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

C++ PROGRAMMING LAB

Prepared by:

Name of Student: ___Sakshi Kore___

Roll No: ___33___

Batch: 2023-27

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

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

14	Create a C++ program to print the following pattern:
	*****
	* *
	* *
	* *
	*****
15	Write a C++ program to display the following pattern: 1 232 34543 4567654 34543 232
16	Write a program to creating an inventory management system for a small store. The system should use object-oriented principles in C++. Your program should have the following features: • Create a **Product** class that represents a product in the inventory. Each **Product** object should have the following attributes: • Product ID (an integer) • Product Name (a string) • Price (a floating-point number) • Quantity in stock (an integer) • Implement a parameterized constructor for the **Product** class to initialize the attributes when a new product is added to the inventory.
17	Write a program to manage student records. Create a class Student with attributes such as name, roll number, and marks. Implement methods for displaying student details, adding new students, and calculating the average marks of all students in the record system.
18	Write a program that implements a basic calculator. Use a class Calculator with methods to perform addition, subtraction, multiplication, and division of two numbers. The program should allow the user to input two numbers and select an operation to perform.

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

Experiment No: 1

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

Theory:

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

ax^2+bx+c=0

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

The quadratic formula provides the solutions to a quadratic equation in the form
x=-b+-root b^2-4ac/2a
b^2−4ac, is known as the discriminant.

Code:
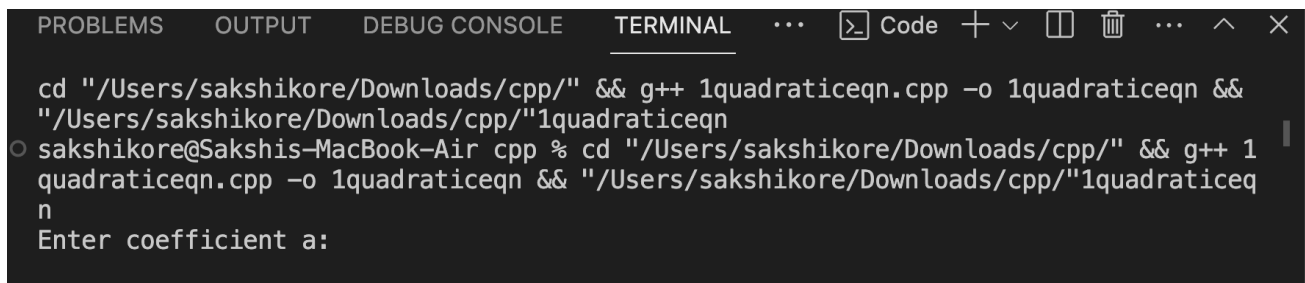
```cpp
#include <iostream>
#include <cmath>
using namespace std;
int main() {
    double a, b, c;
    // Input coefficients from the user
    cout << "Enter coefficient a: ";
    cin >> a;
    cout << "Enter coefficient b: ";
    cin >> b;
    cout << "Enter coefficient c: ";
    cin >> c;
    // Calculate the discriminant (b^2 - 4ac)
    // Check the discriminant to determine the nature of roots
    if (discriminant > 0) {
        // Two distinct real roots
        double root1 = (-b + sqrt(discriminant)) / (2 * a);
        double root2 = (-b - sqrt(discriminant)) / (2 * a);
        cout << "Roots are real and distinct:" << endl;
        cout << "Root 1 = " << root1 << endl;
        cout << "Root 2 = " << root2 << endl;
    } else if (discriminant == 0) {
        // One real root (double root)
        double root = -b / (2 * a);
        cout << "Roots are real and identical:" <<endl;
        cout << "Root = " << root << endl;
    } else {
```

```cpp
        // Complex roots
        double realPart = -b / (2 * a);
        double imaginaryPart = sqrt(-discriminant) / (2 * a);
        cout << "Roots are complex and conjugate:" << endl;
        cout << "Root 1 = " << realPart << " + " << imaginaryPart
<< "i" << endl;
        cout << "Root 2 = " << realPart << " - " << imaginaryPart
<< "i" << endl;
    }
    return 0;
```
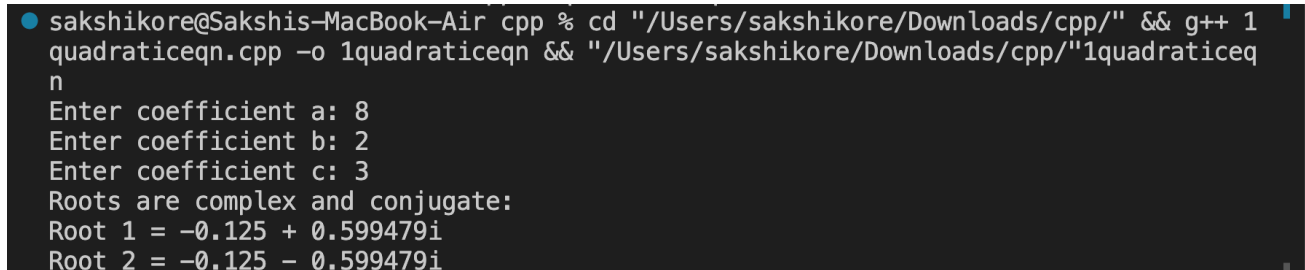
Output: (screenshot)

Test Case: Any two (screenshot)

Case1:

Case2:

Conclusion:

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

Experiment No: 2

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

Theory:

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

Code:

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

int main() {

    double base, power, result = 1;
    // Input base and exponent from the user
    cout << "Enter the base: ";
    cin >> base;
    cout << "Enter the power: ";
    cin >> power;
    // Calculate power using a loop

    for (int i = 0; i < power; i++) {
        result *= base;
    }

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

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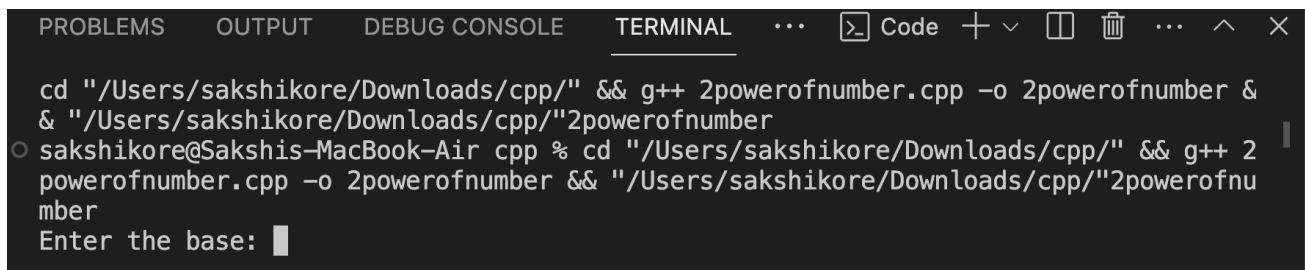
Output: (screenshot)

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   ···   >_ Code  + ∨  ⊓  🗑  ···  ∧  ✕

cd "/Users/sakshikore/Downloads/cpp/" && g++ 2powerofnumber.cpp -o 2powerofnumber &
& "/Users/sakshikore/Downloads/cpp/"2powerofnumber
○ sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 2
powerofnumber.cpp -o 2powerofnumber && "/Users/sakshikore/Downloads/cpp/"2powerofnu
mber
Enter the base: ▊
```

Test Case: Any two (screenshot)

Case1:

```
sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 2
powerofnumber.cpp -o 2powerofnumber && "/Users/sakshikore/Downloads/cpp/"2powerofnu
mber
Enter the base: 7
Enter the power: 4
7 raised to the power 4 is: 2401
sakshikore@Sakshis-MacBook-Air cpp %
```

Case2:

