ITM SKILLS UNIVERSITY

INSTITUTE OF TECHNOLOGY AND MANAGEMENT
SKILLS UNIVERSITY
KHARGHAR, NAVI MUMBAI

C++ PROGRAMMING LAB

Prepared by:

Name of Student: Ashlin Lee George

Roll No: 11

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 toinitialize 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: Ashlin Lee George
Roll Number: 11

Experiment No: 1

Title: Program to find the Roots of a Quadratic Equation

Theory:

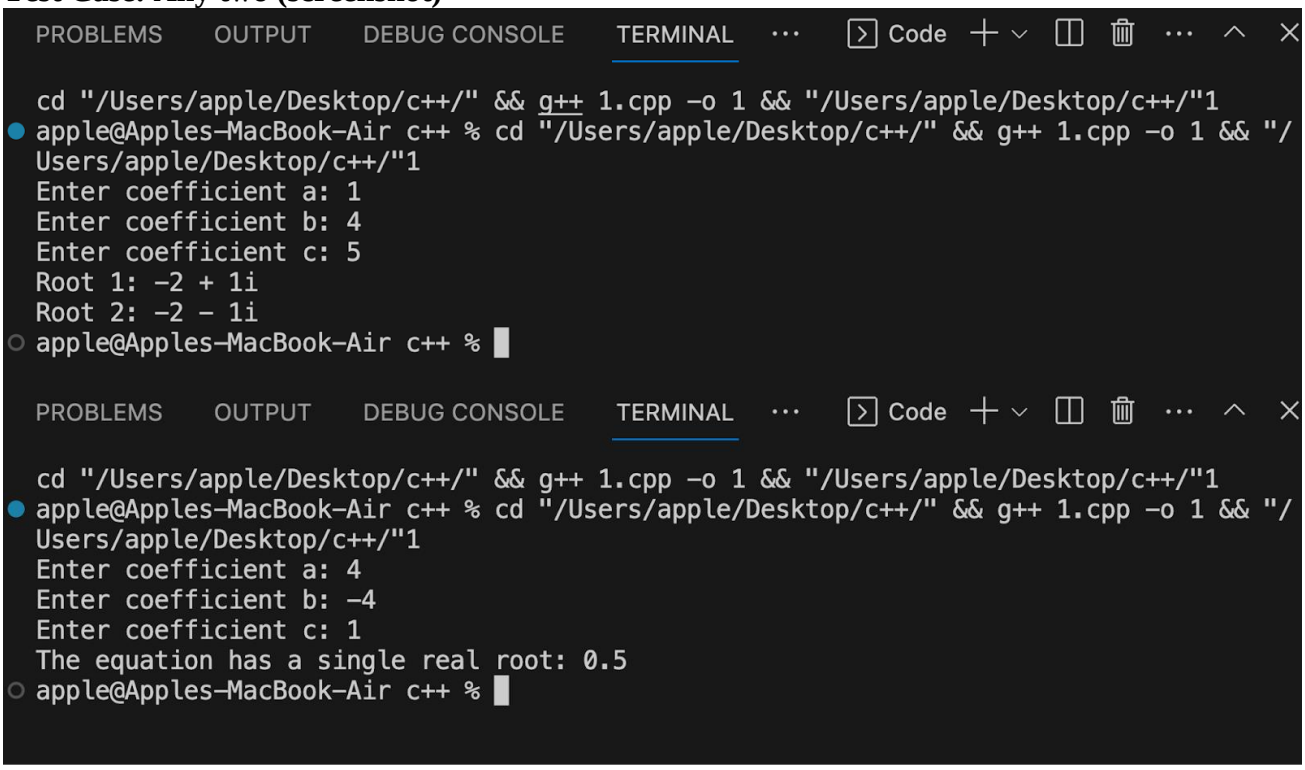
- The program starts by taking user input for the coefficients a,b, and c.
- It calculates the discriminant using the formula b^2 - 4ac.
- Depending on the value of the discriminant, the program calculates and displays the roots.
- If the discriminant is greater than zero, it calculates two distinct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

```
#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:

I've created a code for computing the square roots of a quadratic equation. It relies on the discriminant formula and incorporates test cases along with if-else structures to discern the appropriate formula to utilise.

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:

```
#include <iostream>
using namespace std;
int main() {
double base, exponent;
cout << "Enter the base: ";
cin >> base;
cout << "Enter the exponent: ";
cin >> exponent;
long long int result = 1;
for (int i = 0; i < exponent; ++i) {
result *= base;
}
cout << base << " raised to the power " << exponent << " is: " << result << endl;
return 0;
}
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

By using for loop we can calculate the power of any given number.

Experiment No: 3

Title: Check if a given string is a palindrome

Theory:

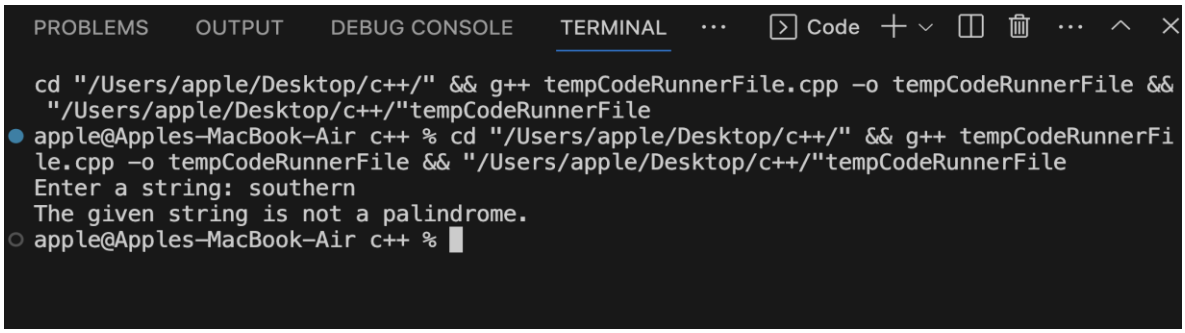
Create a function that takes a string input and traverses it from the first character to the last, comparing pairs—first and last, second and second-to-last, and so on. As long as the characters match, it keeps checking. The moment a mismatch appears, return False. If the loop completes without any mismatches, it means all checked characters were similar, so return True at the end of the function.

