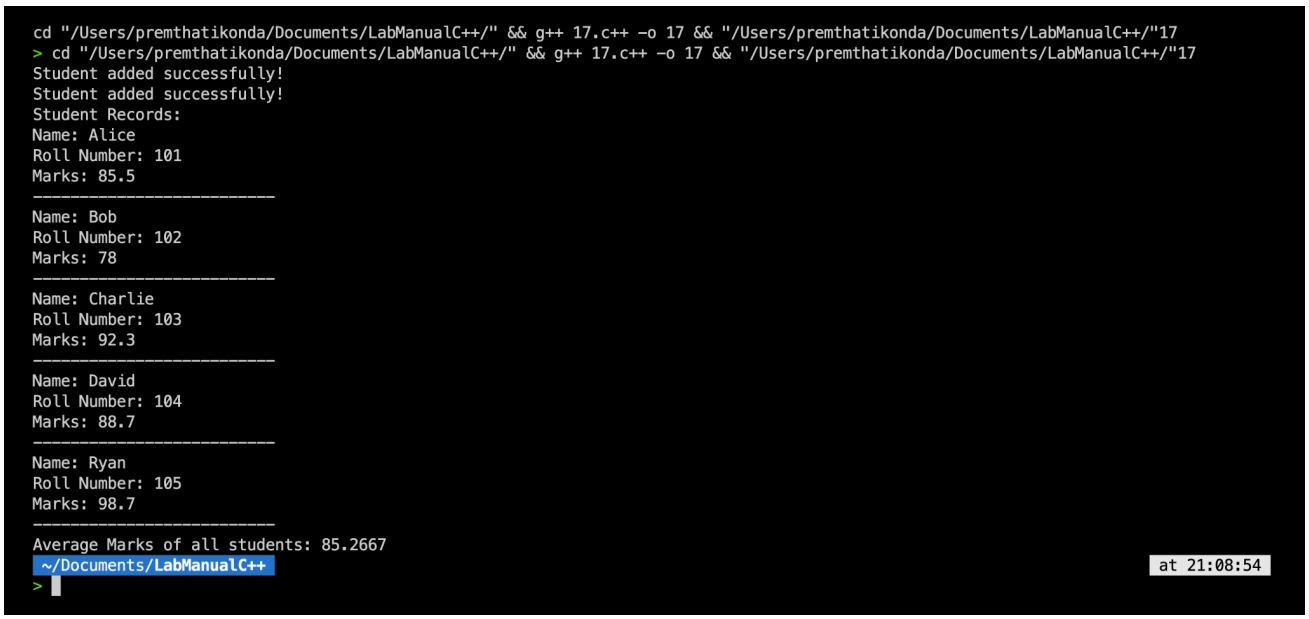
```
    if (numStudents < MAX_STUDENTS) {
        students[numStudents++] = Student(name, roll, marks);
        cout << "Student added successfully!\n";
    }
    else {
        cout << "Cannot add more students. Maximum limit reached.\n";
    }
}



int main() {
    Student studentRecords[MAX_STUDENTS];

    // Adding new students to the record system
    studentRecords[0] = Student("Alice", 101, 85.5);
    studentRecords[1] = Student("Bob", 102, 78.0);
    studentRecords[2] = Student("Charlie", 103, 92.3);

    addStudent(studentRecords, numStudents, "David", 104, 88.7);
    addStudent(studentRecords, numStudents, "Ryan", 105, 98.7);

    cout << "Student Records:\n";
    for (int i = 0; i < MAX_STUDENTS; ++i) {
        studentRecords[i].displayDetails();
    }

    float averageMarks = calculateAverageMarks(studentRecords, 3);
    cout << "Average Marks of all students: " << averageMarks << endl;

    return 0;
```

```
}
```

Output (screenshot):

Test Cases (any 2):





Conclusion:

Hence we can conclude that this code can manage student records by using objects to add students, and the methods to display information and add more students using a function instead of using the constructor.

Experiment No : 18

Title: Basic Calculator

Theory:

Use methods in class to perform the operations like +, -, *, /, %.

Take number input, then operator input and using switch case based on the operator, function call the required methods through the class's object.