```
sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 2
powerofnumber.cpp -o 2powerofnumber && "/Users/sakshikore/Downloads/cpp/"2powerofnu
mber
Enter the base: 12
Enter the power: 3
12 raised to the power 3 is: 1728
sakshikore@Sakshis-MacBook-Air cpp %
```

Conclusion:

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

Experiment No: 3

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

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

Code:

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

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

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

    return true; // Palindrome
}

int main() {
    string inputString;

    cout << "Enter a string: ";
    getline(cin, inputString);

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

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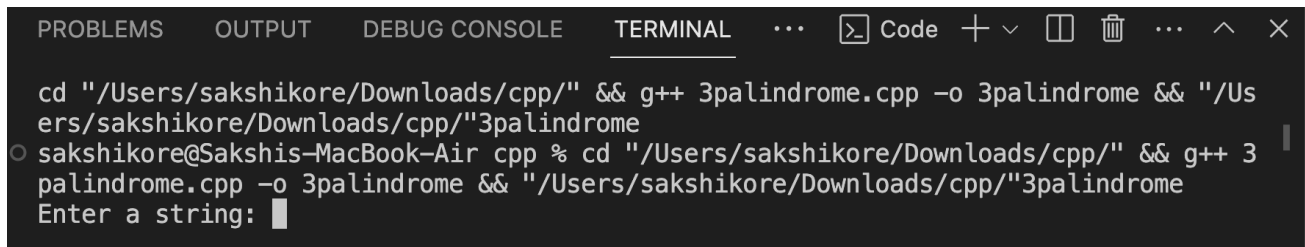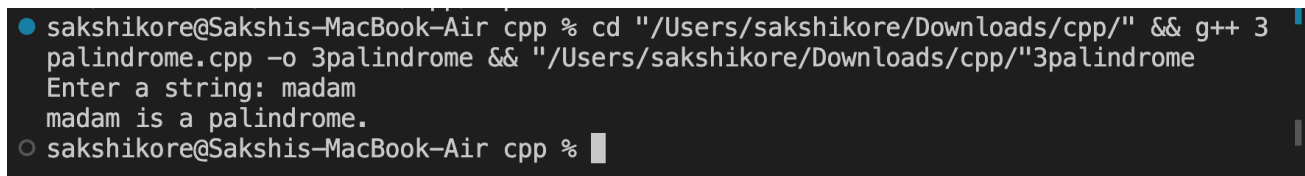
Output: (screenshot)

Test Case: Any two (screenshot)

Case1:

Case2:

Conclusion:

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

Experiment No: 4

Title:

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

Theory:

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

Code:

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

// Function to display the menu
void displayMenu()

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

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

    do
    {
        displayMenu(); // Display the menu

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

        // Process user choice using a switch statement
        switch (choice)
        {
            case 1:
                // Check Balance
```

```cpp
                cout << "Your balance is: "  << balance <<endl;
                break;

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

            case 3:
                // Withdraw
                cout << "Enter the amount to withdraw: ";
                cin >> amount;
                if (amount > 0 && amount <= balance)
                {
                    balance -= amount;
                    cout << "Withdrawal successful. Your new balance is: " << balance <<endl;
                } else
                {
                    cout << "Invalid amount or insufficient funds.";
                }
                break;

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

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

    } while (choice != 4);

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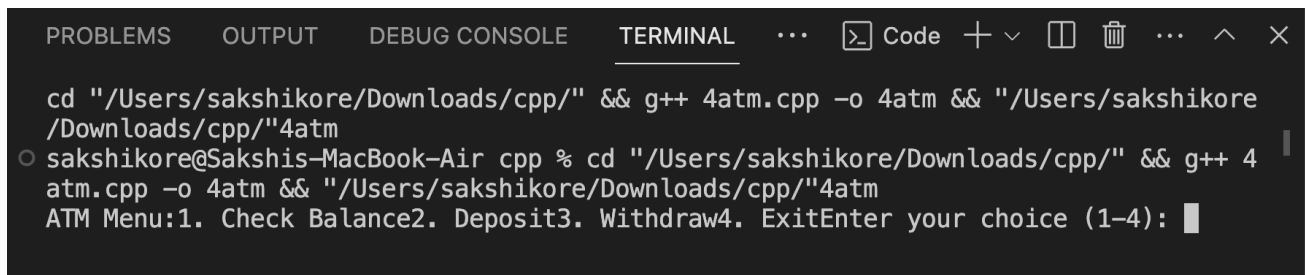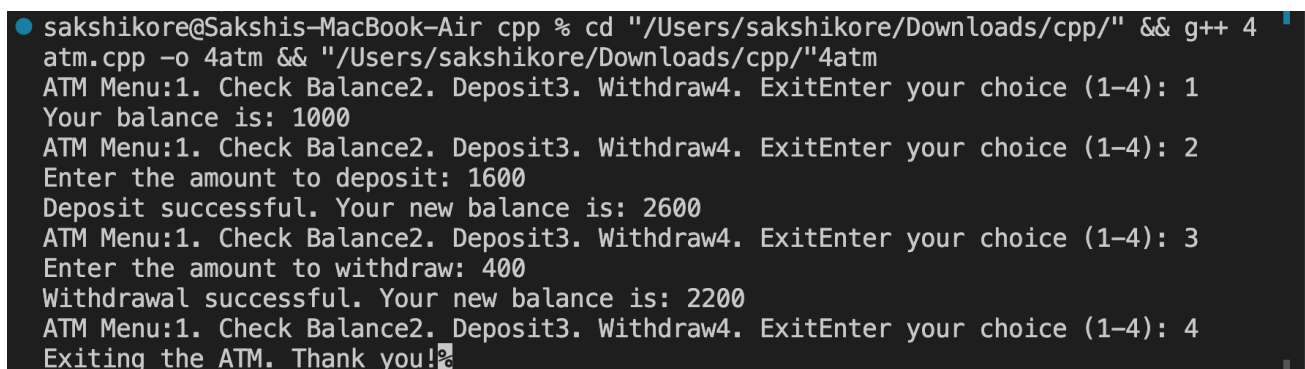
Output: (screenshot)

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   ···  >_ Code  + ∨  ⊞  🗑  ···  ∧  ✕

cd "/Users/sakshikore/Downloads/cpp/" && g++ 4atm.cpp -o 4atm && "/Users/sakshikore
/Downloads/cpp/"4atm
○ sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 4
atm.cpp -o 4atm && "/Users/sakshikore/Downloads/cpp/"4atm
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): ▮
```

Test Case: Any two (screenshot)

Case1:

```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 4
atm.cpp -o 4atm && "/Users/sakshikore/Downloads/cpp/"4atm
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 1
Your balance is: 1000
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 2
Enter the amount to deposit: 1600
Deposit successful. Your new balance is: 2600
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 3
Enter the amount to withdraw: 400
Withdrawal successful. Your new balance is: 2200
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 4
Exiting the ATM. Thank you!%
```

Case2:

```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 4
atm.cpp -o 4atm && "/Users/sakshikore/Downloads/cpp/"4atm
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 1
Your balance is: 1000
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 2
Enter the amount to deposit: 2000
Deposit successful. Your new balance is: 3000
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 3
Enter the amount to withdraw: 800
Withdrawal successful. Your new balance is: 2200
ATM Menu:1. Check Balance2. Deposit3. Withdraw4. ExitEnter your choice (1-4): 4
Exiting the ATM. Thank you!%
```
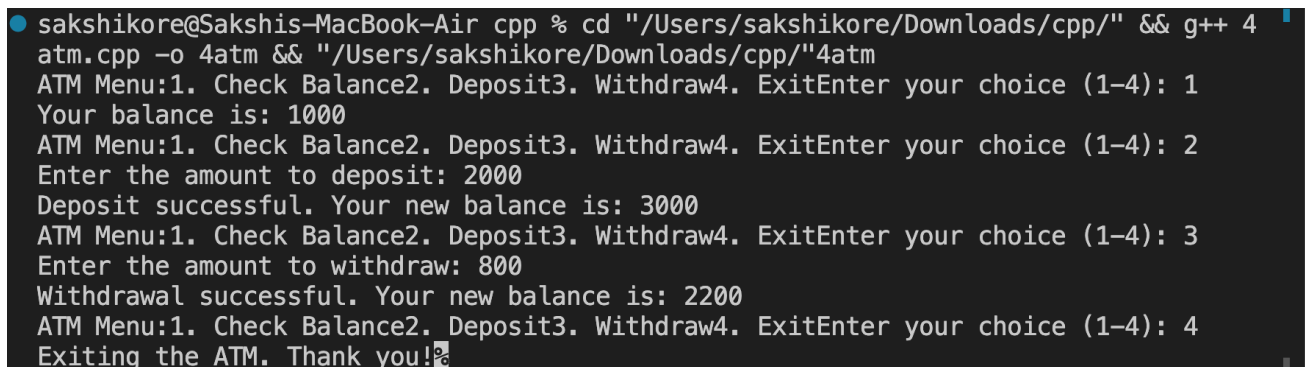
Conclusion:

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

Experiment No: 5

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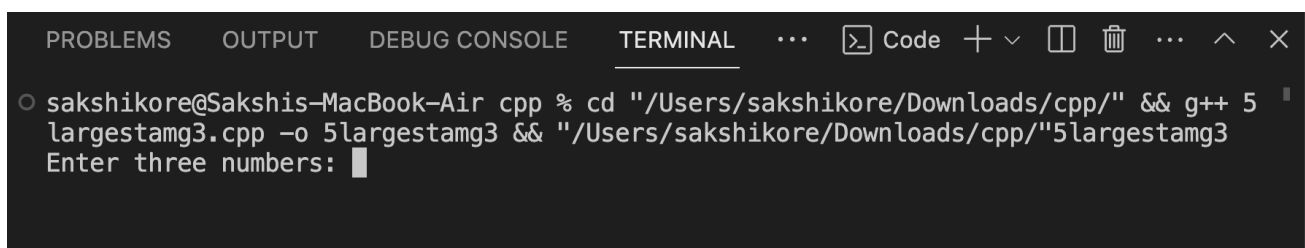
Theory:

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

Code:

```cpp
#include <iostream>
using namespace std;
int main()
{
    double num1, num2, num3;

    // Input three numbers from the user
    cout << "Enter three numbers: ";
    cin >> num1 >> num2 >> num3;
    // Nested if-else statements to find the largest number
    if (num1 >= num2) {
        if (num1 >= num3) {
            cout << "The largest number is: " << num1 << endl;
        } else {
            cout << "The largest number is: " << num3 << endl;
        }
    } else {
        if (num2 >= num3) {
            cout << "The largest number is: " << num2 << endl;
        } else {
            cout << "The largest number is: " << num3 << endl;
        }
    }
    return 0;
}
```

Output: (screenshot)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ...    >_ Code  + ∨  ⬚  🗑  ...  ∧  ✕

○ sakshikore@Sakshis—MacBook—Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 5
  largestamg3.cpp —o 5largestamg3 && "/Users/sakshikore/Downloads/cpp/"5largestamg3
  Enter three numbers: ▮
```

Test Case: Any two (screenshot)

Case1:

```
sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 5
largestamg3.cpp -o 5largestamg3 && "/Users/sakshikore/Downloads/cpp/"5largestamg3
Enter three numbers: 58
92
71
The largest number is: 92
sakshikore@Sakshis-MacBook-Air cpp %
```

Case2:

```
sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 5
largestamg3.cpp -o 5largestamg3 && "/Users/sakshikore/Downloads/cpp/"5largestamg3
Enter three numbers: 13
102
200
The largest number is: 200
```

Conclusion:

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

Experiment No: 6

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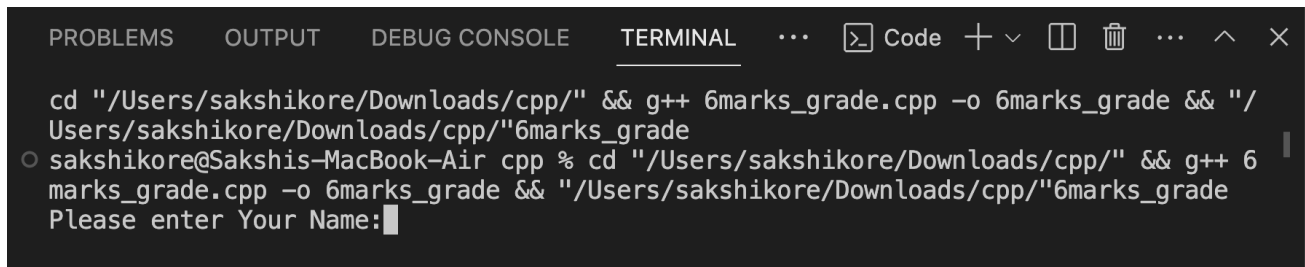

Theory:

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

Code:

```cpp
#include <iostream>
using namespace std;
int main() {
    string name;
    // Input marks for 5 subjects
    double sub1, sub2, sub3, sub4, sub5;
    cout<<"Please enter Your Name:";
    cin>>name;
    cout << "Enter marks for 5 subjects:";
    cin >> sub1 >> sub2 >> sub3 >> sub4 >> sub5;
    // Calculate the average marks
    double averageMarks = (sub1 + sub2 + sub3 + sub4 + sub5) / 5;
    // Determine the grade based on the average marks using if-else-if
ladder
    if (averageMarks >= 90)
    {
        cout<<name<<" "<< "You have grade: A";
    }
    else if (averageMarks >= 80)
    {
        cout<<name<<" "<< "You have grade: B";
    }
    else if (averageMarks >= 70)
    {
        cout<<name<<" "<< "You have grade: C";
    }
    else if (averageMarks >= 60)
    {
        cout<<name<<" "<< "You have grade: D";
    }
    else
    {
        cout<<name<<" "<< "You have grade: F";
    }
    return 0;
}
```
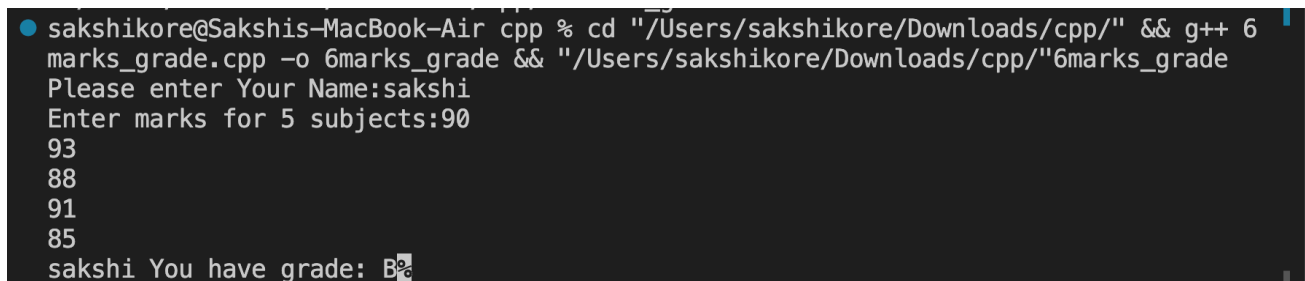
Output: (screenshot)

Test Case: Any two (screenshot)

Case1:

Case2:

Conclusion:

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

Experiment No: 7

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

Theory:

- The program defines a function sumOfDigits that takes an integer and iteratively adds its digits until a single-digit sum is obtained.
- The main function prompts the user to input a number, calls the sumOfDigits function, and displays the final sum.

Code:

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

int main()
{
    int number;

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

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

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

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

    return 0;
}
```

Output: (screenshot)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ···    >_ Code  + ∨  ⊓  🗑  ···  ∧  ✕

cd "/Users/sakshikore/Downloads/cpp/" && g++ 7sum_of_digit.cpp -o 7sum_of_digit &&
"/Users/sakshikore/Downloads/cpp/"7sum_of_digit
sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 7
sum_of_digit.cpp -o 7sum_of_digit && "/Users/sakshikore/Downloads/cpp/"7sum_of_digi
t
Enter a number: ▮
```
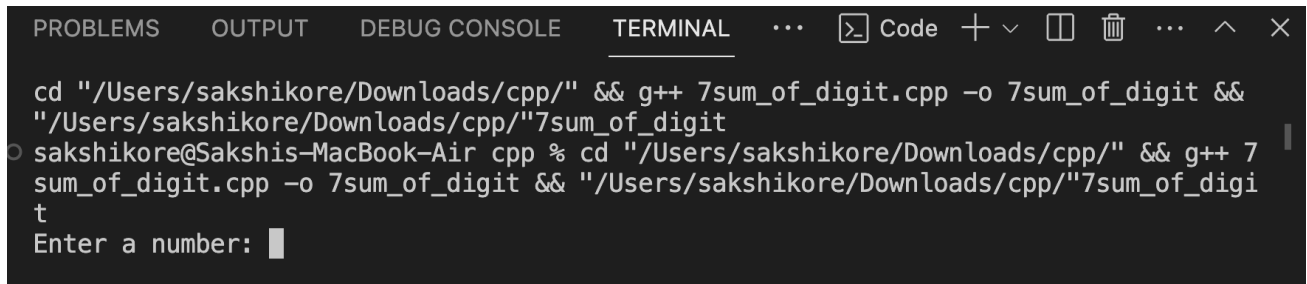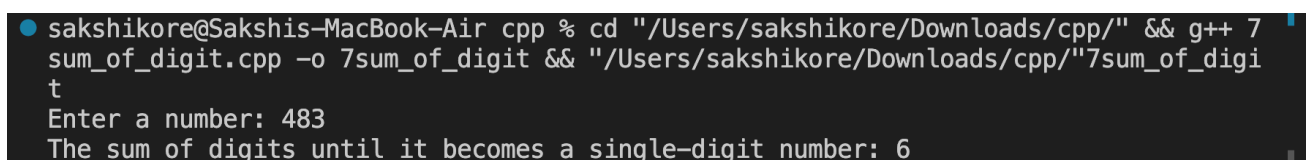
Test Case: Any two (screenshot)

Case1:

```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 7
  sum_of_digit.cpp -o 7sum_of_digit && "/Users/sakshikore/Downloads/cpp/"7sum_of_digi
  t
  Enter a number: 217
  The sum of digits until it becomes a single-digit number: 1
```

Case2:

```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 7
  sum_of_digit.cpp -o 7sum_of_digit && "/Users/sakshikore/Downloads/cpp/"7sum_of_digi
  t
  Enter a number: 483
  The sum of digits until it becomes a single-digit number: 6
```

Conclusion:

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

Experiment No: 8

Title:

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

Theory:

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

Code:

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

int main() {
    int numRows;
    // Input: Number of rows in Pascal's Triangle
    cout << "Enter the number of rows for Pascal's Triangle: ";
    cin >> numRows;
    // Printing Pascal's Triangle
    for (int i = 0; i < numRows; i++) {
        // Adding spaces to center the triangle
        for (int j = 0; j < numRows - i - 1; j++) {
            cout << " ";
        }
        int coeff = 1;
        for (int j = 0; j <= i; j++) {
            // Calculate and print the binomial coefficient for
each position
            cout << coeff << " ";
            // Update the binomial coefficient for the next
position
            coeff = coeff * (i - j) / (j + 1);
        }
        cout << endl;
    }
    return 0;
}
```