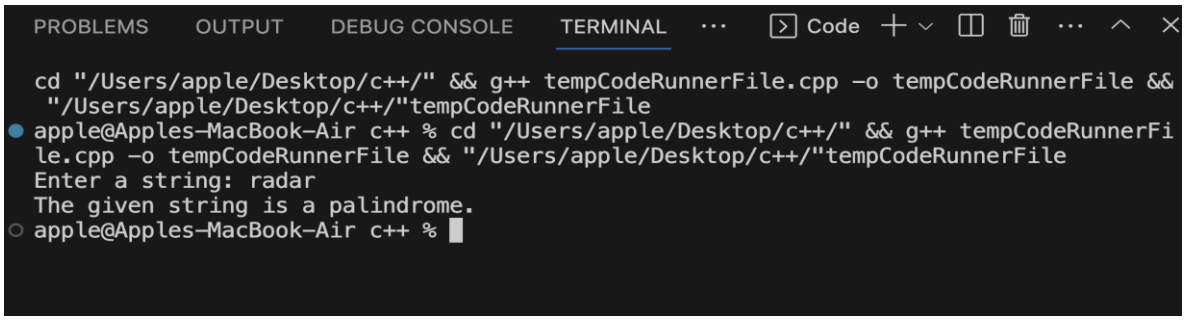
Code:

```
#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]) {
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)

Test Case: Any two (screenshot)





Conclusion:

This program is created to check if an input string is a 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 a switch case structure, we can prompt users to choose their desired action, enabling us to perform functions such as checking their balance, withdrawing or depositing money, or exiting the program based on their selection.

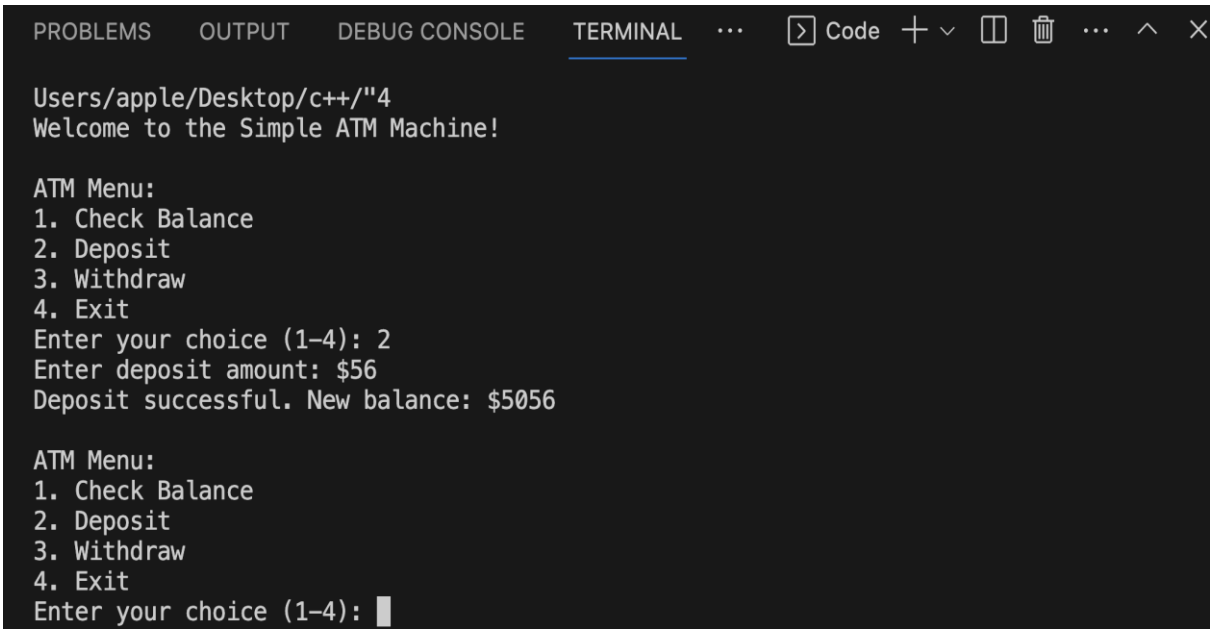
Code:

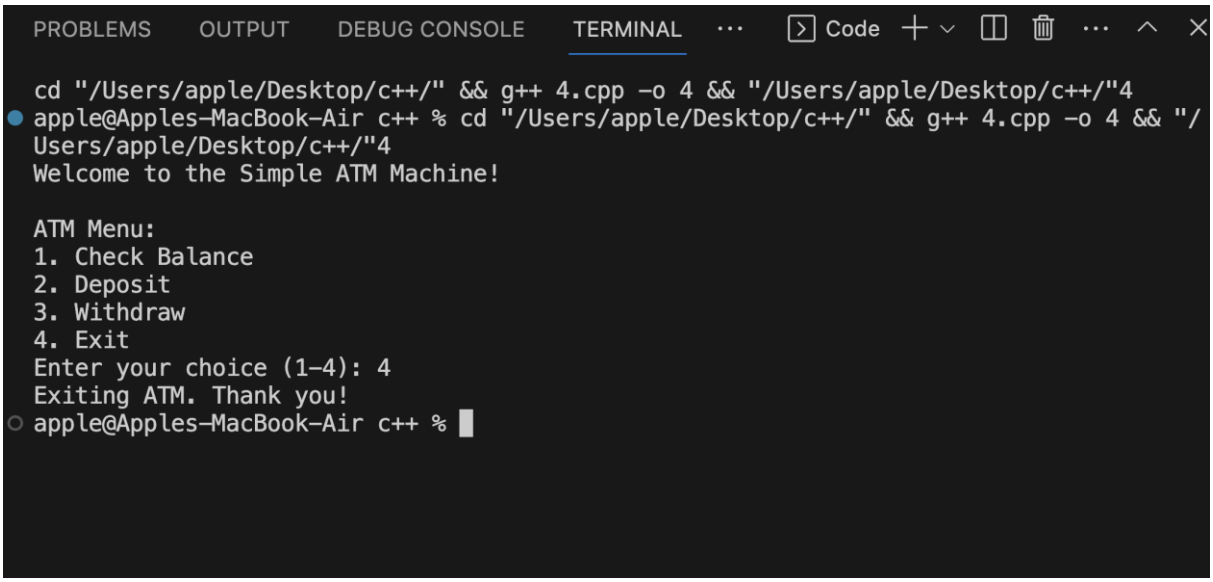
```
#include <iostream>
using namespace std;
void atmMenu() {
double balance = 1000.0;
while (true) {
cout << "\nATM Menu:\n";
cout << "1. Check Balance\n";
cout << "2. Deposit\n";
cout << "3. Withdraw\n";
cout << "4. Exit\n";
cout << "Enter your choice (1-4): ";
int choice;
cin >> choice;
switch (choice) {
case 1:
cout << "Your balance: $" << balance << endl;
break;
case 2:
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:
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:
cout << "Exiting ATM. Thank you!\n";
return;
default:
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)

Test Case: Any two (screenshot)





Conclusion:

Using switch-case we cn create a basic ATM machine.

Experiment No: 5

Title:

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

Theory:

Get three numbers as input. Using nested if-else statement it checks if the num1 is greater than 2nd and 3rd number. Similarly it checks if 2nd is greater than 1st and 3rd. Else it shows 3rd is greatest.

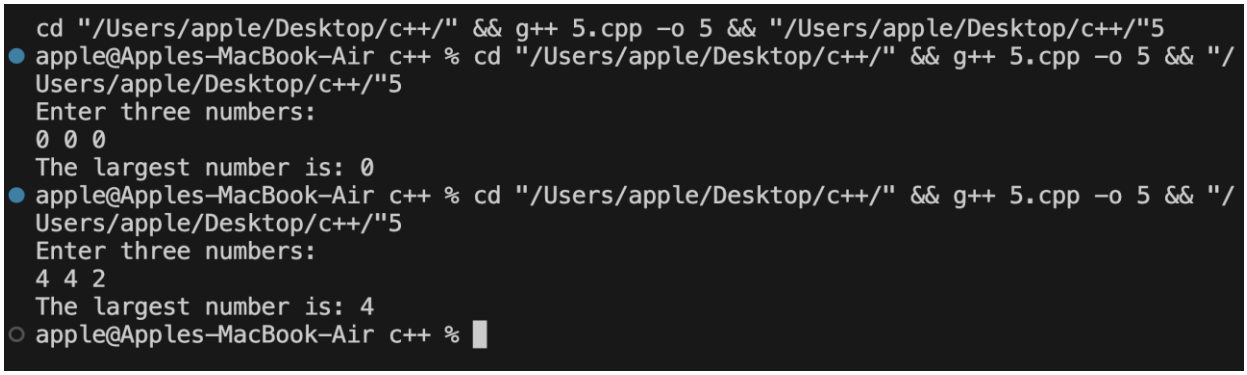
Code:

```
#include <iostream>
using namespace std;
int main() {
double num1, num2, num3;
cout << "Enter three numbers: \n";
cin >> num1 >> num2 >> num3;
if (num1 >= num2){
if (num1 >= num3) {
cout << "The largest number is: " << num1 << endl;
} else {
cout << "The largest number is: " << num3 << endl;
}
}
else{
if (num2 >= num3) {
cout << "The largest number is: " << num2 << endl;
} else {
cout << "The largest number is: " << num3 << endl;
}
}
return 0;
}
```

Output: (screenshot)

```
cd "/Users/apple/Desktop/c++/" && g++ 5.cpp -o 5 && "/Users/apple/Desktop/c++/"5
apple@Apples-MacBook-Air c++ % cd "/Users/apple/Desktop/c++/" && g++ 5.cpp -o 5 && "/
Users/apple/Desktop/c++/"5
Enter three numbers:
5 9 12
The largest number is: 12
apple@Apples-MacBook-Air c++ %
```

Test Case: Any two (screenshot)



Conclusion:

Using a simple nested-if we can determine which number is the largest amongst the three.

Experiment No: 6

Title:

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

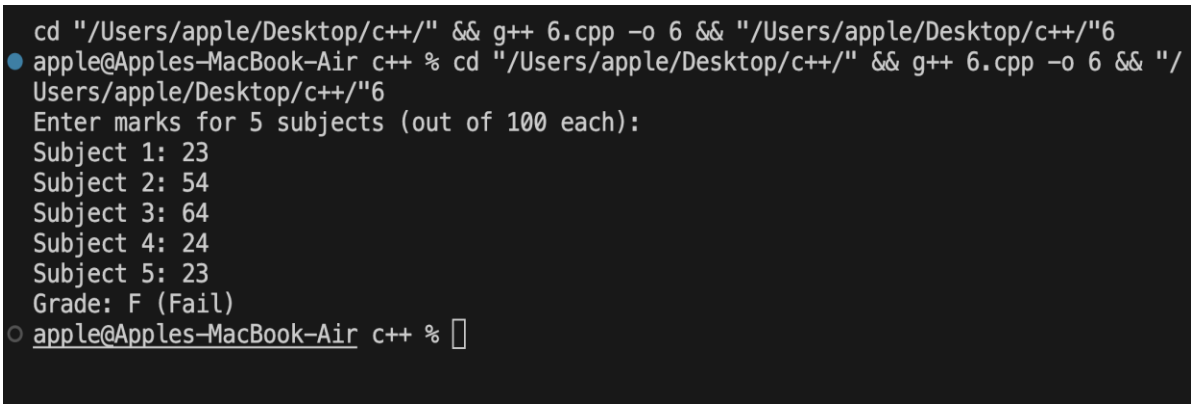
Theory:

Take marks input for the 5 marks, calculate the average by dividing the sum of those 5 marks by 5 (total 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;
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)



Test Case: Any two (screenshot)

Conclusion:

By using simple if-else we can obtain the grades of the student based on the entered subject marks.

Experiment No: 7

Title:

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

Theory:

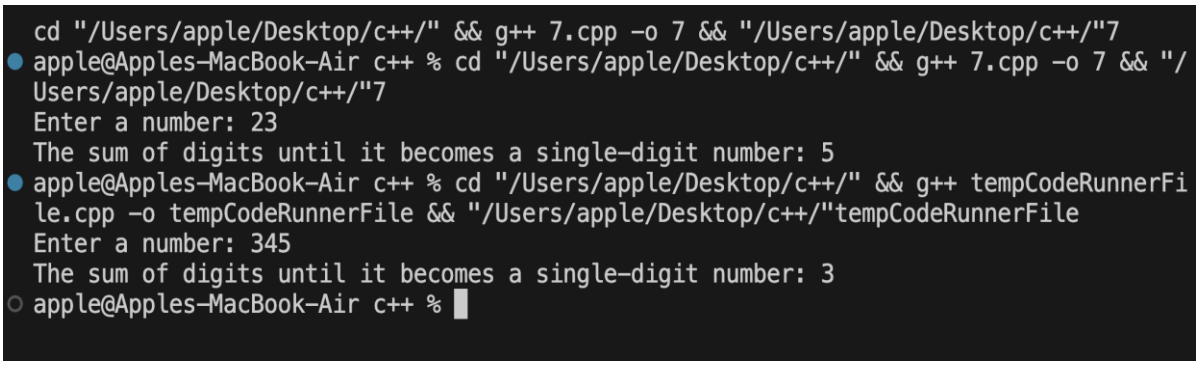
The `sumOfDigits` function uses a `while` loop to traverse through the digits, continuously adding them up. In the main function, this `sumOfDigits` function is repeatedly used until the entered number reduces to less than 10. Finally, the resulting single-digit value is presented to the user.

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 Case: Any two (screenshot)



Conclusion:

Using while loop we can get the sum of digits of the entered number.

Experiment No: 8

Title:

Pascal's triangle using nested loops

Theory:

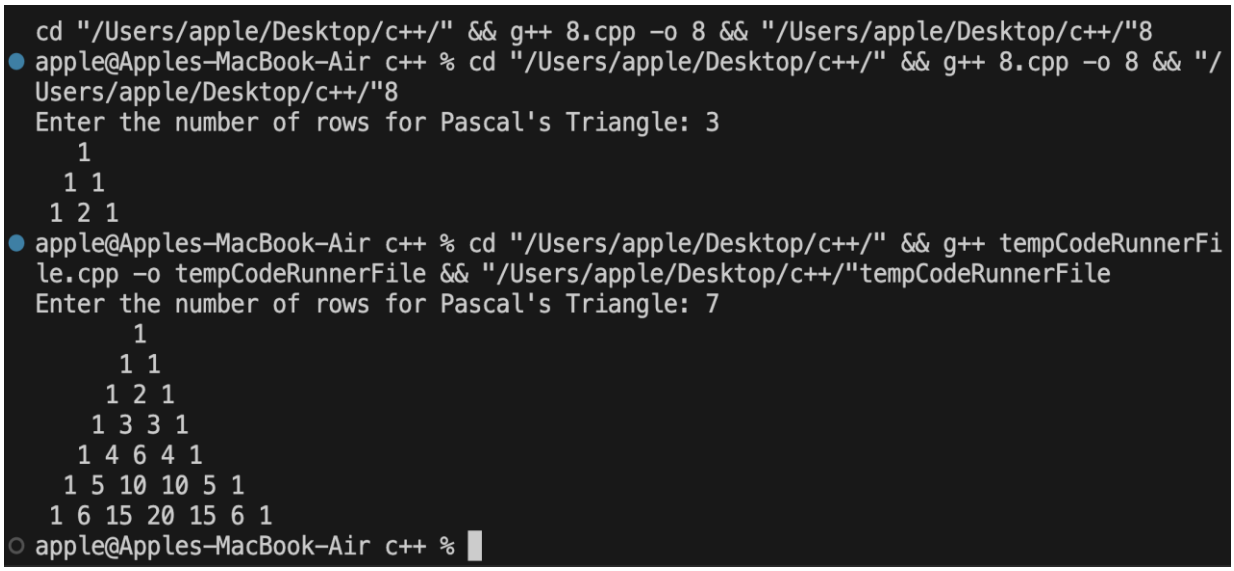
Input the number of rows required for the pattern. Using two nested loops to traverse each row and its coefficients. The inner loop calculates the coefficient to be displayed using the formula n! / k!(n-k)! for each position. Each number will be separated by spaces to form the triangular shape."

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;
for (int j = 0; j < numRows - i; ++j) {
cout << " ";
}
for (int j = 0; j <= i; ++j) {
cout << coefficient << " ";
coefficient = coefficient * (i - j) / (j + 1);
}
cout << endl;
}
return 0;
}
```

Output: (screenshot)

Test Case: Any two (screenshot)



Conclusion:

Using nested loops and the crafted formula, pascal's triangle can be printed for the entered number of rows.

Experiment No: 9

Title:

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

Theory:

Take limit from user. In 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;
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;
for (int i = 1; i <= N; ++i) {
double term = 1.0 * i / factorial(i);
sum += term;
cout << i << "/" << i << "! ";
if (i < N) {
cout << "+ ";
}
}
cout << "\nSum of the series: " << sum << endl;
return 0;
}
```

Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

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

Experiment No: 10

Title:

Check an array of strings and return it in alphabetical order

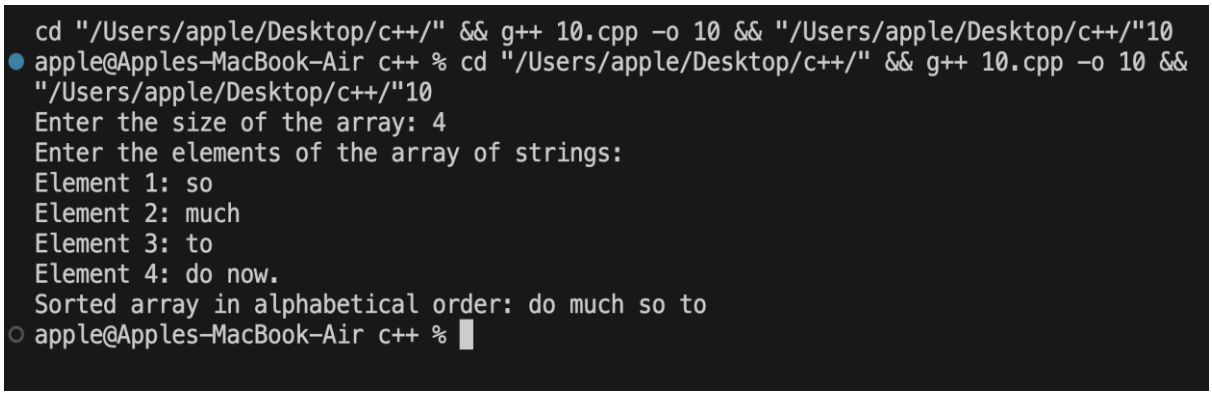
Theory:

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.

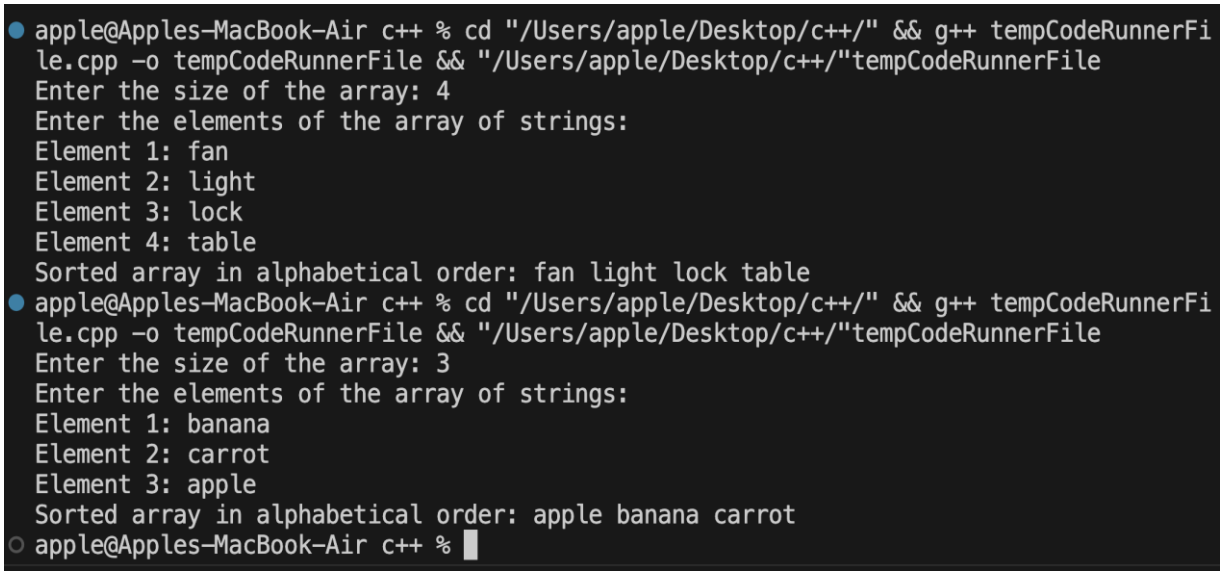
Code:

```
#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]) {
string temp = arr[j];
arr[j] = arr[j + 1];
arr[j + 1] = temp;
}
}
}
}
int main() {
int size;
cout << "Enter the size of the array: ";
cin >> size;
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)



Test Case: Any two (screenshot)



Conclusion:

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

Experiment No: 11

Title:

Check if an array is in ascending order.

Theory:

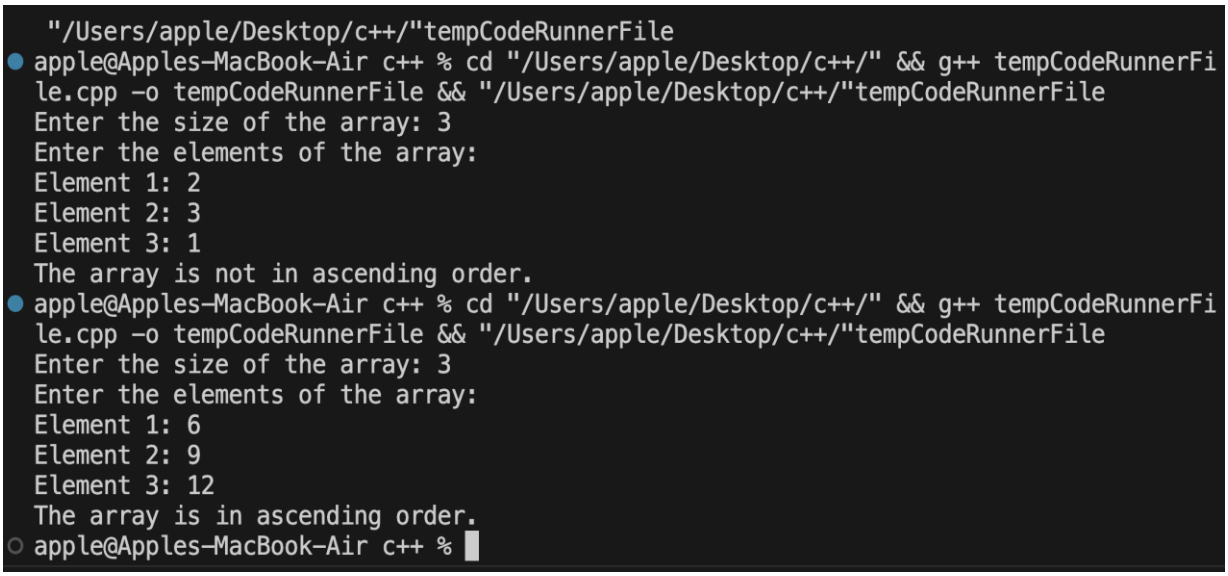
Traverse through the array, checking values against each other. The moment you find a condition that's met, announce that the array isn't in ascending order. Otherwise, once the entire loop finishes, declare that it is indeed in ascending order.

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 Case: Any two (screenshot)



Conclusion:

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: 12

Title:

Calculate the sum of elements in each row of a matrix

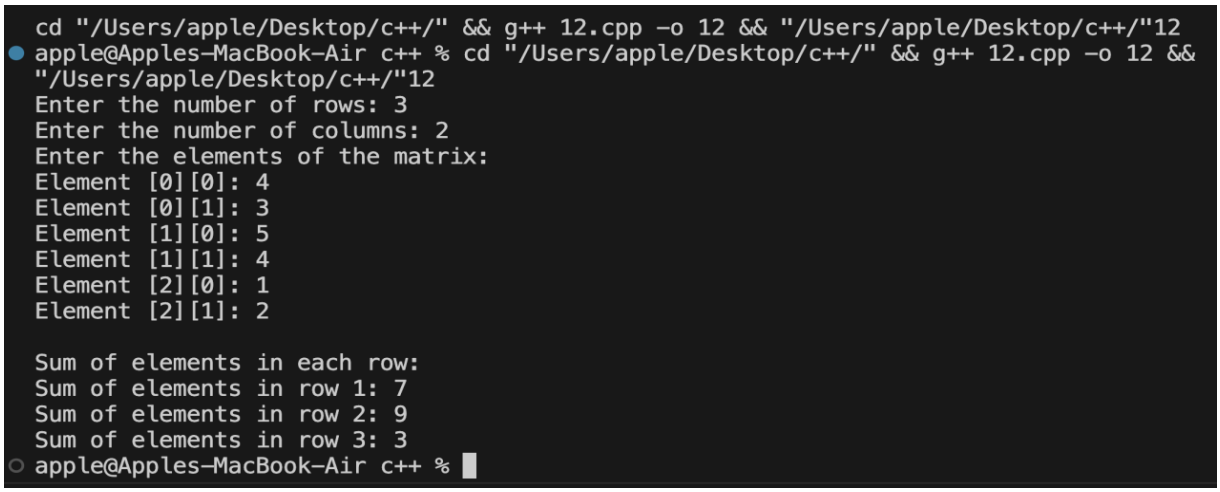
Theory:

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.

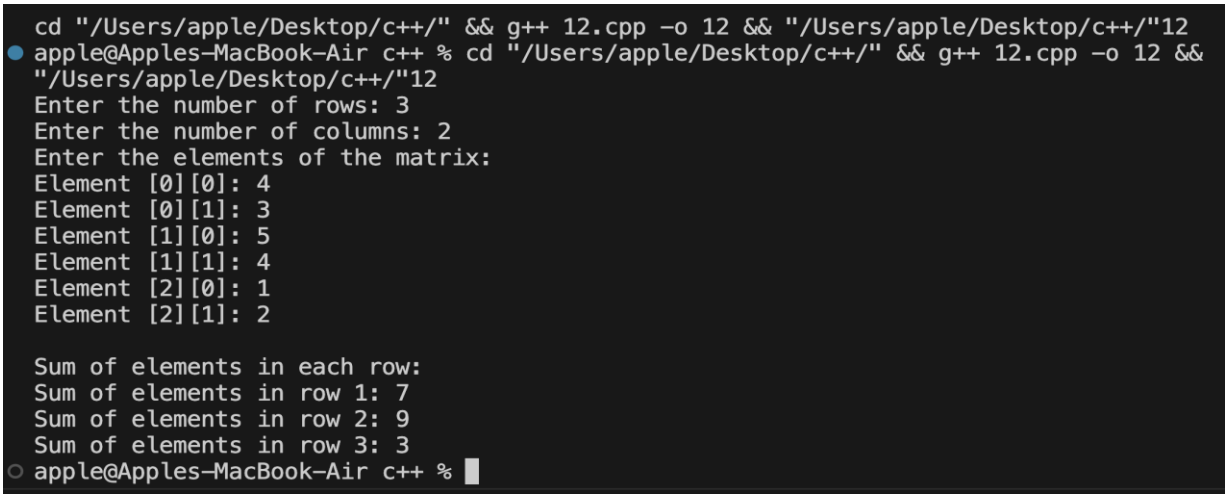
Code:

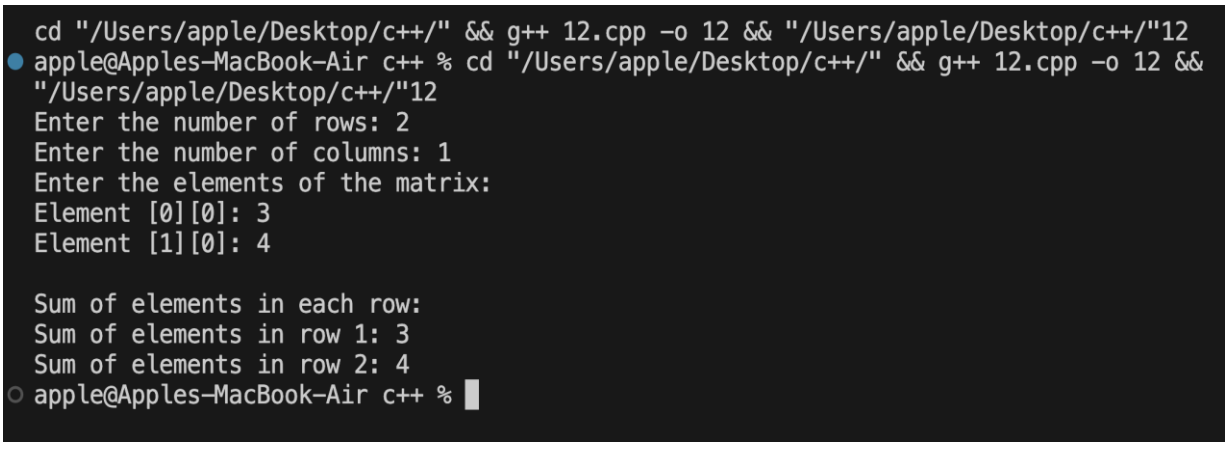
```
#include <iostream>
using namespace std;
int main() {
int rows, cols;
cout << "Enter the number of rows: ";
cin >> rows;
cout << "Enter the number of columns: ";
cin >> cols;
int matrix[rows][cols];
cout << "Enter the elements of the matrix:\n";
for (int i = 0; i < rows; ++i) {
for (int j = 0; j < 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 Case: Any two (screenshot)





Conclusion:

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:

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

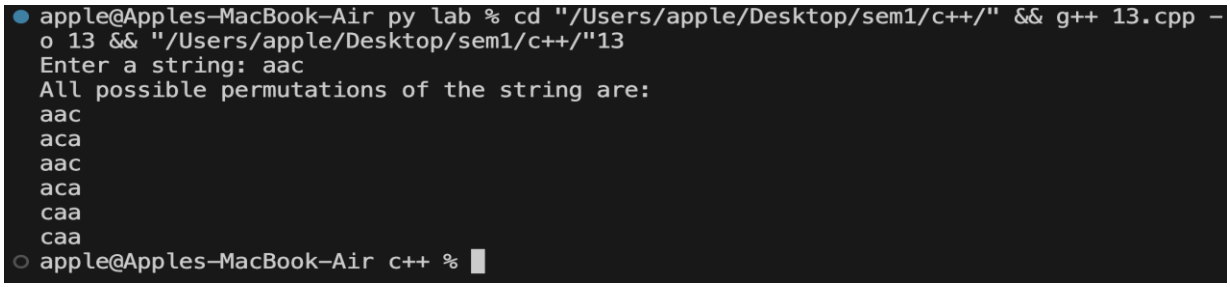
Theory:

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:

```
#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)



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.

Experiment No: 14

Title:

Print pattern:

*****
* *
* *
* *
* ****

Theory:

By using 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=5;
// 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 Case: Any two (screenshot)

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

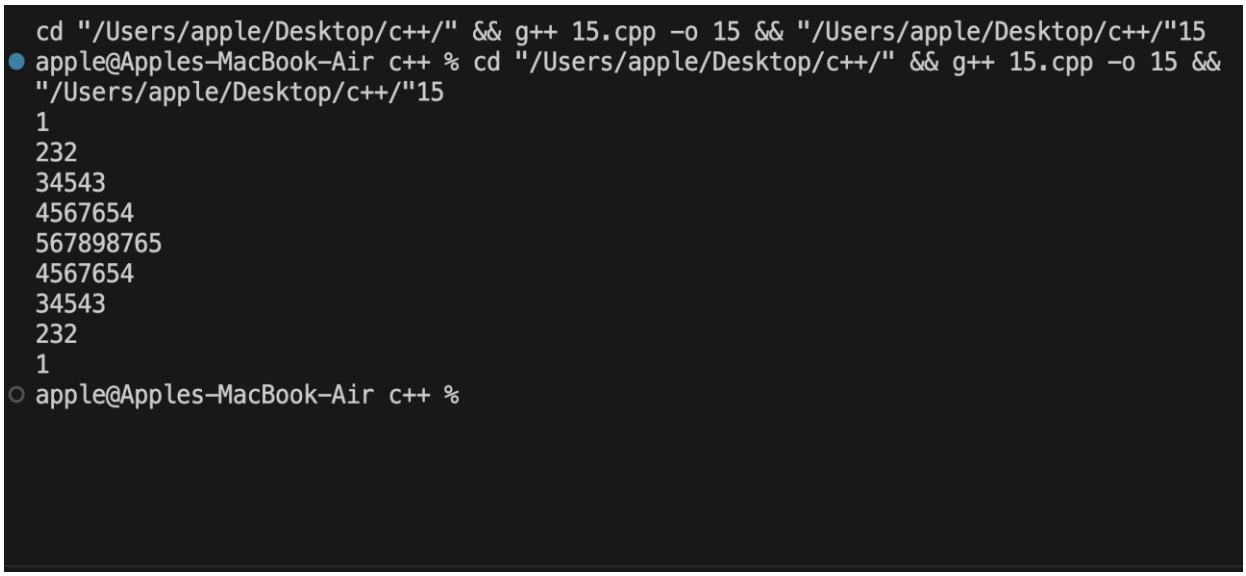
Theory:

Using 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.

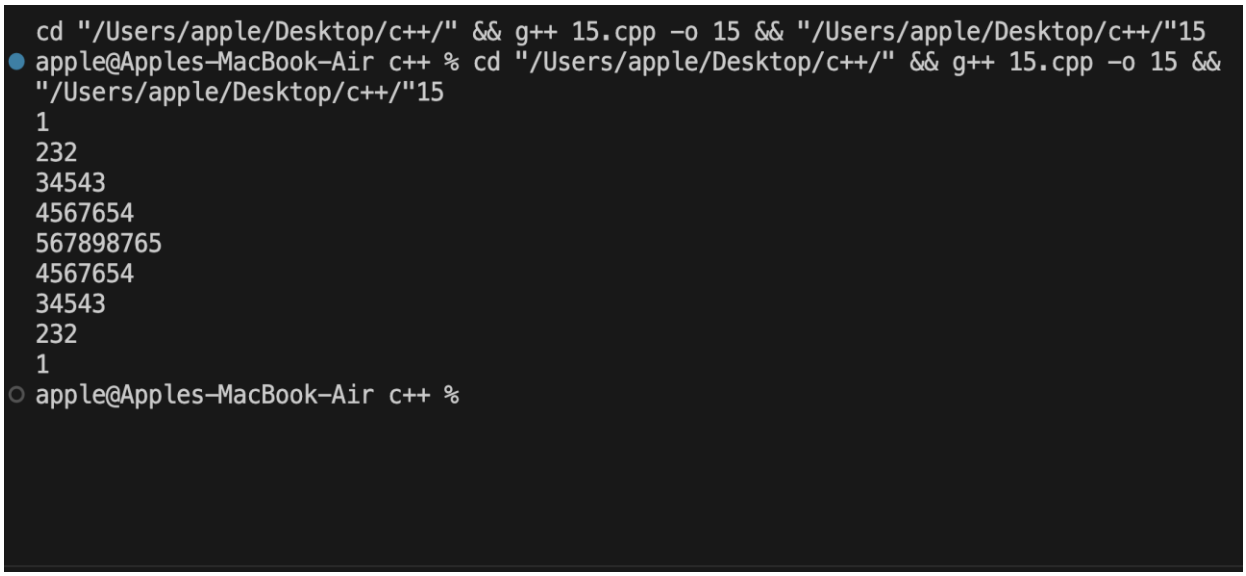
Code:

```
#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)



Test Case: Any two (screenshot)



Conclusion:

Using nested loops, the given 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:

```
#include <iostream>
#include <string>
using namespace std;
class Product {
private:
int productId;
string productName;
double price;
int quantityInStock;
public:
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() {
Product product1(101, "Apple", 3.99, 100);
Product product2(102, "Banana", 4.99, 310);
Product product3(103, "Pear", 2.99, 52);
cout << "Inventory Information:\n";
product1.displayProductInfo();
product2.displayProductInfo();
product3.displayProductInfo();
return 0;
}
```

Output: (screenshot)

```
"/Users/apple/Desktop/c++/"16
Inventory Information:
Product ID: 101
Product Name: Apple
Price: $3.99
Quantity in Stock: 100
--------------------------
Product ID: 102
Product Name: Banana
Price: $4.99
Quantity in Stock: 310
--------------------------
Product ID: 103
Product Name: Pear
Price: $2.99
Quantity in Stock: 52
--------------------------
apple@Apples-MacBook-Air c++ %
```

Test Case: Any two (screenshot)

```
"/Users/apple/Desktop/c++/"16
Inventory Information:
Product ID: 101
Product Name: Apple
Price: $3.99
Quantity in Stock: 100
--------------------------
Product ID: 102
Product Name: Banana
Price: $4.99
Quantity in Stock: 310
--------------------------
Product ID: 103
Product Name: Pear
Price: $2.99
Quantity in Stock: 52
--------------------------
apple@Apples-MacBook-Air c++ %
```

Conclusion:

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

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:

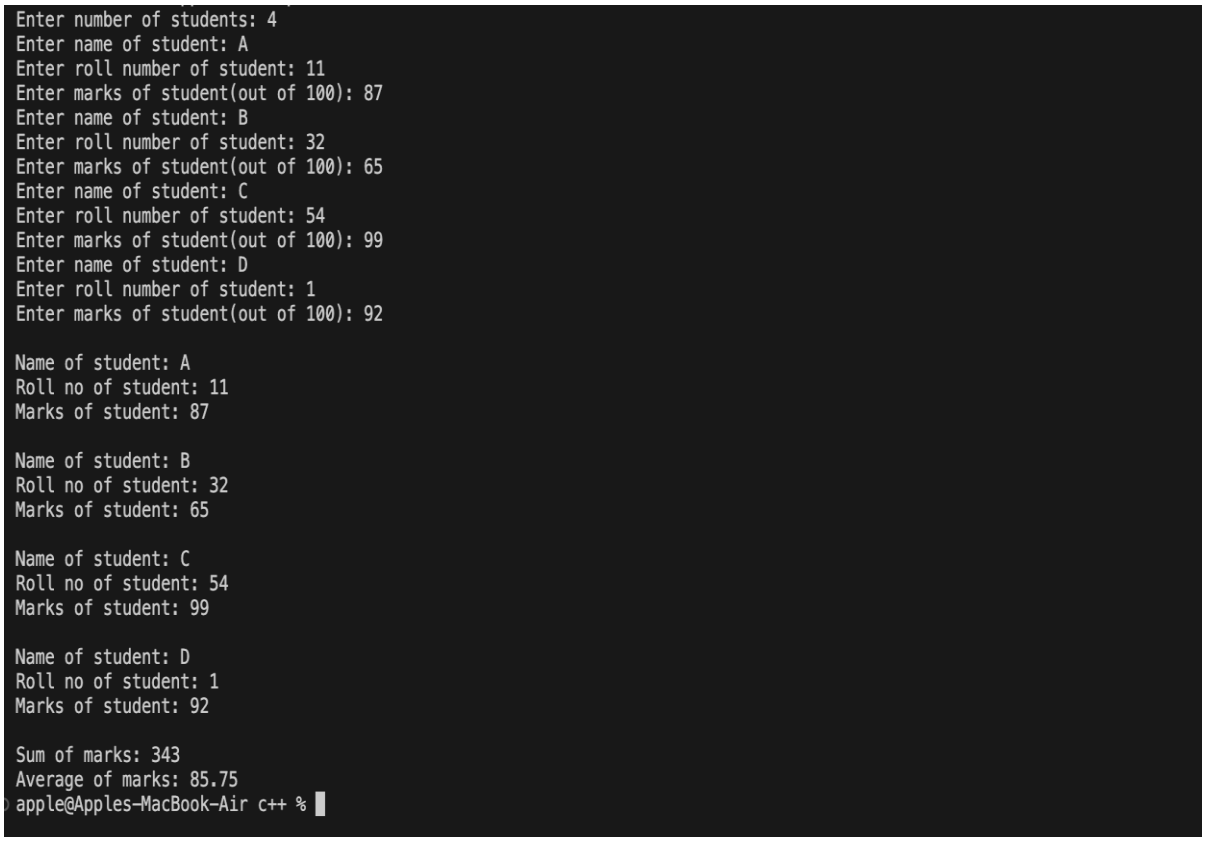
```
#include <iostream>
#include <string>
using namespace std;

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;
    }
};

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:

```
apple@Apples-MacBook-Air py lab % cd "/Users/apple/Desktop/sem1/c++/" && g++ 17.cpp -
o 17 && "/Users/apple/Desktop/sem1/c++/"17
Enter number of students: 2
Enter name of student: Ashlin
Enter roll number of student: 11
Enter marks of student(out of 100): 88
Enter name of student: Ash
Enter roll number of student: 12
Enter marks of student(out of 100): 76

Name of student: Ashlin
Roll no of student: 11
Marks of student: 88

Name of student: Ash
Roll no of student: 12
Marks of student: 76

Sum of marks: 164
Average of marks: 82
apple@Apples-MacBook-Air c++ %
```

Test Case: Any two (screenshot)



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.

Experiment No: 18

Title: Basic Calculator

Theory:

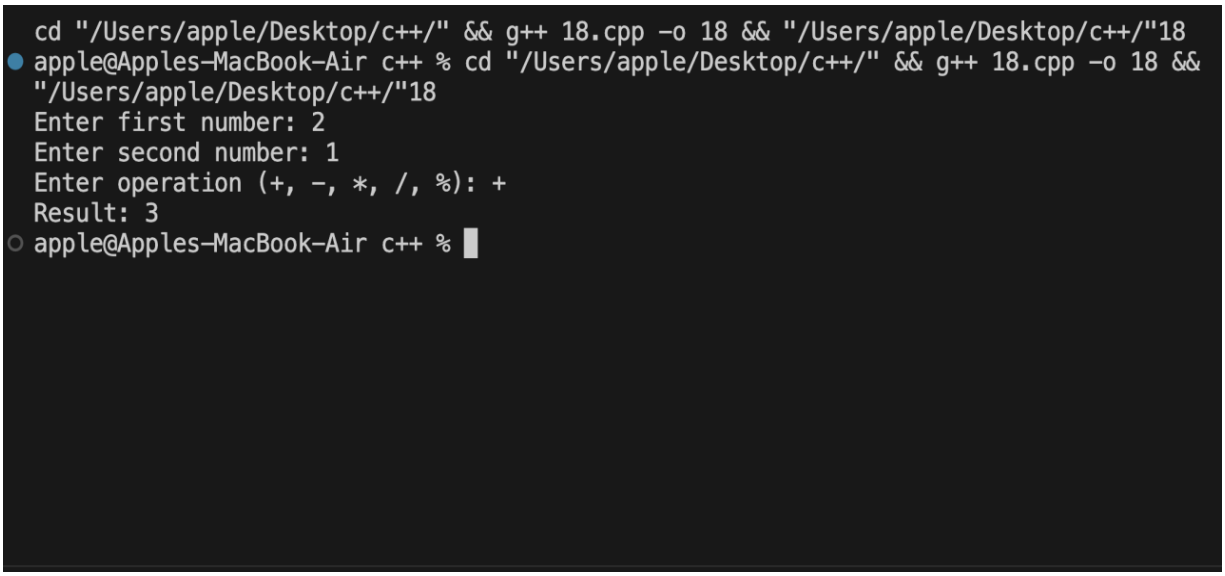
Use functions in class to perform the operations like +, -, * , /, %. Take number input, then operator input and using switch case based on the operator, function call the required methods through the class's object.

Code:

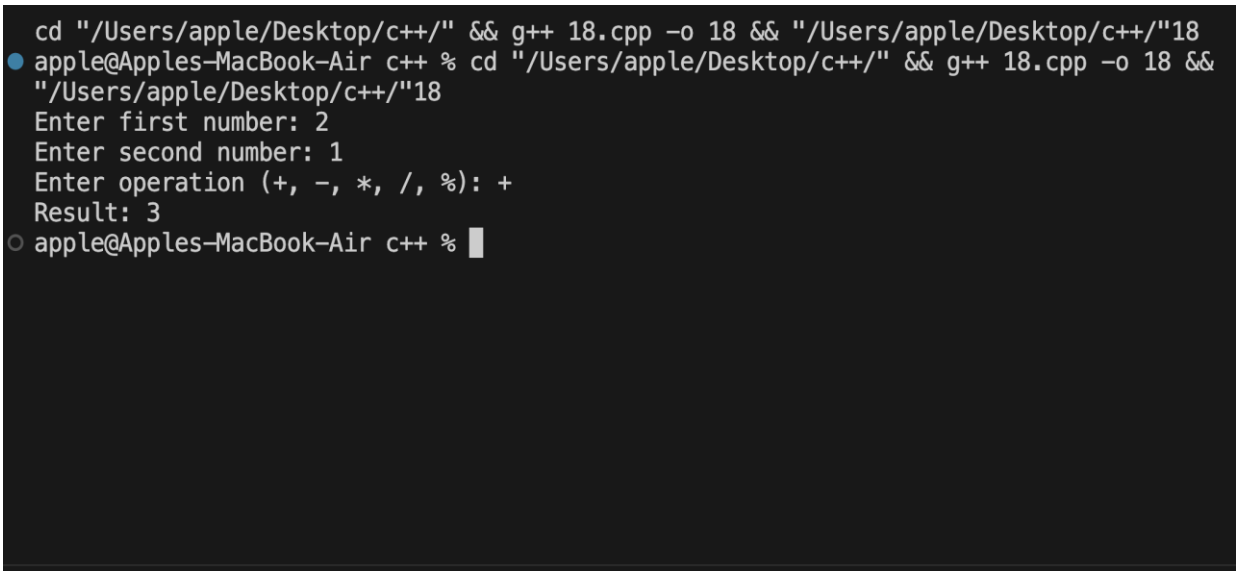
```
#include <iostream>
using namespace std;
class Calculator{
public:
double add(double a, double b) {
return a + b;
}
double subtract(double a, double b) {
return a - b;
}
double multiply(double a, double b) {
return a * b;
}
double divide(double a, double b) {
if (b != 0) {
return a / b;
} else {
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;
```

```
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 Case: Any two (screenshot)



Conclusion:

Basic calculator can be made using function calls through an object of the class containing all the methods to operate on the given numbers.

Experiment No: 19

Title:

Shopping Cart

Theory:

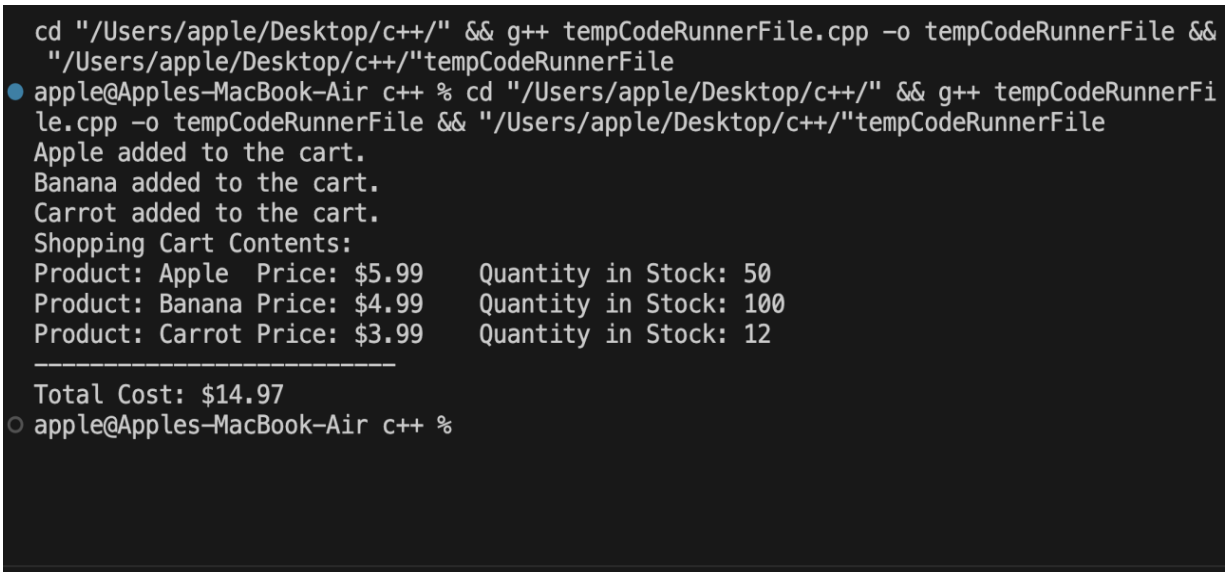
The program's primary function is executed within the main function. It involves creating instances of the Product class to represent individual products. Then, these products are added to a Shopping Cart, which is an object belonging to the shoppingCart class. This cart allows for displaying the contained products and computing the overall total cost.

Code:

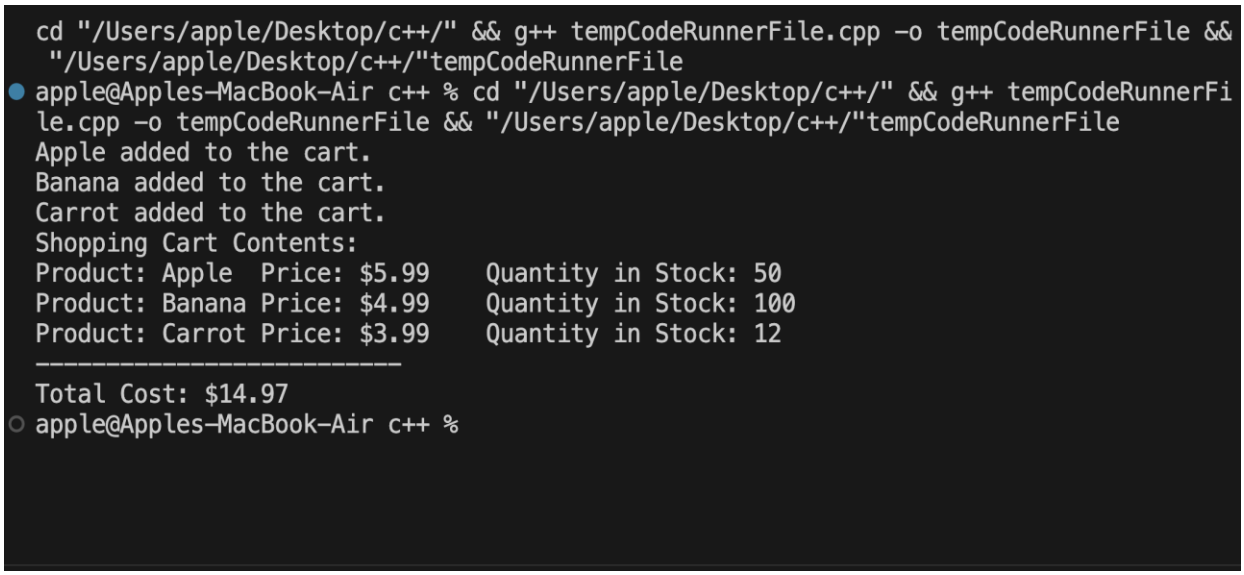
```
#include <iostream>
using namespace std;
const int MAX_PRODUCTS = 100;
const int MAX_CART_SIZE = 10;
class Product {
private:
string name;
double price;
int quantityInStock;
public:
Product(string n, double prc, int qty)
: name(n), price(prc), quantityInStock(qty) {}
string getName() const { return name; }
double getPrice() const { return price; }
int getQuantityInStock() { return quantityInStock; }
void displayDetails() const {
cout << "Product: " << name << "\tPrice: $" << price << "\tQuantity in Stock: " <<
quantityInStock << endl;
}
};
class ShoppingCart {
private:
Product* cart[MAX_CART_SIZE];
int cartSize;
public:
ShoppingCart() : cartSize(0) {}
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";
}
```

```
}
double calculateTotalCost() {
double totalCost = 0;
for (int i = 0; i < cartSize; ++i) {
totalCost += cart[i]->getPrice();
}
return totalCost;
}
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() {
Product product1("Apple", 5.99, 50);
Product product2("Banana", 4.99, 100);
Product product3("Carrot", 3.99, 12);
ShoppingCart cart;
cart.addProduct(&product1);
cart.addProduct(&product2);
cart.addProduct(&product3);
cart.displayCartContents();
return 0;
}
```

Output: (screenshot)



Test Case: Any two (screenshot)



Conclusion:

Create a shopping cart simulation using classes, functions, and pointers as key components. The highlight here is on utilising pointers effectively within the program to enhance its functionality and efficiency.

Experiment No: 20

Title:

Classroom student management

Theory:

The `Student` class is designed to handle student grades and use a constructor for setting up initial values and a destructor for managing memory release. It has methods such as `addGrades` to include new grades, `calculateAverageGrade` to determine the average grade, and `displayStudentInfo` to showcase comprehensive student information.

Code:

```
#include <iostream>
#include <string>
#include <vector>

using namespace std;

class Student {
private:
    string studentName;
    vector<int> grades;

public:
    Student(const string& name) : studentName(name) {
    }

    ~Student() {
    }

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

    double calculateAverageGrade() {
        if (grades.empty()) {
            return 0.0;
        }

        int sum = 0;
        for (int grade : grades) {
            sum += grade;
        }

        return static_cast<double>(sum) / grades.size();
    }
```

```
    void displayStudentInfo() {
        cout << "Student Name: " << studentName << endl;
        cout << "Grades: ";
        for (int grade : grades) {
            cout << grade << " ";
        }
        cout << endl;
        cout << "Average Grade: " << calculateAverageGrade() << endl;
    }
};

int main() {
    string name;
    cout << "Enter student name: ";
    getline(cin, name);

    Student student(name);

    int numOfGrades;
    cout << "Enter the number of grades: ";
    cin >> numOfGrades;

    for (int i = 0; i < numOfGrades; ++i) {
        int grade;
        cout << "Enter grade " << i + 1 << ": ";
        cin >> grade;
        student.addGrade(grade);
    }

    cout << "\nStudent Information:\n";
    student.displayStudentInfo();

    return 0;
}
```

Output: (screenshot)

```
Enter student name: Ash
Enter the number of grades: 4
Enter grade 1: 98
Enter grade 2: 77
Enter grade 3: 89
Enter grade 4: 79

Student Information:
Student Name: Ash
Grades: 98 77 89 79
Average Grade: 85.75
apple@Apples-MacBook-Air c++ %
```

Test Case: Any two (screenshot)

Conclusion:

The program showcases OOP principles through the `Student` class. It demonstrates encapsulation, abstraction, and resource management using constructors and destructors. User interaction allows inputting grades, calculating averages, and displaying comprehensive student information, exemplifying effective object-oriented design for managing student data in a classroom setting.