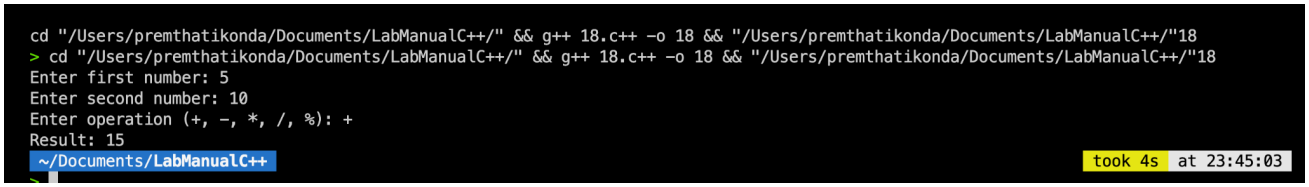
Code:

```cpp
#include <iostream>

using namespace std;


// Calculator class with methods for basic operations

class Calculator{

   public:


       double add(double a, double b) {

           return a + b;

       }


       double subtract(double a, double b) {

           return a - b;

       }


       double multiply(double a, double b) {

           return a * b;

       }


       double divide(double a, double b) {

           if (b != 0) {
```

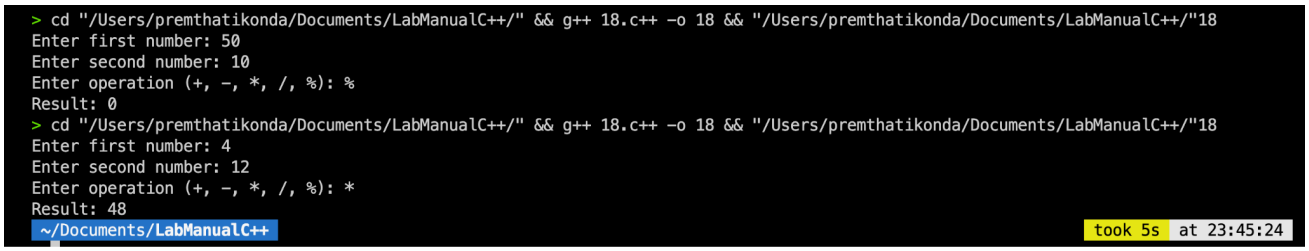
```
            return a / b;
        } else {
            cout << "Error: Division by zero.\n";
            return 0;
        }
    }

    double modulus(int a, int b){
        if (b == 0) {
            cout << "Error: Cannot find the remainder of division by zero.\n";
            return 0;
        }
        else{
            return a % b;
        }
    }
};

int main()
{
   Calculator calculator;

   double num1, num2;
   char operation;

   // Input two numbers
   cout << "Enter first number: ";
   cin >> num1;

   cout << "Enter second number: ";
```

```
    cin >> num2;

    cout << "Enter operation (+, -, *, /, %): ";
    cin >> operation;

    switch (operation) {
        case '+':
            cout << "Result: " << calculator.add(num1, num2) << endl;
            break;
        case '-':
            cout << "Result: " << calculator.subtract(num1, num2) << endl;
            break;
        case '*':
            cout << "Result: " << calculator.multiply(num1, num2) << endl;
            break;
        case '/':
            cout << "Result: " << calculator.divide(num1, num2) << endl;
            break;
        case '%':
            cout << "Result: " << calculator.modulus(num1, num2) << endl;
            break;
        default:
            cout << "Invalid operation.\n";
            break;
    }

    return 0;

}
```

Output (screenshot):



Test Cases (any 2):



Conclusion:

Hence a basic calculator can be made via using function calls through an object of the class containing all the methods to operate on numbers.

Experiment No : 19

Title: Shopping Cart

Theory: The main function runs the program. Make instances of the Product class to create a single product. Then add each product to the Shopping cart, which is an object of the shoppingCart class, in which you can display products in the cart, and also calculate total cost.

Code:

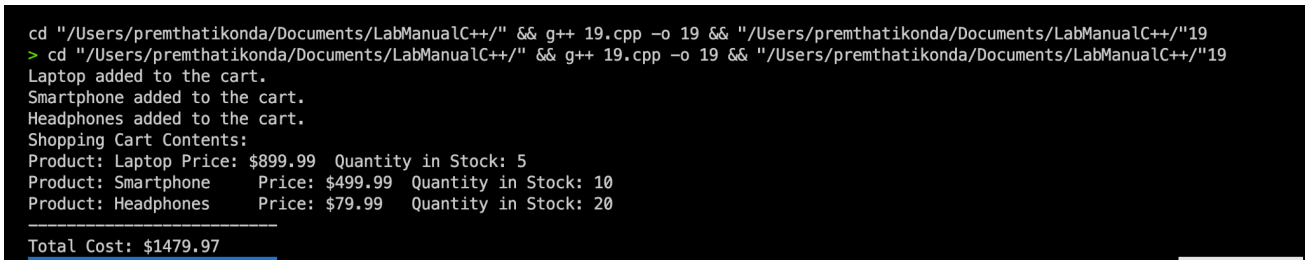
```
#include <iostream>


using namespace std;


const int MAX_PRODUCTS = 100;

const int MAX_CART_SIZE = 10;


class Product {

private:

   string name;

   double price;

   int quantityInStock;


public:

   // Constructor to initialize attributes

   Product(string n, double prc, int qty)

       : name(n), price(prc), quantityInStock(qty) {}


   // Getter methods

   string getName() const { return name; }

   double getPrice() const { return price; }

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