Output: (screenshot)


```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ···    >_ Code  + ∨  ⊡  🗑  ···  ∧  ✕

○ sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 8
  pascals_tri.cpp -o 8pascals_tri && "/Users/sakshikore/Downloads/cpp/"8pascals_tri
  Enter the number of rows for Pascal's Triangle: █
```
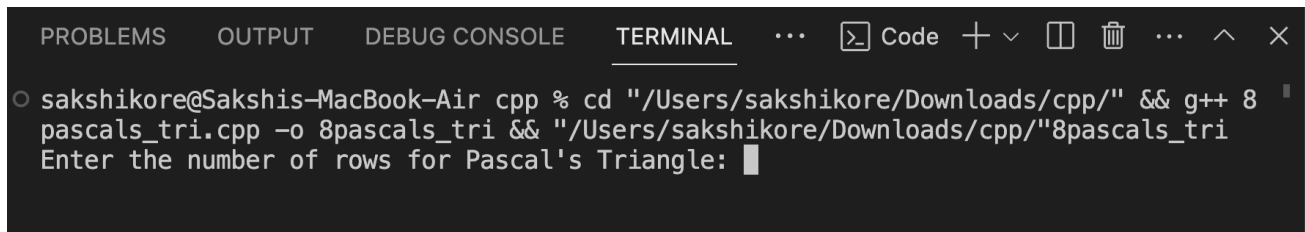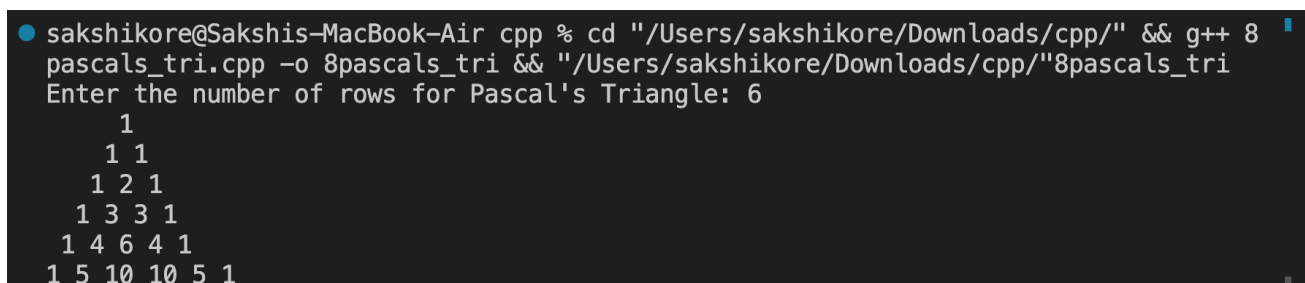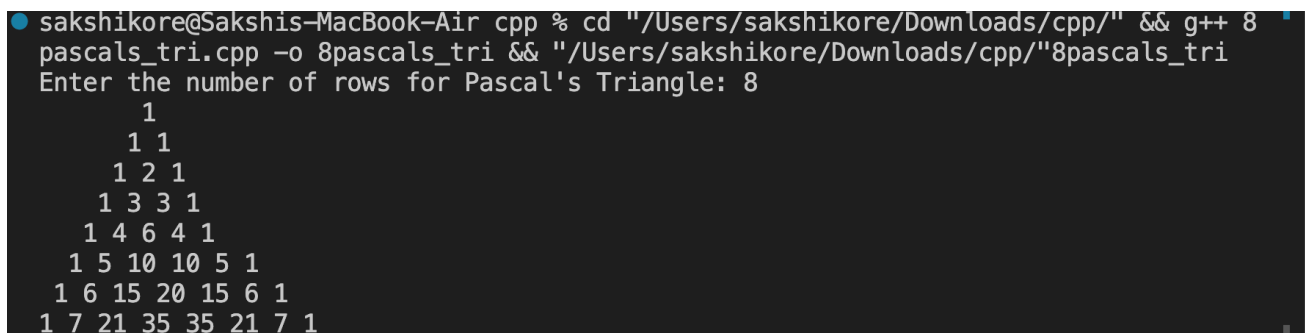
Test Case: Any two (screenshot)

Case1:


```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 8
  pascals_tri.cpp -o 8pascals_tri && "/Users/sakshikore/Downloads/cpp/"8pascals_tri
  Enter the number of rows for Pascal's Triangle: 6
      1
     1 1
    1 2 1
   1 3 3 1
  1 4 6 4 1
 1 5 10 10 5 1
```

Case2:


```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 8
  pascals_tri.cpp -o 8pascals_tri && "/Users/sakshikore/Downloads/cpp/"8pascals_tri
  Enter the number of rows for Pascal's Triangle: 8
        1
       1 1
      1 2 1
     1 3 3 1
    1 4 6 4 1
   1 5 10 10 5 1
  1 6 15 20 15 6 1
 1 7 21 35 35 21 7 1
```

Conclusion:

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

Experiment No: 9

Title:

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

Theory:

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

Code:

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

int main() {
    int N;

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

    double sum = 0;
    double term = 1;

    // Calculating the sum of the series
    for (int i = 1; i <= N; i++) {
        term *= i;   // Calculate factorial
        sum += i / term;
    }

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

    return 0;
}
```

Output: (screenshot)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ...    >_ Code  + ∨  ⊓ 🗑 ...  ∧ ✕

cd "/Users/sakshikore/Downloads/cpp/" && g++ 9sum_of_series.cpp -o 9sum_of_series &
& "/Users/sakshikore/Downloads/cpp/"9sum_of_series
○ sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 9
sum_of_series.cpp -o 9sum_of_series && "/Users/sakshikore/Downloads/cpp/"9sum_of_se
ries
Enter the value of N:
```
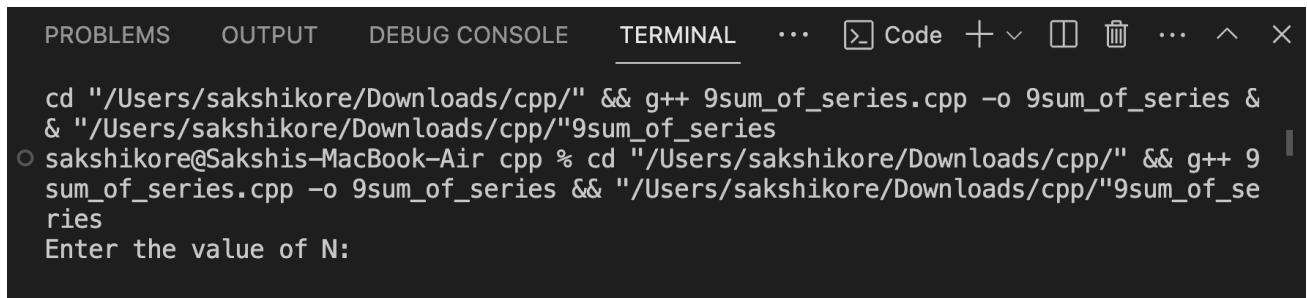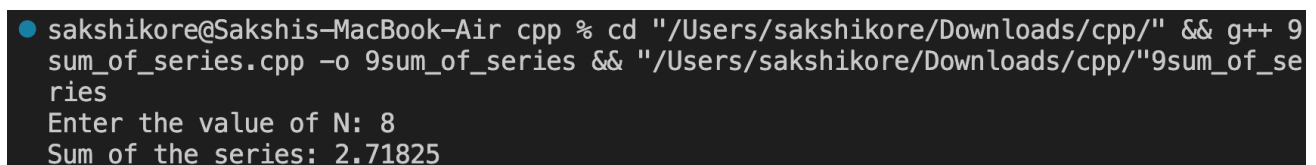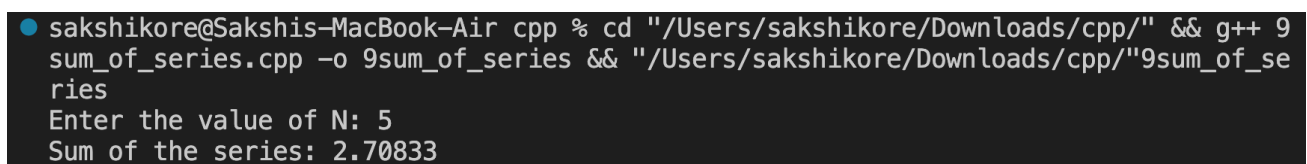
Test Case: Any two (screenshot)

Case1:

```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 9
sum_of_series.cpp -o 9sum_of_series && "/Users/sakshikore/Downloads/cpp/"9sum_of_se
ries
Enter the value of N: 8
Sum of the series: 2.71825
```

Case2

```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 9
sum_of_series.cpp -o 9sum_of_series && "/Users/sakshikore/Downloads/cpp/"9sum_of_se
ries
Enter the value of N: 5
Sum of the series: 2.70833
```

Conclusion:

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

Experiment No: 10

Title:

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

Theory:

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

Code:

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

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

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

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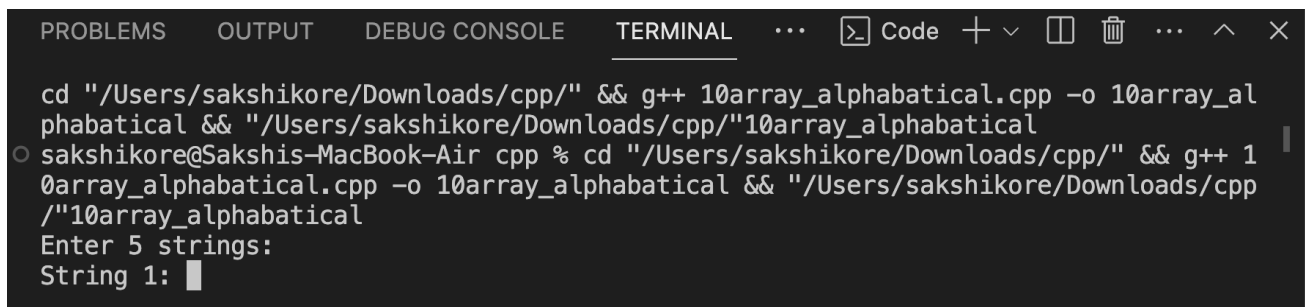
Output: (screenshot)

```
PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   ...   >_ Code  + ∨   ⊓  🗑  ...  ∧  ✕

cd "/Users/sakshikore/Downloads/cpp/" && g++ 10array_alphabatical.cpp -o 10array_al
phabatical && "/Users/sakshikore/Downloads/cpp/"10array_alphabatical
○ sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
0array_alphabatical.cpp -o 10array_alphabatical && "/Users/sakshikore/Downloads/cpp
/"10array_alphabatical
Enter 5 strings:
String 1: █
```
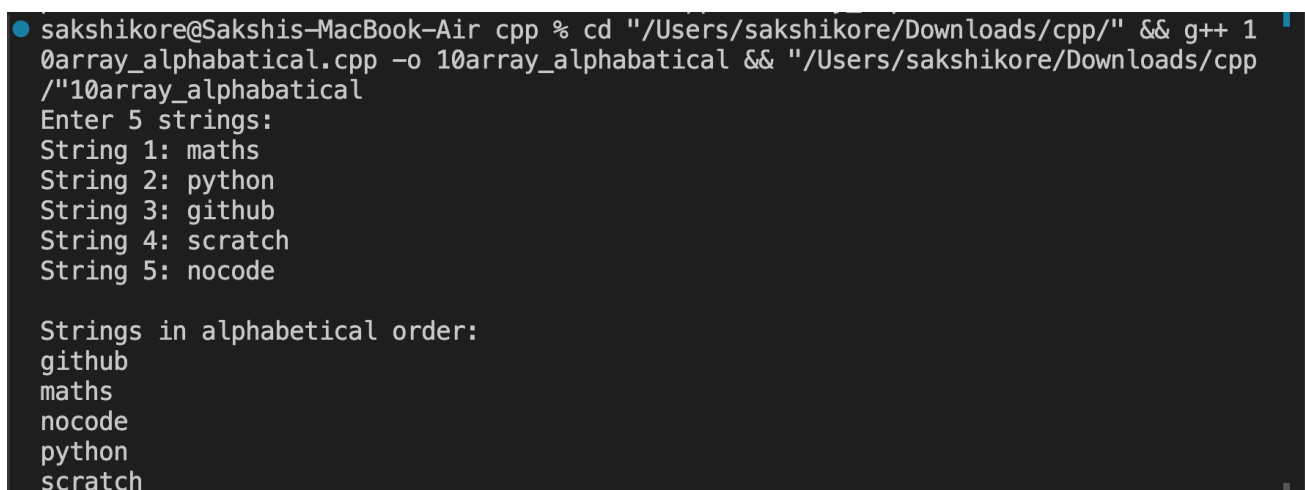
Test Case: Any two (screenshot)
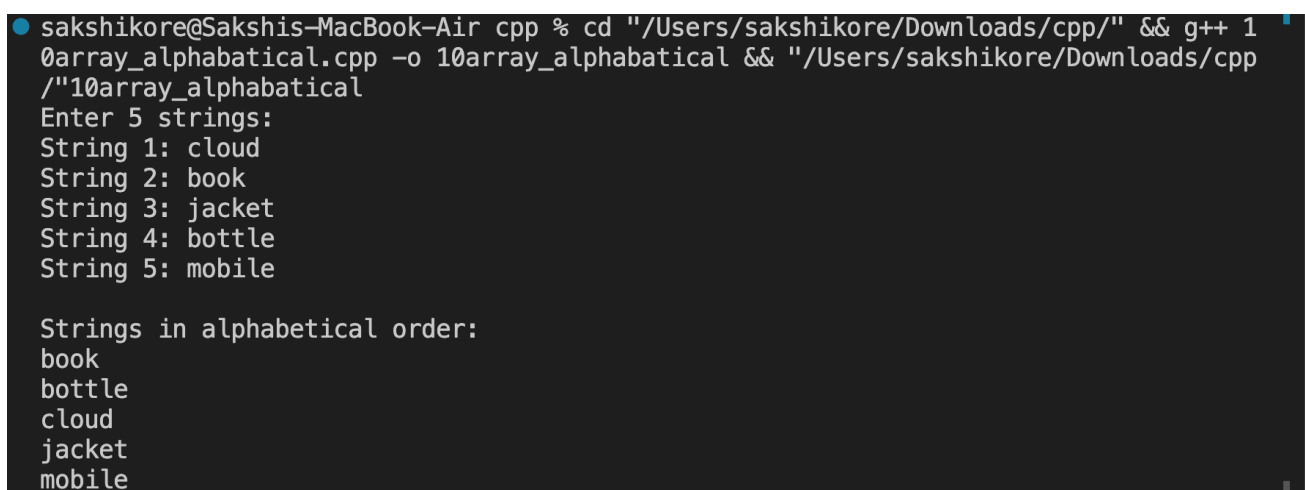
Case1:

```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
0array_alphabatical.cpp -o 10array_alphabatical && "/Users/sakshikore/Downloads/cpp
/"10array_alphabatical
Enter 5 strings:
String 1: maths
String 2: python
String 3: github
String 4: scratch
String 5: nocode

Strings in alphabetical order:
github
maths
nocode
python
scratch
```

Case2:

```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
0array_alphabatical.cpp -o 10array_alphabatical && "/Users/sakshikore/Downloads/cpp
/"10array_alphabatical
Enter 5 strings:
String 1: cloud
String 2: book
String 3: jacket
String 4: bottle
String 5: mobile

Strings in alphabetical order:
book
bottle
cloud
jacket
mobile
```

Conclusion:

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

Experiment No: 11

Title:

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

Theory:

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

Code:
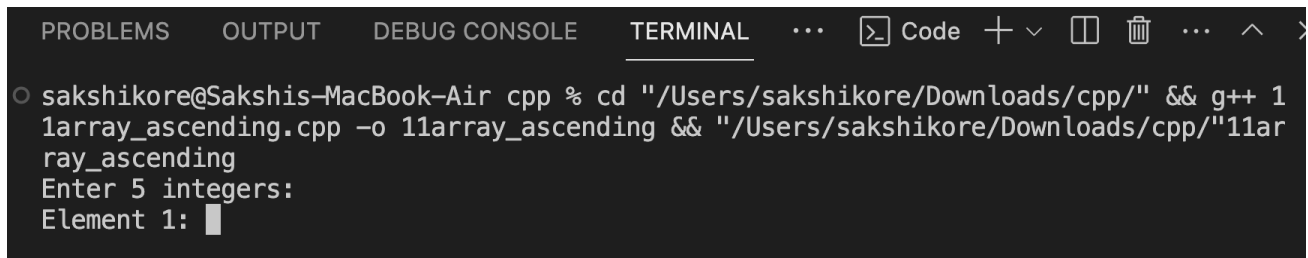
```cpp
#include <iostream>

using namespace std;

int main() {
    const int size = 5;   // Change the size based on the size of your array
    int array[size];

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

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

    // If the loop completed without breaking, the array is sorted
    if (i == size) {
        cout << "The array is sorted in ascending order.\n";
    }
    return 0;
}
```

Output: (screenshot)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ···    >_ Code  + ∨  ⊞  🗑  ···  ∧  ⟩

○ sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
  1array_ascending.cpp -o 11array_ascending && "/Users/sakshikore/Downloads/cpp/"11ar
  ray_ascending
  Enter 5 integers:
  Element 1: ▊
```
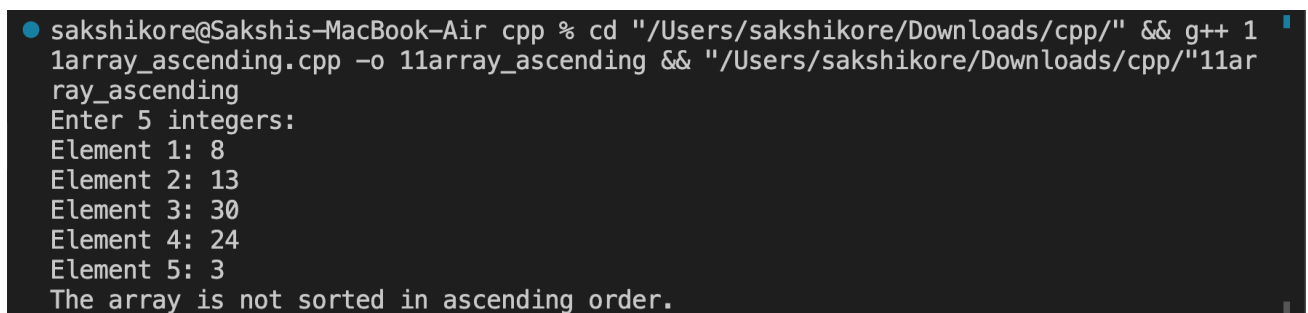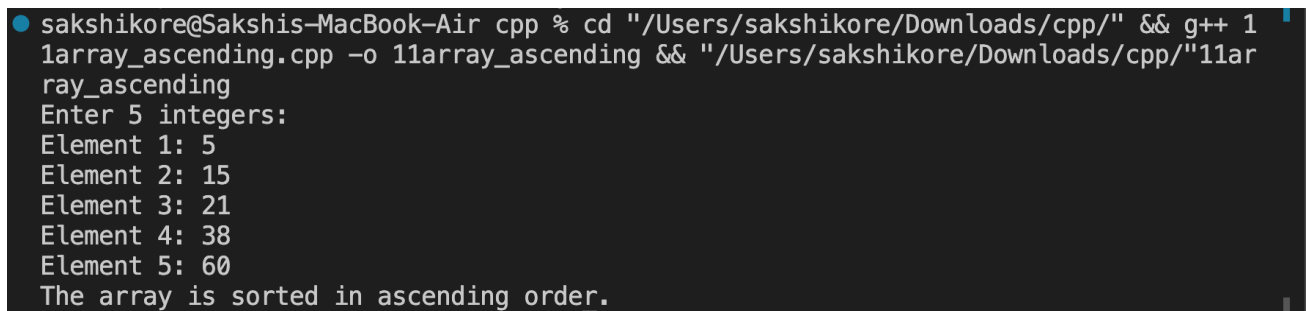
Test Case: Any two (screenshot)

Case1:

```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
  1array_ascending.cpp -o 11array_ascending && "/Users/sakshikore/Downloads/cpp/"11ar
  ray_ascending
  Enter 5 integers:
  Element 1: 8
  Element 2: 13
  Element 3: 30
  Element 4: 24
  Element 5: 3
  The array is not sorted in ascending order.
```

Case2:

```
● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
  1array_ascending.cpp -o 11array_ascending && "/Users/sakshikore/Downloads/cpp/"11ar
  ray_ascending
  Enter 5 integers:
  Element 1: 5
  Element 2: 15
  Element 3: 21
  Element 4: 38
  Element 5: 60
  The array is sorted in ascending order.
```

Conclusion:

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

Experiment No: 12

Title:

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

Theory:

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

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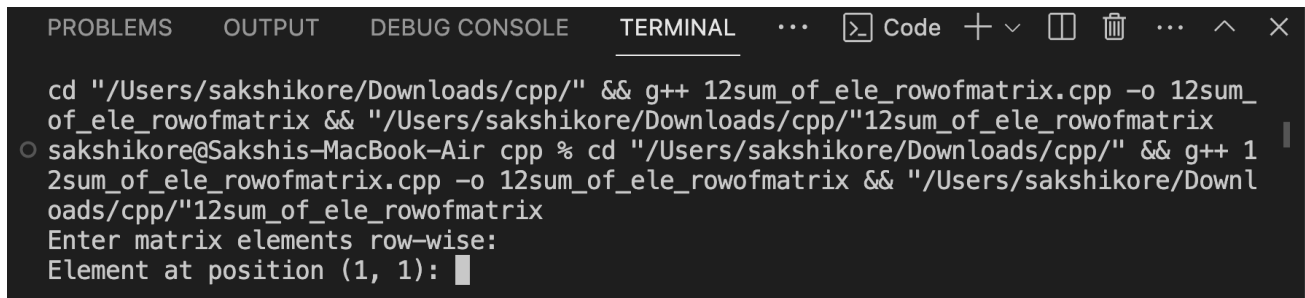
```cpp
#include <iostream>
using namespace std;

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

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

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

    return 0;
}
```

Output: (screenshot)

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ···    >_ Code  + ∨  ⊓  🗑  ···  ∧  ✕

cd "/Users/sakshikore/Downloads/cpp/" && g++ 12sum_of_ele_rowofmatrix.cpp -o 12sum_
of_ele_rowofmatrix && "/Users/sakshikore/Downloads/cpp/"12sum_of_ele_rowofmatrix
○ sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
2sum_of_ele_rowofmatrix.cpp -o 12sum_of_ele_rowofmatrix && "/Users/sakshikore/Downl
oads/cpp/"12sum_of_ele_rowofmatrix
Enter matrix elements row-wise:
Element at position (1, 1): ▮
```
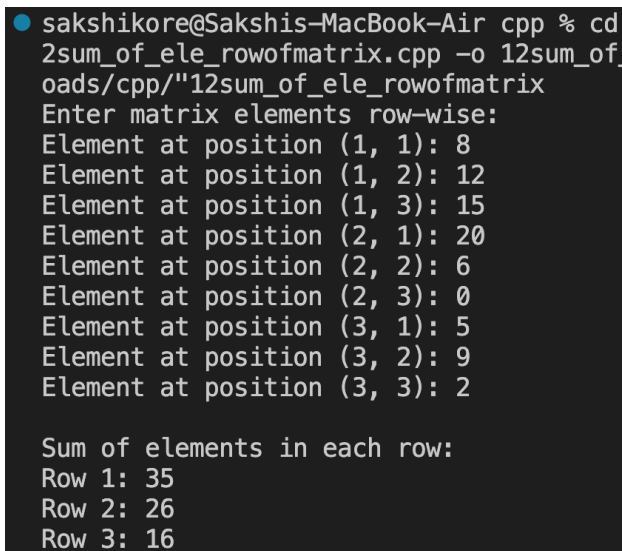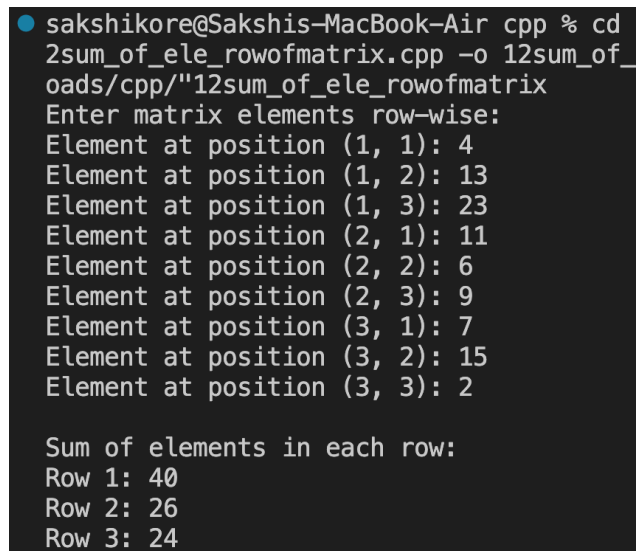
Test Case: Any two (screenshot)

Case1:

```
● sakshikore@Sakshis-MacBook-Air cpp % cd
2sum_of_ele_rowofmatrix.cpp -o 12sum_of_
oads/cpp/"12sum_of_ele_rowofmatrix
Enter matrix elements row-wise:
Element at position (1, 1): 8
Element at position (1, 2): 12
Element at position (1, 3): 15
Element at position (2, 1): 20
Element at position (2, 2): 6
Element at position (2, 3): 0
Element at position (3, 1): 5
Element at position (3, 2): 9
Element at position (3, 3): 2

Sum of elements in each row:
Row 1: 35
Row 2: 26
Row 3: 16
```

Case 2:

```
● sakshikore@Sakshis-MacBook-Air cpp % cd
2sum_of_ele_rowofmatrix.cpp -o 12sum_of_
oads/cpp/"12sum_of_ele_rowofmatrix
Enter matrix elements row-wise:
Element at position (1, 1): 4
Element at position (1, 2): 13
Element at position (1, 3): 23
Element at position (2, 1): 11
Element at position (2, 2): 6
Element at position (2, 3): 9
Element at position (3, 1): 7
Element at position (3, 2): 15
Element at position (3, 3): 2

Sum of elements in each row:
Row 1: 40
Row 2: 26
Row 3: 24
```

Conclusion:

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

Experiment No: 13

Title:

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

Theory:

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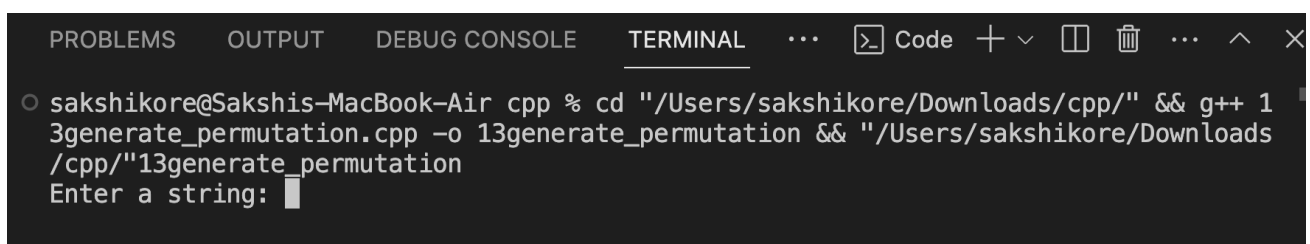
Code:

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

using namespace std;

int main() {
    string str;
    // Input: Taking a string from the user
    cout << "Enter a string: ";
    cin >> str;
    // Sorting the string to get the initial permutation
    sort(str.begin(), str.end());
    // Displaying the initial permutation
    cout << "\nAll permutations of the string:\n";
    do
    {
        cout << str << endl;
    } while (next_permutation(str.begin(), str.end()));
    return 0;
}
```

Output: (screenshot)

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   ···   >_ Code + ∨ ⊓ 🗑 ··· ∧ ✕

○ sakshikore@Sakshis–MacBook–Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
  3generate_permutation.cpp –o 13generate_permutation && "/Users/sakshikore/Downloads
  /cpp/"13generate_permutation
  Enter a string: ▮

Test Case: Any two (screenshot)

Case1:

```
sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
3generate_permutation.cpp -o 13generate_permutation && "/Users/sakshikore/Downloads
/cpp/"13generate_permutation
Enter a string: god

All permutations of the string:
dgo
dog
gdo
god
odg
ogd
```

Case2:

```
sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
3generate_permutation.cpp -o 13generate_permutation && "/Users/sakshikore/Downloads
/cpp/"13generate_permutation
Enter a string: fun

All permutations of the string:
fnu
fun
nfu
nuf
ufn
unf
```

Conclusion:

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

Title:

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

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

Theory:

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

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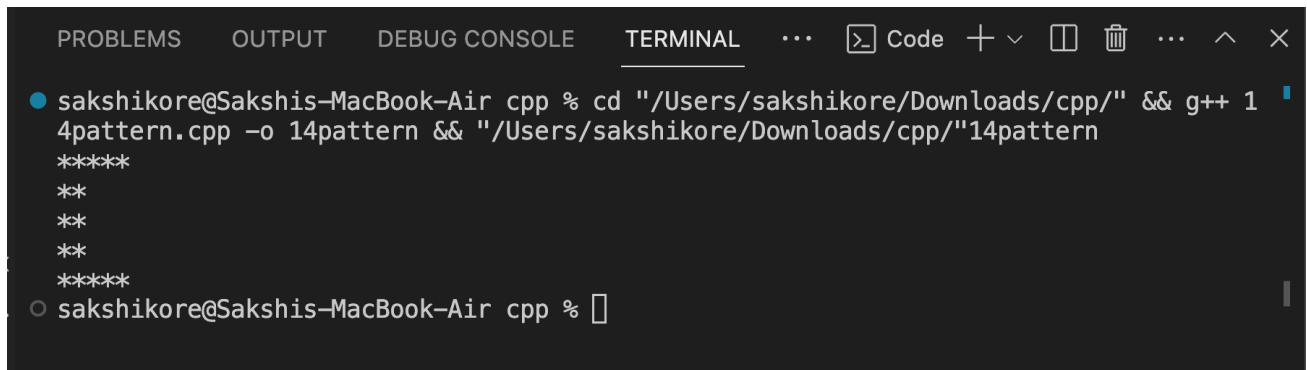
```cpp
#include <iostream>
using namespace std;

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

    // Printing the pattern
    for (int i = 1; i <= size; ++i) {
        for (int j = 1; j <= size; ++j) {
            // Printing '*' for the first and last rows, and the first and last columns
            if (i == 1 || i == size || j == 1 || j == size) {
                cout << "*";
            } else {
                // Printing space for other positions
                cout << "";
            }
        }
        cout << endl;
    }

    return 0;
}
```

Output: (screenshot)


```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ⋯    >_ Code  + ∨  ⊞  🗑  ⋯  ∧  ✕

● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
  4pattern.cpp -o 14pattern && "/Users/sakshikore/Downloads/cpp/"14pattern
  *****
  **
  **
  **
  *****
○ sakshikore@Sakshis-MacBook-Air cpp % ▯
```

Conclusion:

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

Title:

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

Theory:

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

Code:

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

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

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

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

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

```cpp
        }

        for (int j = num - 1; j >= 1; --j)

        {
            cout << j;
        }

        cout << endl;
    }

    return 0;
```

Output: (screenshot)

PROBLEMS   OUTPUT   DEBUG CONSOLE   TERMINAL   ···   >_ Code + ∨ ▯ 🗑 ··· ∧ ✕

● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
  5pattern.cpp -o 15pattern && "/Users/sakshikore/Downloads/cpp/"15pattern
  1
  121
  12321
  1234321
  12321
  121
  1
○ sakshikore@Sakshis-MacBook-Air cpp %

Conclusion:

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

Title:

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

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

Theory:

- The program defines a Product class with private attributes (productId, productName, price, quantityInStock) and public methods to display information, get quantity, and update quantity.
- Two Product objects are created using the parameterized constructor, and their initial information is displayed.
- The program simulates a sale by updating the quantity of one product and displays the updated information.

Code:

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

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

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

```cpp
        {}

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

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

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

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

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

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

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

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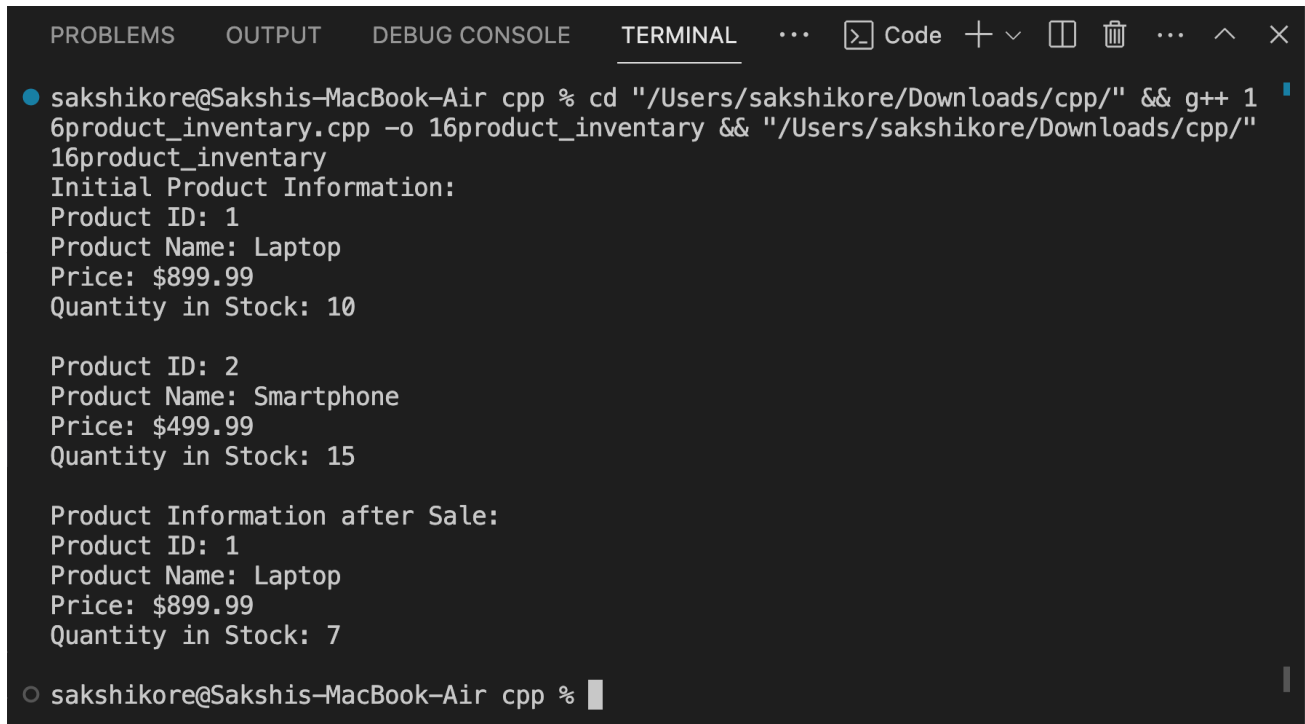
Output: (screenshot)


```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ···    >_ Code  + ∨  ⊓  🗑  ···  ∧  ✕

● sakshikore@Sakshis-MacBook-Air cpp % cd "/Users/sakshikore/Downloads/cpp/" && g++ 1
  6product_inventory.cpp -o 16product_inventory && "/Users/sakshikore/Downloads/cpp/"
  16product_inventory
  Initial Product Information:
  Product ID: 1
  Product Name: Laptop
  Price: $899.99
  Quantity in Stock: 10

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

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

○ sakshikore@Sakshis-MacBook-Air cpp %
```

Conclusion:

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

Title:

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

Theory:

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

Code:

```cpp
#include <iostream>
using namespace std;
class Student {
private:
    string name;
    int rollNumber;
    float marks;
public:
    Student(const string& studentName, int studentRollNumber, float studentMarks)
        : name(studentName), rollNumber(studentRollNumber), marks(studentMarks) {}
    void displayStudentDetails() const {
        cout << "Name: " << name << endl;
        cout << "Roll Number: " << rollNumber << endl;
        cout << "Marks: " << marks << endl;
        cout << "-----------------------\n";
    }
    const string& getName() const { return name; }
    void setName(const string& studentName) { name = studentName; }
    int getRollNumber() const { return rollNumber; }
    void setRollNumber(int studentRollNumber) { rollNumber = studentRollNumber; }
    float getMarks() const { return marks; }
```

```cpp
    void setMarks(float studentMarks) { marks = studentMarks; }
};
class StudentRecordSystem {
private:
    vector<Student> studentRecords;
public:
    void addStudent() {
        string name;
        int rollNumber;
        float marks;
        cout << "Enter student details:\n";
        cout << "Name: ";
        cin.ignore();
        getline(cin, name);
        cout << "Roll Number: ";
        cin >> rollNumber;
        cout << "Marks: ";
        cin >> marks;
        studentRecords.push_back(Student(name, rollNumber, marks));
        cout << "Student added successfully!\n";
    }
    void displayAllStudents() const {
        if (studentRecords.empty()) {
            cout << "No student records available.\n";
        } else {
            cout << "Student Details:\n";
            for (const auto& student : studentRecords) {
                student.displayStudentDetails();
            }
        }
    }
    void calculateAndDisplayAverageMarks() const {
        if (studentRecords.empty()) {
            cout << "No student records available.\n";
        } else {
            float totalMarks = 0;
            for (const auto& student : studentRecords) {
                totalMarks += student.getMarks();
            }
            float averageMarks = totalMarks /
studentRecords.size();
            cout << "Average Marks of All Students: " <<
averageMarks << std::endl;
        }
    }
};
int main() {
    StudentRecordSystem recordSystem;
    recordSystem.addStudent();
```

```
    recordSystem.addStudent();
    recordSystem.addStudent();
    recordSystem.displayAllStudents();
    recordSystem.calculateAndDisplayAverageMarks();
    return 0;
}
```

Output: (screenshot)

```
Enter student details:
Name: sakshi kore
Roll Number: 33
Marks: 90
Student added successfully!
Enter student details:
Name: arpita jadhav
Roll Number: 28
Marks: 92
Student added successfully!
Enter student details:
Name: yashika thakur
Roll Number: 23
Marks: 88
Student added successfully!
Student Details:
Name: akshi kore
Roll Number: 33
Marks: 90
---------------------------
Name: arpita jadhav
Roll Number: 28
Marks: 92
---------------------------
Name: yashika thakur
Roll Number: 23
Marks: 88
---------------------------
Average Marks of All Students: 90
```

Conclusion:

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

Title:

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

Theory:

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

Code:

```cpp
#include <iostream>

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

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

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

```cpp
            return 0;
        }
    }
};

int main() {
    // Create an instance of the Calculator class
    Calculator calculator;
    // Input two numbers from the user
    float num1, num2;
    std::cout << "Enter the first number: ";
    std::cin >> num1;
    std::cout << "Enter the second number: ";
    std::cin >> num2;

    // Display menu for operation selection
    std::cout << "Select operation:\n";
    std::cout << "1. Addition\n";
    std::cout << "2. Subtraction\n";
    std::cout << "3. Multiplication\n";
    std::cout << "4. Division\n";
    int choice;
    std::cout << "Enter your choice (1-4): ";
    std::cin >> choice;
    // Perform the selected operation
    switch (choice) {
        case 1:
            std::cout << "Result: " << calculator.add(num1, num2)
<< "\n";
            break;
        case 2:
            std::cout << "Result: " << calculator.subtract(num1,
num2) << "\n";
            break;
        case 3:
            std::cout << "Result: " << calculator.multiply(num1,
num2) << "\n";
            break;
        case 4:
            std::cout << "Result: " << calculator.divide(num1,
num2) << "\n";
            break;
        default:
            std::cerr << "Error: Invalid choice.\n";
            break;
    }
    return 0;
}
```
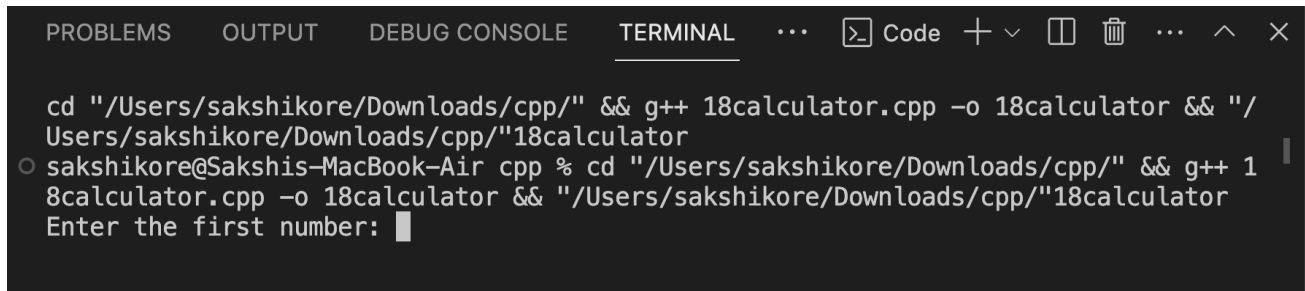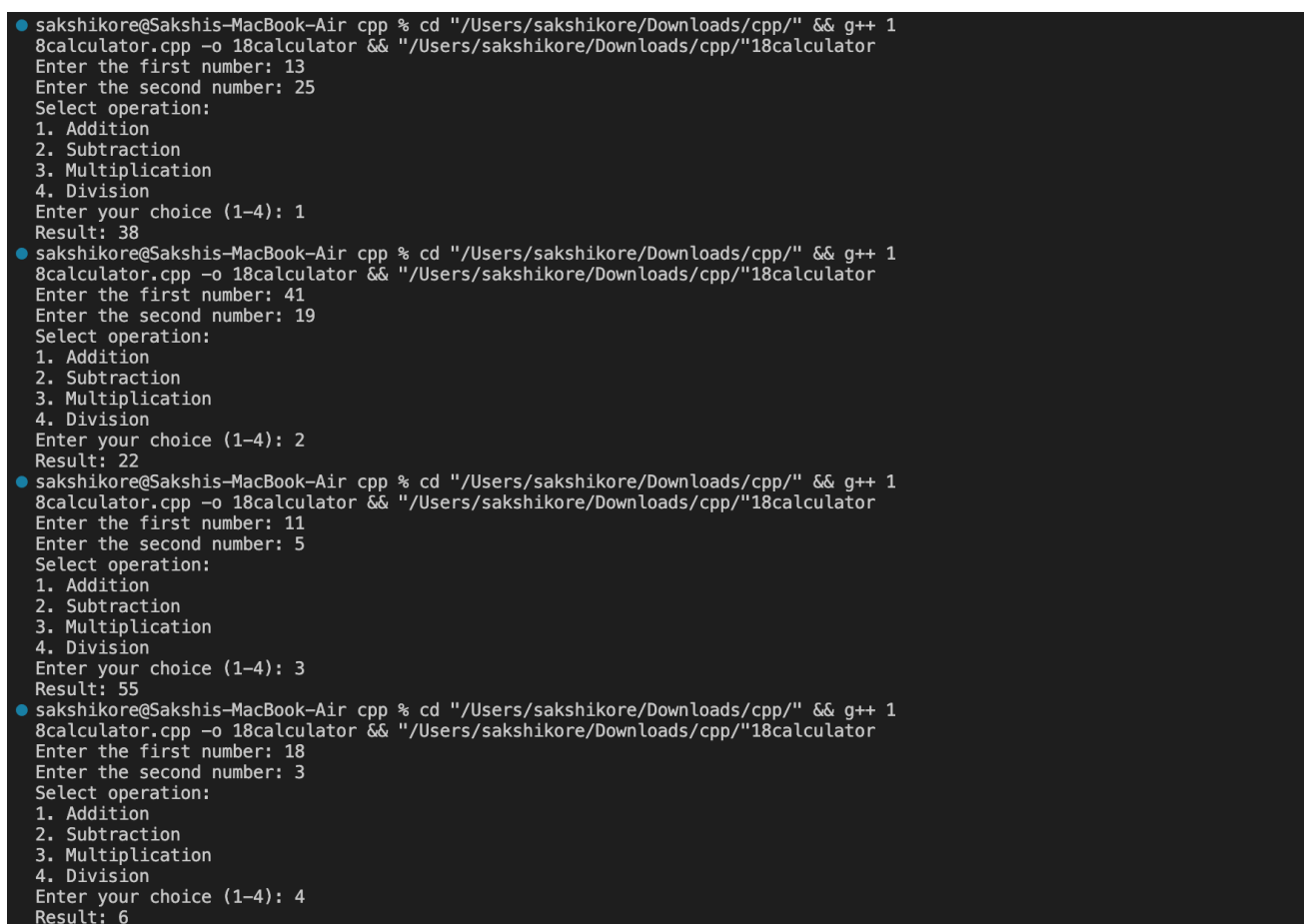
Output: (screenshot)

Test Case: Any two (screenshot)

Conclusion:

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

Experiment No: 19

Title:

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

Theory:

Creating a class Product and creating methods for adding product name, price, and
quantity to the cart and displaying the cart at the end.

Code:

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

class Product
{

private:
    string name, prod[5];
    float prices[5], sum = 0;
    int quantity, quan[5], n;

public:
    Product()
    {
        cout << "Enter number of products: ";
        cin >> n;
        prod[n];
        prices[n];
        quan[n];

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

        {
            cout << "Enter name of product: ";
            cin.ignore();
            getline(cin, name);
```

```cpp
            prod[i] = name;
            cout << "Enter cost: ";
            cin >> prices[i];
            // prices[i]=price;
            cout << "Enter quantity: ";
            cin >> quantity;
            quan[i] = quantity;
            sum += (prices[i] * quan[i]);
        }
    }

    void cart()
    {
        cout << "Cart: " << endl
             << "Product Name"
             << "\t"
             << "Price"
             << "\t"
             << "Quantity"
             << "\t" << endl;

        for (int i = 0; i < n; i++)
        {
            cout << prod[i] << "\t\t" << prices[i] << "\t" <<
quan[i] << endl;
        }
        cout << "Total cost: " << sum << endl;
    }
};

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

Output: (screenshot)


```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    ...    Code

 4 warnings generated.
 Enter number of products: 0
 Cart:
 Product Name      Price    Quantity
 Total cost: 0
 sakshikore@Sakshis-MacBook-Air cpp %
```

Test Case: Any two (screenshot)

```
Enter number of products: 4
Enter name of product: Earphones
Enter cost: 899
Enter quantity: 1
Enter name of product: Tops
Enter cost: 450
Enter quantity: 2
Enter name of product: Watch
Enter cost: 1200
Enter quantity: 1
Enter name of product: Shoes
Enter cost: 1490
Enter quantity: 1
Cart:
Product Name    Price   Quantity
Earphones               899     1
Tops            450     2
Watch           1200    1
Shoes           1490    1
Total cost: 4489
```

Conclusion:

Hence, by using for loop to ask for product details by the user for user given number of products and displaying the cart at the end with total cost and product details.

Experiment No: 20

Title:

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

Theory:

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

Code:

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

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

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

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

    // Method to add grades
    void addGrade(int grade) {
        if (numGrades < 10) { // Assuming a maximum of 10 grades
            grades[numGrades++] = grade;
```

```cpp
        } else {
            cout << "Cannot add more than 10 grades.\n";
        }
    }

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

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

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

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

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

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

    // Display student details
    student1.displayStudentDetails();

    return 0;
}
```

Output: (screenshot)

Conclusion:

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