```
    // Method to display product details
    void displayDetails() const {
        cout << "Product: " << name << "\tPrice: $" << price << "\tQuantity in Stock: " << quantityInStock << endl;
    }
};

// ShoppingCart class representing the shopping cart
class ShoppingCart {
private:
    Product* cart[MAX_CART_SIZE];
    int cartSize;

public:
    // Constructor
    ShoppingCart() : cartSize(0) {}

    // Method to add a product to the shopping cart
    void addProduct(Product* product) {
        if (cartSize < MAX_CART_SIZE && product->getQuantityInStock() > 0) {
            cart[cartSize++] = product;
            cout << product->getName() << " added to the cart.\n";
        } else {
            cout << "Cannot add more products to the cart.\n";
        }
    }

    // Method to calculate the total cost of products in the cart
    double calculateTotalCost() {
```

```
        double totalCost = 0;
        for (int i = 0; i < cartSize; ++i) {
            totalCost += cart[i]->getPrice();
        }
        return totalCost;
    }

    // Method to display the contents of the cart
    void displayCartContents() {
        cout << "Shopping Cart Contents:\n";
        for (int i = 0; i < cartSize; ++i) {
            cart[i]->displayDetails();
        }
        cout << "---------------------------\n";
        cout << "Total Cost: $" << calculateTotalCost() << endl;
    }
};

int main() {
    // Creating products
    Product product1("Laptop", 899.99, 5);
    Product product2("Smartphone", 499.99, 10);
    Product product3("Headphones", 79.99, 20);

    // Creating a shopping cart
    ShoppingCart cart;

    // Adding products to the shopping cart
    cart.addProduct(&product1);
    cart.addProduct(&product2);
```

```
    cart.addProduct(&product3);


    // Displaying the contents of the cart
    cart.displayCartContents();


    return 0;
}
```

Output (screenshot):



Conclusion: Shopping cart simulation made using classes and functions, emphasizing on usage of pointers.

Experiment No : 20

Title: Classroom student management

Theory:

**The `Student` class manages student grades, employing a constructor for initialization and a destructor for memory cleanup. Methods include `addGrades` for incorporating grades, `calculateAverageGrade` for computing the average, and `displayStudentInfo` for presenting student details. **

Code:

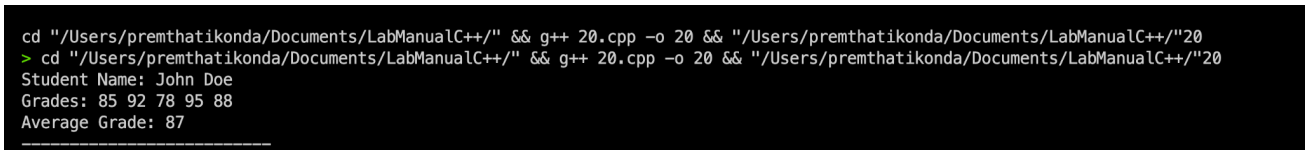
```
#include <iostream>

#include <string>



using namespace std;



class Student {

private:

   string studentName;

   int* grades;

   int numGrades;
```

```
public:
    // Constructor to initialize attributes
    Student(const string& name, int num)
        : studentName(name), numGrades(num) {
        // Allocate memory for grades array
        grades = new int[numGrades];
        // Initialize grades array
        for (int i = 0; i < numGrades; ++i) {
            grades[i] = 0;
        }
    }

    // Destructor to release allocated memory
    ~Student() {
        delete[] grades;
    }

    // Method to add grades
    void addGrades(const int* newGrades) {
        for (int i = 0; i < numGrades; ++i) {
            grades[i] = newGrades[i];
        }
    }

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

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

        return total / numGrades;
    }

    // Method to display student information
    void displayStudentInfo() const {
        cout << "Student Name: " << studentName << endl;
        cout << "Grades: ";
        for (int i = 0; i < numGrades; ++i) {
            cout << grades[i] << " ";
        }
        cout << "\nAverage Grade: " << calculateAverageGrade() << endl;
        cout << "----------------------------\n";
    }
};

int main() {
    // Example usage
    const int numGrades = 5;
    int gradesArray[numGrades] = {85, 92, 78, 95, 88};

    // Creating a student object
    Student student("John Doe", numGrades);

    // Adding grades to the student
    student.addGrades(gradesArray);
```

```cpp
    // Displaying student information
    student.displayStudentInfo();

    return 0;
}
```

Output (screenshot):



Conclusion:

This program has key functionalities, ensuring proper resource management and providing a concise interface for grade-related operations.

Experiment No : 1

Title:

Theory:

Code:

Output (screenshot):

Test Cases (any 2):

Conclusion:

Experiment No : 1

Title:

Theory:

Code:

Output (screenshot):

Test Cases (any 2):

Conclusion:

Experiment No : 1

Title:

Theory:

Code:

Output (screenshot):

Test Cases (any 2):

Conclusion: