200+ C++ Programs for Beginners

Hernando Abella

ALUNA PUBLISHING HOUSE

Thank you for trusting our Publishing House. If you have the opportunity to evaluate our work and give us a comment on Amazon, we will appreciate it very much!

Table of contents

1. Print Hello World

This program displays the text "*Hello, World!*" on the screen. It is the traditional first program when learning any new programming language.

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

int main() {
    cout << "Hello, World!" << endl;
    return 0;
}

// Hello, World!
```

2. Add Two Numbers

This program asks the user to enter two numbers, adds them together, and displays the result. It introduces input/output and simple arithmetic in C++.

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

int main() {
    double num1, num2;

    // Read two numbers from the user
    cout << "Enter the first number: ";
    cin >> num1;
    cout << "Enter the second number: ";
    cin >> num2;

    // Output the sum
    cout << "The sum of " << num1 << " and " << num2
         << " is: " << num1 + num2 << endl;

    return 0;
}

// Enter the first number: 12
// Enter the second number: 8
// The sum of 12 and 8 is: 20
```

3. Find the Square Root

This program asks the user for a non-negative number and calculates its square root using the `sqrt()` function from the C++ math library. If the number is negative, it shows an error message.

```
#include <iostream>
#include <cmath>   // for sqrt()
#include <iomanip> // for output formatting
using namespace std;

int main() {
    double input_number;

    // Read a non-negative number from the user
    cout << "Enter a non-negative number: ";
    cin >> input_number;

    // Check if the number is non-negative and calculate the
square root
    if (input_number >= 0.0) {
        cout << fixed << setprecision(2);
        cout << "The square root of " << input_number
                    << " is: " << setprecision(5) <<
sqrt(input_number) << endl;
    } else {
        cout << "Please enter a valid non-negative number."
<< endl;
    }

    return 0;
}

// Example Run:
// Enter a non-negative number: 3
// The square root of 3.00 is: 1.73205
```

4. Calculate the Area of a Triangle

This program asks the user to enter the base and height of a triangle, calculates its area using the formula `0.5 * base * height`, and then displays the result.

```
#include <iostream>
#include <iomanip> // for output formatting
using namespace std;

int main() {
    double base, height;

    // Read base and height from the user
    cout << "Enter the base of the triangle: ";
    cin >> base;
    cout << "Enter the height of the triangle: ";
    cin >> height;

    // Calculate the area of the triangle
    double area = 0.5 * base * height;

    cout << fixed << setprecision(2);
    cout << "The area of the triangle with base " << base
         << " and height " << height
         << " is: " << area << endl;

    return 0;
}

// Enter the base of the triangle: 3
// Enter the height of the triangle: 4
// The area of the triangle with base 3.00 and height 4.00
is: 6.00
```

5. Swap Two Variables

This program takes two variables (strings) from the user, swaps their values using a temporary variable, and then displays the values before and after swapping.

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

int main() {
    string variable1, variable2, temp;

    // Read variables from the user
    cout << "Enter the first variable: ";
    getline(cin, variable1);

    cout << "Enter the second variable: ";
    getline(cin, variable2);

    // Display before swapping
    cout << "Before swapping: Variable1 = " << variable1
         << ", Variable2 = " << variable2 << endl;

    // Swapping the variables using a temporary variable
    temp = variable1;
    variable1 = variable2;
    variable2 = temp;

    // Display after swapping
    cout << "After swapping: Variable1 = " << variable1
         << ", Variable2 = " << variable2 << endl;

    return 0;
}

// Enter the first variable: a
// Enter the second variable: b
// Before swapping: Variable1 = a, Variable2 = b
// After swapping: Variable1 = b, Variable2 = a
```

6. Solve Quadratic Equation

This program solves a quadratic equation of the form `ax^2 + bx + c = 0`. It calculates the discriminant to determine whether the roots are real and distinct, real and equal, or complex.

```
#include <iostream>
#include <cmath>    // for sqrt()
#include <iomanip>  // for output formatting
using namespace std;

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

    // Read coefficients from the user
    cout << "Enter the coefficient a: ";
    cin >> a;
    if (a == 0.0) {
        cout << "Coefficient 'a' cannot be zero." << endl;
        return 1;
    }
    cout << "Enter the coefficient b: ";
    cin >> b;
    cout << "Enter the coefficient c: ";
    cin >> c;

    // Calculate discriminant
    double discriminant = b * b - 4 * a * c;

    // Determine roots based on discriminant
    cout << fixed << setprecision(2);
    if (discriminant > 0) {
        double root1 = (-b + sqrt(discriminant)) / (2 * a);
        double root2 = (-b - sqrt(discriminant)) / (2 * a);
        cout << "The roots are: " << root1 << " and " <<
root2 << endl;
    } else if (discriminant == 0) {
        double root = -b / (2 * a);
        cout << "The root is: " << root << endl;
```

```
    } else {
        cout << "The equation has complex roots." << endl;
    }

    return 0;
}

// Enter the coefficient a: 1
// Enter the coefficient b: -3
// Enter the coefficient c: 2
// The roots are: 2.00 and 1.00
```

7. Convert Kilometers to Miles

This program converts a distance from kilometers to miles using the conversion factor `1 km ≈ 0.621371 miles`.

```
#include <iostream>
#include <iomanip> // for output formatting
using namespace std;

int main() {
    double kilometers;

    // Read distance in kilometers from user
    cout << "Enter the distance in kilometers: ";
    cin >> kilometers;

    // Convert kilometers to miles
    double miles = kilometers * 0.621371;

    cout << fixed << setprecision(2) << kilometers << "
kilometers is approximately ";
    cout << setprecision(6) << miles << " miles." << endl;

    return 0;
}

// Enter the distance in kilometers: 2
// 2.00 kilometers is approximately 1.242742 miles.
```

8. Convert Celsius to Fahrenheit

This program converts a temperature from Celsius to Fahrenheit using the formula `F = (C × 9/5) + 32`.

```
#include <iostream>
#include <iomanip> // for output formatting
using namespace std;

int main() {
    double celsius;

    // Read temperature in Celsius from user
    cout << "Enter the temperature in Celsius: ";
    cin >> celsius;

    // Convert Celsius to Fahrenheit
    double fahrenheit = (celsius * 9.0 / 5.0) + 32.0;

    cout << fixed << setprecision(2) << celsius << "
degrees Celsius is equal to ";
    cout << setprecision(1) << fahrenheit << " degrees
Fahrenheit." << endl;

    return 0;
}

// Enter the temperature in Celsius: 33
// 33.00 degrees Celsius is equal to 91.4 degrees
Fahrenheit.
```

9. Generate a Random Number

This program generates a random number within a user-specified range. It uses the C++ random number generation functions seeded with the current time.

```
#include <iostream>
#include <iomanip> // for output formatting
#include <cstdlib> // for rand() and srand()
#include <ctime>   // for time()
using namespace std;

int main() {
    double min_range, max_range;

    // Prompt user for the range
    cout << "Enter the minimum value of the range: ";
    cin >> min_range;
    cout << "Enter the maximum value of the range: ";
    cin >> max_range;

    // Check if input is valid
    if (min_range < max_range) {
        // Seed the random number generator
        srand(static_cast<unsigned int>(time(0)));

        // Generate a random number within the specified
range
        double random_number = ((double)rand() / RAND_MAX)
* (max_range - min_range) + min_range;

        cout << fixed << setprecision(2) << "A random
number between "
             << min_range << " and " << max_range << " is:
";
        cout << setprecision(10) << random_number << endl;
    } else {
        cout << "Please enter valid numbers, ensuring that
the minimum value is less than the maximum value." << endl;
```

```
    }

    return 0;
}

// Enter the minimum value of the range: 3
// Enter the maximum value of the range: 4
// A random number between 3.00 and 4.00 is: 3.9904595578
```

10. Check if a number is Positive, Negative, or Zero

This program asks the user to enter a number and determines whether it is positive, negative, or zero. It introduces conditional statements in C++.

```cpp
#include <iostream>
#include <iomanip> // for output formatting
using namespace std;

int main() {
    double number;

    // Prompt user for a number
    cout << "Enter a number: ";
    if (!(cin >> number)) {
        cout << "Please enter a valid number." << endl;
        return 1; // Exit if input is invalid
    }

    // Check if the number is positive, negative, or zero
    cout << fixed << setprecision(2);
    if (number > 0.0) {
        cout << number << " is a positive number." << endl;
    } else if (number < 0.0) {
        cout << number << " is a negative number." << endl;
    } else {
        cout << "The entered number is zero." << endl;
    }

    return 0;
}

// Enter a number: 3
// 3.00 is a positive number.
```

11. Check if a Number is Odd or Even

This program checks whether an integer entered by the user is odd or even using the modulo operator `%`.

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

int main() {
    int number;

    // Prompt user for a number
    cout << "Enter a number: ";
    if (!(cin >> number)) {
        cout << "Please enter a valid integer." << endl;
        return 1; // Exit if input is invalid
    }

    // Check if the number is odd or even
    if (number % 2 == 0) {
        cout << number << " is an even number." << endl;
    } else {
        cout << number << " is an odd number." << endl;
    }

    return 0;
}

// Enter a number: 3
// 3 is an odd number.
```

12. Find the Largest Among Three Numbers

This program asks the user to enter three numbers and determines the largest among them using conditional comparisons.

```
#include <iostream>
#include <iomanip> // for output formatting
using namespace std;

// Function to read a number with input validation
double read_number(const string &prompt) {
    double number;
    cout << prompt;
    while (!(cin >> number)) {
        cout << "Please enter a valid number." << endl;
        cin.clear();                // Clear the error flag
        cin.ignore(10000, '\n');    // Discard invalid input
        cout << prompt;
    }
    return number;
}

int main() {
    double num1 = read_number("Enter the first number: ");
    double num2 = read_number("Enter the second number: ");
    double num3 = read_number("Enter the third number: ");

    double largest_number = num1;
    if (num2 > largest_number) largest_number = num2;
    if (num3 > largest_number) largest_number = num3;

    cout << fixed << setprecision(2);
    cout << "The largest number among " << num1 << ", " <<
num2 << ", and " << num3
         << " is: " << largest_number << endl;

    return 0;
```

```
}

// Enter the first number: 3
// Enter the second number: 4
// Enter the third number: 3
// The largest number among 3.00, 4.00, and 3.00 is: 4.00
```

13. Check Prime Number

This program checks whether a given integer greater than 1 is a prime number by testing divisibility up to its square root.

```
#include <iostream>
#include <cmath> // for sqrt()
using namespace std;

int main() {
    int number;
    cout << "Enter a number: ";

    // Read the input
    if (!(cin >> number) || number <= 1) {
        cout << "Please enter a valid integer greater than
1." << endl;
        return 1;
    }

    // Check if the number is prime
    bool is_prime = true; // Assume it is prime initially
    for (int i = 2; i <= sqrt(number); i++) {
        if (number % i == 0) {
            is_prime = false; // Not prime
            break;
        }
    }

    // Output the result
    if (is_prime) {
        cout << number << " is a prime number." << endl;
    } else {
        cout << number << " is not a prime number." <<
endl;
    }

    return 0;
}
```

```
// Enter a number: 4
// 4 is not a prime number.
```

14. Print All Prime Numbers in an Interval

This program prints all prime numbers within a user-specified interval. It uses a helper function to check whether each number is prime.

```
#include <iostream>
#include <cmath> // for sqrt()
using namespace std;

// Function to check if a number is prime
bool is_prime(int n) {
    if (n < 2) return false; // Numbers less than 2 are not
prime
    for (int i = 2; i * i <= n; i++) // Check up to the
square root of n
        if (n % i == 0) return false; // Not prime
    return true; // Prime
}

int main() {
    int start, end;
    cout << "Enter the starting number: ";
    cin >> start;
    cout << "Enter the ending number: ";
    cin >> end;

    if (start > 1 && start < end) {
        cout << "Prime numbers in [" << start << " - " <<
end << "]:" << endl;
        for (int n = start; n <= end; n++)
            if (is_prime(n)) cout << n << endl;
    } else {
        cout << "Invalid range. Start should be > 1 and <
end." << endl;
    }

    return 0;
```

```
}

// Example Run:
// Enter the starting number: 33
// Enter the ending number: 44
// Prime numbers in [33 - 44]:
// 37
// 41
// 43
```

15. Find the Factorial of a Number

This program calculates the factorial of a non-negative integer using a loop. Factorials grow quickly, so we use `unsigned long long` to store large results.

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

// Function to calculate factorial
unsigned long long factorial(int n) {
    unsigned long long result = 1; // Use unsigned long
long to handle larger results
    for (int i = 2; i <= n; i++) {
        result *= i; // Calculate factorial
    }
    return result;
}

int main() {
    int number;
    cout << "Enter a non-negative integer: ";

    // Read user input
    if (cin >> number && number >= 0) {
        cout << "The factorial of " << number << " is: " <<
factorial(number) << endl;
    } else {
        cout << "Please enter a valid non-negative
integer." << endl;
    }

    return 0;
}

// Enter a non-negative integer: 3
// The factorial of 3 is: 6
```

16. Display the Multiplication Table

This program displays the multiplication table of a number entered by the user from 1 to 10.

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

int main() {
    int number;
    cout << "Enter a number for the multiplication table:
";

    // Read user input
    if (cin >> number) {
        cout << "Multiplication table for " << number <<
":" << endl;

        // Display the multiplication table
        for (int i = 1; i <= 10; i++) {
            cout << number << " x " << i << " = " << number
* i << endl;
        }
    } else {
        cout << "Please enter a valid integer." << endl;
    }

    return 0;
}

// Enter a number for the multiplication table: 2
// Multiplication table for 2:
// 2 x 1 = 2
// 2 x 2 = 4
// 2 x 3 = 6
// 2 x 4 = 8
// 2 x 5 = 10
// 2 x 6 = 12
// 2 x 7 = 14
```

```
// 2 x 8 = 16….
```

17. Print the Fibonacci Sequence

This program prints the first `n` terms of the Fibonacci sequence, where `n` is entered by the user. It uses an array to store the sequence.

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

int main() {
    int num_terms;
    cout << "Enter the number of terms in the Fibonacci
sequence: ";

    // Read user input
    if (cin >> num_terms) {
        if (num_terms <= 0) {
            cout << "Please enter a positive integer." <<
endl;
            return 1;
        }

        unsigned long long fib[num_terms];
        fib[0] = 0;
        if (num_terms > 1) {
            fib[1] = 1;
        }

        for (int i = 2; i < num_terms; i++) {
            fib[i] = fib[i - 1] + fib[i - 2];
        }

        // Print the Fibonacci sequence
        for (int i = 0; i < num_terms; i++) {
            cout << fib[i];
            if (i < num_terms - 1) {
                cout << ", ";
            }
        }
```

```
        cout << endl;
    } else {
        cout << "Please enter a valid positive integer." <<
endl;
    }

    return 0;
}

// Enter the number of terms in the Fibonacci sequence: 6
// 0, 1, 1, 2, 3, 5
```

18. Check Armstrong Number

This program checks whether a given positive integer is an Armstrong number. An Armstrong number is a number that is equal to the sum of its digits raised to the power of the number of digits.

```
#include <iostream>
#include <cmath> // for pow()
using namespace std;

int main() {
    unsigned int number, temp, remainder, digits = 0, sum =
0;
    cout << "Enter a number: ";

    // Read user input
    if (cin >> number) {
        temp = number;

        // Calculate the number of digits
        while (temp != 0) {
            digits++;
            temp /= 10;
        }

        temp = number;

        // Calculate the sum of the digits raised to the
power of the number of digits
        while (temp != 0) {
            remainder = temp % 10;
            sum += pow(remainder, digits);
            temp /= 10;
        }

        // Check if the sum equals the original number
        if (sum == number) {
            cout << number << " is an Armstrong number." <<
endl;
```

```
        } else {
            cout << number << " is not an Armstrong
number." << endl;
        }
    } else {
        cout << "Please enter a valid positive integer." <<
endl;
    }

    return 0;
}

// Enter a number: 153
// 153 is an Armstrong number.
```

19. Find Armstrong Number in an Interval

This program prints all Armstrong numbers in a user-specified interval. An Armstrong number is equal to the sum of its digits raised to the power of the number of digits.

```
#include <iostream>
#include <cmath> // for pow()
using namespace std;

// Function to check if a number is Armstrong
bool is_armstrong(unsigned int num) {
    unsigned int temp = num, digits = 0, sum = 0;

    // Count the number of digits
    while (temp != 0) {
        digits++;
        temp /= 10;
    }

    temp = num;

    // Calculate the sum of each digit raised to the power
of the number of digits
    while (temp != 0) {
        unsigned int remainder = temp % 10;
        sum += pow(remainder, digits);
        temp /= 10;
    }

    return sum == num;
}

int main() {
    unsigned int start, end;

    cout << "Enter the starting number: ";
    cin >> start;
```

```
    cout << "Enter the ending number: ";
    cin >> end;

    if (start > 0 && start < end) {
        cout << "Armstrong numbers in the interval [" <<
start << ", " << end << "]:" << endl;
        for (unsigned int i = start; i <= end; i++) {
            if (is_armstrong(i)) {
                cout << i << endl;
            }
        }
    } else {
        cout << "Please enter valid positive integers with
the starting number less than the ending number." << endl;
    }

    return 0;
}

// Enter the starting number: 100
// Enter the ending number: 200
// Armstrong numbers in the interval [100, 200]:
// 153
```

20. Make a Simple Calculator

This program performs basic arithmetic operations (`+`, `-`, `*`, `/`) on two numbers entered by the user. It demonstrates the use of `switch` statements and input validation.

```
#include <iostream>
#include <iomanip> // for output formatting
using namespace std;

// Function to read a number
double read_number(const string &prompt) {
    double number;
    cout << prompt;
    cin >> number;
    return number;
}

// Function to read an operation
char read_operation() {
    char op;
    cout << "Enter an operation (+, -, *, /): ";
    cin >> op;
    return op;
}

int main() {
    double num1 = read_number("Enter the first number: ");
    double num2 = read_number("Enter the second number: ");
    char op = read_operation();

    double result;
    switch (op) {
        case '+':
            result = num1 + num2;
            break;
        case '-':
            result = num1 - num2;
            break;
        case '*':
```

```
            result = num1 * num2;
            break;
        case '/':
            if (num2 == 0.0) {
                cout << "Cannot divide by zero." << endl;
                return 1;
            }
            result = num1 / num2;
            break;
        default:
            cout << "Invalid operation." << endl;
            return 1;
    }

    cout << fixed << setprecision(2);
    cout << "Result: " << num1 << " " << op << " " << num2
<< " = " << result << endl;

    return 0;
}

// Enter the first number: 5
// Enter the second number: 2
// Enter an operation (+, -, *, /): *
// Result: 5.00 * 2.00 = 10.00
```

21. Find the Sum of Natural Numbers

This program calculates the sum of the first `n` natural numbers entered by the user using the formula `n*(n+1)/2`.

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

int main() {
    unsigned long long n;
    cout << "Enter a positive integer: ";

    // Read input and check if it's a valid positive
integer
    if (!(cin >> n) || n == 0) {
        cout << "Please enter a valid positive integer." <<
endl;
        return 1;
    }

    // Calculate the sum of natural numbers
    unsigned long long sum_of_natural_numbers = (n * (n +
1)) / 2;
    cout << "The sum of natural numbers from 1 to " << n <<
" is: " << sum_of_natural_numbers << endl;

    return 0;
}

// Enter a positive integer: 3
// The sum of natural numbers from 1 to 3 is: 6
```

22. Check if the Numbers Have the Same Last Digit

This program checks whether two numbers entered by the user have the same last digit.

```
#include <iostream>
#include <cstdlib> // for abs()
using namespace std;

// Function to read a number with input validation
long long read_number(const string &prompt) {
    long long num;
    cout << prompt;
    while (!(cin >> num)) {
        cout << "Please enter a valid integer: ";
        cin.clear(); // clear the error flag
        cin.ignore(numeric_limits<streamsize>::max(),
'\n'); // discard invalid input
    }
    return num;
}

int main() {
    long long num1 = read_number("Enter the first number:
");
    long long num2 = read_number("Enter the second number:
");

    if (abs(num1) % 10 == abs(num2) % 10) {
        cout << "The last digit of " << num1 << " is the
same as the last digit of " << num2 << "." << endl;
    } else {
        cout << "The last digit of " << num1 << " is
different from the last digit of " << num2 << "." << endl;
    }

    return 0;
```

```
}

// Enter the first number: 3
// Enter the second number: 4
// The last digit of 3 is different from the last digit of
4.
```

23. Find HCF or GCD

This program calculates the HCF (GCD) of two positive integers entered by the user using the Euclidean algorithm.

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

// Function to read a positive number with input validation
unsigned long long read_positive_number(const string
&prompt) {
    unsigned long long num;
    cout << prompt;
    while (!(cin >> num) || num == 0) {
        cout << "Please enter a valid positive integer: ";
        cin.clear(); // clear error flag
        cin.ignore(numeric_limits<streamsize>::max(),
'\n'); // discard invalid input
    }
    return num;
}

// Function to calculate GCD using Euclidean algorithm
unsigned long long gcd(unsigned long long a, unsigned long
long b) {
    while (b != 0) {
        unsigned long long temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int main() {
    unsigned long long num1 = read_positive_number("Enter
the first positive integer: ");
    unsigned long long num2 = read_positive_number("Enter
the second positive integer: ");
```

```
    cout << "The HCF (GCD) of " << num1 << " and " << num2
<< " is: " << gcd(num1, num2) << endl;

    return 0;
}

// Example Run:
// Enter the first positive integer: 3
// Enter the second positive integer: 4
// The HCF (GCD) of 3 and 4 is: 1
```

24. Find LCM

This program calculates the LCM (Least Common Multiple) of two positive integers entered by the user using the formula `LCM(a, b) = (a * b) / GCD(a, b)`.

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

// Function to read a positive number with input validation
unsigned long long read_positive_number(const string
&prompt) {
    unsigned long long num;
    cout << prompt;
    while (!(cin >> num) || num == 0) {
        cout << "Please enter a valid positive integer: ";
        cin.clear(); // clear error flag
        cin.ignore(numeric_limits<streamsize>::max(),
'\n'); // discard invalid input
    }
    return num;
}

// Function to calculate GCD using Euclidean algorithm
unsigned long long gcd(unsigned long long a, unsigned long
long b) {
    while (b != 0) {
        unsigned long long temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

int main() {
    unsigned long long num1 = read_positive_number("Enter
the first positive integer: ");
    unsigned long long num2 = read_positive_number("Enter
the second positive integer: ");
```

```
    unsigned long long lcm = (num1 * num2) / gcd(num1,
num2);
    cout << "The LCM of " << num1 << " and " << num2 << "
is: " << lcm << endl;

    return 0;
}

// Enter the first positive integer: 3
// Enter the second positive integer: 4
// The LCM of 3 and 4 is: 12
```

25. Find the Factors of a Number

This program finds and displays all factors of a positive integer entered by the user.

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

// Function to read a positive number with input validation
unsigned long long read_positive_number(const string
&prompt) {
    unsigned long long num;
    cout << prompt;
    while (!(cin >> num) || num == 0) {
        cout << "Please enter a valid positive integer: ";
        cin.clear(); // clear error flag
        cin.ignore(numeric_limits<streamsize>::max(),
'\n'); // discard invalid input
    }
    return num;
}
int main() {
    unsigned long long number = read_positive_number("Enter
a positive integer: ");
    cout << "Factors of " << number << ":" << endl;

    // Find and display the factors
    for (unsigned long long i = 1; i <= number; i++) {
        if (number % i == 0) {
            cout << i << endl;
        }
    }

    return 0;
}
// Enter a positive integer: 3
// Factors of 3:
// 1
// 3
```

26. Find Sum of Natural Numbers Using Recursion

This program calculates the sum of the first `n` natural numbers using a recursive function.

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

// Recursive function to calculate the sum of natural
numbers
unsigned long long sum_of_natural_numbers(unsigned long
long n) {
    if (n == 0) {
        return 0;
    }
    return n + sum_of_natural_numbers(n - 1);
}

// Function to read a positive number with input validation
unsigned long long read_positive_number(const string
&prompt) {
    unsigned long long num;
    cout << prompt;
    while (!(cin >> num) || num == 0) {
        cout << "Please enter a valid positive integer: ";
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(),
'\n');
    }
    return num;
}

int main() {
    unsigned long long number = read_positive_number("Enter
a positive integer: ");
```

```
    cout << "The sum of natural numbers up to " << number
<< " is: "
         << sum_of_natural_numbers(number) << endl;

    return 0;
}

// Enter a positive integer: 5
// The sum of natural numbers up to 5 is: 15
```

27. Guess a Random Number

This program generates a random number between 1 and 100 and asks the user to guess it, giving hints whether the guess is too high or too low until the correct number is guessed.

```
#include <iostream>
#include <cstdlib>  // For rand() and srand()
#include <ctime>    // For time()
using namespace std;

// Function to read the user's guess with validation
unsigned int get_guess() {
    unsigned int guess;
    cout << "Guess the random number (1-100): ";
    if (!(cin >> guess)) {
        cin.clear(); // clear error flag
        cin.ignore(numeric_limits<streamsize>::max(),
'\n'); // discard invalid input
        return 0; // Return 0 for invalid input
    }
    return guess;
}

int main() {
    srand(time(NULL)); // Seed random number generator
    unsigned int random_number = rand() % 100 + 1; //
Random number between 1 and 100
    unsigned int attempts = 0;

    while (true) {
        unsigned int guess = get_guess();
        if (guess == 0) {
            cout << "Please enter a valid number." << endl;
            continue; // Prompt for valid guess
        }

        attempts++;
        if (guess < random_number) {
            cout << "Too low!" << endl;
```

```
        } else if (guess > random_number) {
            cout << "Too high!" << endl;
        } else {
            cout << "Congratulations! You guessed " <<
random_number
                 << " in " << attempts << " attempts." <<
endl;
            break; // Exit the loop
        }
    }

    return 0;
}

// Guess the random number (1-100): 50
// Too low!
// Guess the random number (1-100): 75
// Too high!
// Guess the random number (1-100): 63
// Congratulations! You guessed 63 in 3 attempts.
```

28. Shuffle Deck of Cards

This program creates a standard deck of 52 cards, prints it, shuffles the deck randomly, and prints the shuffled deck.

```
#include <iostream>
#include <string>
#include <cstdlib> // For rand() and srand()
#include <ctime>   // For time()
#include <algorithm> // For swap
using namespace std;

const int DECK_SIZE = 52;

// Structure for a card
struct Card {
    string rank; // Rank (2-10, Jack, Queen, King, Ace)
    string suit; // Suit (Hearts, Diamonds, Clubs, Spades)
};

// Function to create a deck of cards
void create_deck(Card deck[]) {
    string ranks[] = {"2", "3", "4", "5", "6", "7", "8",
"9", "10", "Jack", "Queen", "King", "Ace"};
    string suits[] = {"Hearts", "Diamonds", "Clubs",
"Spades"};

    int index = 0;
    for (int i = 0; i < 4; i++) {
        for (int j = 0; j < 13; j++) {
            deck[index].rank = ranks[j];
            deck[index].suit = suits[i];
            index++;
        }
    }
}

// Function to shuffle the deck of cards
void shuffle_deck(Card deck[]) {
    for (int i = DECK_SIZE - 1; i > 0; i--) {
```

```
        int j = rand() % (i + 1); // Random index
        swap(deck[i], deck[j]);    // Swap cards
    }
}

// Function to print the deck of cards
void print_deck(const Card deck[]) {
    for (int i = 0; i < DECK_SIZE; i++) {
        cout << deck[i].rank << " of " << deck[i].suit <<
endl;
    }
}

int main() {
    srand(time(NULL)); // Seed the random number generator

    Card deck[DECK_SIZE];
    create_deck(deck); // Create the deck

    cout << "Initial Deck:" << endl;
    print_deck(deck);  // Print the initial deck

    shuffle_deck(deck); // Shuffle the deck

    cout << "\nShuffled Deck:" << endl;
    print_deck(deck);   // Print the shuffled deck

    return 0;
}

// Initial Deck:
// 2 of Hearts
// 3 of Hearts
// ...
// Shuffled Deck:
// Queen of Clubs
// 7 of Diamonds
// Ace of Spades
// ...
```

29. Display Fibonacci Sequence Using Recursion

This program prints the first `n` terms of the Fibonacci sequence using a recursive function.

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

// Recursive function to calculate Fibonacci number
unsigned int fibonacci(unsigned int n) {
    if (n <= 1) return n; // Base case
    return fibonacci(n - 1) + fibonacci(n - 2); //
Recursive case
}

int main() {
    unsigned int num_terms;

    cout << "Enter the number of terms in the Fibonacci
sequence: ";
    if (!(cin >> num_terms)) {
        cout << "Please enter a valid non-negative
integer." << endl;
        return 1;
    }

    cout << "Fibonacci sequence of " << num_terms << "
terms:" << endl;
    for (unsigned int i = 0; i < num_terms; i++) {
        cout << fibonacci(i) << endl;
    }

    return 0;
}

// Enter the number of terms in the Fibonacci sequence: 6
// Fibonacci sequence of 6 terms:
// 0
```

```
// 1
// 1
// 2
// 3
// 5
```

30. Find Factorial of Number Using Recursion

This program calculates the factorial of a non-negative integer using a recursive function.

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

// Recursive function to calculate factorial
unsigned long long factorial(int n) {
    if (n == 0 || n == 1) return 1; // Base case
    return n * factorial(n - 1);    // Recursive case
}

int main() {
    int number;

    cout << "Enter a non-negative integer: ";
    if (!(cin >> number) || number < 0) {
        cout << "Please enter a valid non-negative
integer." << endl;
        return 1;
    }

    cout << "The factorial of " << number << " is: " <<
factorial(number) << endl;

    return 0;
}

// Enter a non-negative integer: 5
// The factorial of 5 is: 120
```

31. Convert Decimal to Binary

This program converts a non-negative decimal number to its binary representation using an array to store digits.

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

// Function to convert decimal to binary
void decimal_to_binary(unsigned int num) {
    if (num == 0) {
        cout << "The binary equivalent is: 0" << endl;
        return;
    }

    unsigned int binary[32]; // Array to store binary
digits
    int index = 0;

    // Convert decimal to binary
    while (num > 0) {
        binary[index++] = num % 2; // Store remainder
(binary digit)
        num /= 2; // Divide by 2
    }

    // Print binary in reverse order
    cout << "The binary equivalent is: ";
    for (int i = index - 1; i >= 0; i--) {
        cout << binary[i];
    }
    cout << endl;
}

int main() {
    unsigned int decimal_number;

    cout << "Enter a decimal number: ";
    if (!(cin >> decimal_number)) {
```

```
        cout << "Please enter a valid non-negative
integer." << endl;
        return 1;
    }

    decimal_to_binary(decimal_number);

    return 0;
}

// Enter a decimal number: 5
// The binary equivalent is: 101
```

32. Find ASCII Value of Character

This program reads a character from the user and displays its ASCII value.

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

int main() {
    char character;

    cout << "Enter a character: ";
    cin >> character;  // Read a single character

    cout << "The ASCII value of '" << character << "' is: "
<< static_cast<int>(character) << endl;

    return 0;
}

/*
Example Runs:

Enter a character: a
The ASCII value of 'a' is: 97

Enter a character: Z
The ASCII value of 'Z' is: 90
*/
```

33. Check Whether a String is Palindrome or Not

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#define MAX_LENGTH 100

// Function to check if a string is a palindrome
int is_palindrome(const char *str) {
    int left = 0;
    int right = strlen(str) - 1;

    while (left < right) {
        if (str[left] != str[right]) {
            return 0; // Not a palindrome
        }
        left++;
        right--;
    }
    return 1; // Is a palindrome
}

// Function to clean the input string
void clean_string(const char *input, char *cleaned) {
    int j = 0;
    for (int i = 0; input[i] != '\0'; i++) {
        if (isalnum(input[i])) {
            cleaned[j++] = tolower(input[i]);
        }
    }
    cleaned[j] = '\0'; // Null-terminate the cleaned string
}

int main() {
    char input[MAX_LENGTH];
    char cleaned[MAX_LENGTH];
```

```c
    // Prompt user for a string
    printf("Enter a string: ");
    fgets(input, sizeof(input), stdin);

    // Clean the string
    clean_string(input, cleaned);

    // Check if the cleaned string is a palindrome
    if (is_palindrome(cleaned)) {
        printf("\"%s\" is a palindrome.\n", input);
    } else {
        printf("\"%s\" is not a palindrome.\n", input);
    }

    return 0;
}

/*

Enter a string: A man, a plan, a canal, Panama
"A man, a plan, a canal, Panama
" is a palindrome.

Enter a string: asasd
"asasd
" is not a palindrome.
*/
```

34. Sort Words in Alphabetical Order

This program checks if a given string is a palindrome, ignoring case and non-alphanumeric characters.

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

// Function to clean the input string (remove non-
alphanumeric and convert to lowercase)
string clean_string(const string &input) {
    string cleaned;
    for (char ch : input) {
        if (isalnum(ch)) {
            cleaned += tolower(ch);
        }
    }
    return cleaned;
}

// Function to check if a string is a palindrome
bool is_palindrome(const string &str) {
    int left = 0, right = str.length() - 1;
    while (left < right) {
        if (str[left] != str[right]) return false;
        left++;
        right--;
    }
    return true;
}

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

    string cleaned = clean_string(input);
```

```
    if (is_palindrome(cleaned)) {
        cout << "\"" << input << "\" is a palindrome." <<
endl;
    } else {
        cout << "\"" << input << "\" is not a palindrome."
<< endl;
    }

    return 0;
}

/*
Example Runs:

Enter a string: A man, a plan, a canal, Panama
"A man, a plan, a canal, Panama" is a palindrome.

Enter a string: asasd
"asasd" is not a palindrome.
*/
```

35. Replace Characters of a String

```c
#include <stdio.h>
#include <string.h>

void replace_characters(char *str, char target, char
replacement) {
    for (int i = 0; str[i] != '\0'; i++) {
        if (str[i] == target) {
            str[i] = replacement;
        }
    }
}

int main() {
    char input_string[100];
    char target_char, replacement_char;

    // Prompt user for a string
    printf("Enter a string: ");
    fgets(input_string, sizeof(input_string), stdin);
    input_string[strcspn(input_string, "\n")] = 0; // Remove
newline character

    if (strlen(input_string) == 0) {
        printf("Please enter a valid string.\n");
        return 1;
    }

    // Prompt user for target and replacement characters
    printf("Enter the target character: ");
    scanf(" %c", &target_char); // Leading space to consume
any newline

    printf("Enter the replacement character: ");
    scanf(" %c", &replacement_char);

    // Replace characters in the input string
          replace_characters(input_string,    target_char,
replacement_char);
```

```c
    // Display the modified string
    printf("Modified String: %s\n", input_string);

    return 0;
}

/*
Enter a string: Hello World!
Enter the target character: o
Enter the replacement character: O
Modified String: HellO WOrld!
*/
```

36. Reverse a String

This program reverses a string entered by the user.

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

// Function to reverse a string in place
void reverse_string(string &str) {
    int length = str.length();
    for (int i = 0; i < length / 2; i++) {
        swap(str[i], str[length - i - 1]);
    }
}

int main() {
    string input_string;

    // Prompt user for a string
    cout << "Enter a string: ";
    getline(cin, input_string);

    if (input_string.empty()) {
        cout << "Please enter a valid string." << endl;
        return 1;
    }

    // Reverse the string
    reverse_string(input_string);

    // Display the reversed string
    cout << "Reversed String: " << input_string << endl;

    return 0;
}

// Enter a string: Hello World!
// Reversed String: !dlroW olleH
```

37. Check the Number of Occurrences of a Character in the String

This program counts how many times a specific character appears in a user-provided string.

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

int main() {
    string input_string;
    char target_char;
    int count = 0;

    // Prompt user for a string
    cout << "Enter a string: ";
    getline(cin, input_string);

    if (input_string.empty()) {
        cout << "Please enter a valid string." << endl;
        return 1;
    }

    // Prompt user for the character to count
    cout << "Enter the character to count: ";
    cin >> target_char;

    // Count occurrences of the target character
    for (char c : input_string) {
        if (c == target_char) {
            count++;
        }
    }

    // Display the result
    cout << "Number of occurrences of '" << target_char <<
"' in '" << input_string << "': " << count << endl;
```

```
    return 0;
}

// Enter a string: Hello World!
// Enter the character to count: l
// Number of occurrences of 'l' in 'Hello World!': 3
```

38. Convert the First Letter of a String into UpperCase

This program reads a string from the user and converts only the first character of the string to uppercase, leaving the rest of the string unchanged. It handles empty strings gracefully.

```
#include <iostream>
#include <string>
#include <cctype>  // For toupper
using namespace std;

// Function to capitalize the first letter
void capitalizeFirstLetter(string &str) {
    if (!str.empty()) {
        str[0] = toupper(str[0]);
    }
}

int main() {
    string input_string;

    // Prompt user for a string
    cout << "Enter a string: ";
    getline(cin, input_string);

    if (!input_string.empty()) {
        // Capitalize the first letter
        capitalizeFirstLetter(input_string);

        // Display the result
        cout << "String with First Letter Uppercase: " <<
input_string << endl;
    } else {
        cout << "Please enter a valid string." << endl;
    }

    return 0;
```

```
}

/*
Enter a string: hello
String with First Letter Uppercase: Hello

Enter a string: asd
String with First Letter Uppercase: Asd
*/
```

39. Count the Number of Vowels in a String

This program reads a string from the user and counts the total number of vowels (`a`, `e`, `i`, `o`, `u`) present in the string. It considers both uppercase and lowercase vowels and handles empty strings gracefully.

```
#include <iostream>
#include <string>
#include <cctype>

int count_vowels(const std::string &str) {
    int count = 0;
    for (char ch : str) {
        char lower_ch = std::tolower(ch); // Convert to
lowercase for uniformity
        if (lower_ch == 'a' || lower_ch == 'e' || lower_ch
== 'i' ||
            lower_ch == 'o' || lower_ch == 'u') {
            count++;
        }
    }
    return count;
}

int main() {
    std::string input_string;

    // Prompt user for a string
    std::cout << "Enter a string: ";
    std::getline(std::cin, input_string);

    // Check if input is valid
    if (!input_string.empty()) {
        int number_of_vowels = count_vowels(input_string);
        std::cout << "Number of vowels in '" <<
input_string << "': "
```

```
                << number_of_vowels << std::endl;
    } else {
        std::cout << "Please enter a valid string." <<
std::endl;
    }

    return 0;
}

/*
Enter a string: Hello World
Number of vowels in 'Hello World': 3
*/
```

40. Check Whether a String Starts and Ends with Certain Characters

This program reads a string from the user and checks whether it starts with a specified set of characters and ends with another specified set. The user provides both the starting and ending characters, and the program validates the string accordingly.

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

// Function to check if a string starts with a given prefix
bool starts_with(const std::string &str, const std::string
&prefix) {
    if (prefix.size() > str.size()) return false;
    return str.substr(0, prefix.size()) == prefix;
}

// Function to check if a string ends with a given suffix
bool ends_with(const std::string &str, const std::string
&suffix) {
    if (suffix.size() > str.size()) return false;
    return str.substr(str.size() - suffix.size()) ==
suffix;
}

int main() {
    std::string input_string, start_chars, end_chars;

    // Prompt user for main string
    std::cout << "Enter a string: ";
    std::getline(std::cin, input_string);

    if (input_string.empty()) {
        std::cout << "Please enter a valid string." <<
std::endl;
        return 1;
    }
```

```
    // Prompt user for starting and ending characters
    std::cout << "Enter the starting characters: ";
    std::getline(std::cin, start_chars);

    std::cout << "Enter the ending characters: ";
    std::getline(std::cin, end_chars);

    // Check start and end
    if (starts_with(input_string, start_chars) &&
ends_with(input_string, end_chars)) {
        std::cout << "The string '" << input_string
                  << "' starts with '" << start_chars
                  << "' and ends with '" << end_chars <<
"'." << std::endl;
    } else {
        std::cout << "The string '" << input_string
                  << "' does not start with '" <<
start_chars
                  << "' or end with '" << end_chars << "'."
<< std::endl;
    }

    return 0;
}

/*
Enter a string: asd
Enter the starting characters: a
Enter the ending characters: d
The string 'asd' starts with 'a' and ends with 'd'.
*/
```

41. Replace All Occurrences of a String

This program searches for all occurrences of a specific substring within a string and replaces them with another substring provided by the user. It handles multiple occurrences and constructs a new modified string.

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

// Function to replace all occurrences of a substring in a
string
std::string replace_all_occurrences(const std::string
&original, const std::string &search, const std::string
&replacement) {
    std::string result = original;
    size_t pos = 0;

    // Loop until no more occurrences are found
    while ((pos = result.find(search, pos)) !=
std::string::npos) {
        result.replace(pos, search.length(), replacement);
        pos += replacement.length(); // Move past the
replacement
    }

    return result;
}

int main() {
    std::string original_string = "Hello world, world!";
    std::string search_string = "world";
    std::string replacement_string = "universe";

    // Replace all occurrences
    std::string modified_string =
replace_all_occurrences(original_string, search_string,
replacement_string);
```

```
    // Display results
    std::cout << "Original String: " << original_string <<
std::endl;
    std::cout << "Modified String: " << modified_string <<
std::endl;

    return 0;
}

/*
Original String: Hello world, world!
Modified String: Hello universe, universe!
*/
```

42. Create Multiline Strings

This C++ program demonstrates how to create and display a string that spans multiple lines. It uses C++ `std::string` and escape sequences (`\n`) to format line breaks easily.

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

int main() {
    // Multiline string using escape sequences
    std::string multiline_string =
        "This is a multiline string.\n"
        "It spans multiple lines.\n"
        "You can include line breaks and indentation
easily.\n";

    // Print the multiline string
    std::cout << multiline_string;

    return 0;
}

/*
Example Output:
This is a multiline string.
It spans multiple lines.
You can include line breaks and indentation easily.
*/
```

43. Format Numbers as Currency Strings

This C++ program formats a floating-point number as a currency string, adding thousand separators and rounding to two decimal places. It uses `std::ostringstream` for flexible string formatting.

```
#include <iostream>
#include <iomanip>
#include <sstream>
#include <string>

std::string format_currency(double amount) {
    std::ostringstream oss;

    // Handle negative amounts
    if (amount < 0) {
        oss << "-";
        amount = -amount;
    }

    // Separate integer and fractional parts
    long long integer_part = static_cast<long
long>(amount);
    int fractional_part = static_cast<int>((amount -
integer_part) * 100 + 0.5);

    // Format integer part with thousand separators
    std::string int_str;
    while (integer_part > 0) {
        int_str = std::to_string(integer_part % 1000) +
(int_str.empty() ? "" : ",") + int_str;
        integer_part /= 1000;
    }
    if (int_str.empty()) int_str = "0";

    // Build final formatted string
    oss << "$" << int_str << "." << std::setw(2) <<
std::setfill('0') << fractional_part;
    return oss.str();
}
```

```
int main() {
    double amount = 1234567.89; // Example amount
    std::string formatted_amount = format_currency(amount);

    std::cout << "Formatted Amount: " << formatted_amount
<< std::endl;
    return 0;
}

/*
Formatted Amount: $1,234,567.89
*/
```

44. Generate Random String

This program generates a random alphanumeric string of a specified length using C++'s `<random>` library for better randomness and thread safety.

```
#include <iostream>
#include <random>
#include <string>

std::string generate_random_string(size_t length) {
    const std::string characters =
        "ABCDEFGHIJKLMNOPQRSTUVWXYZ"
        "abcdefghijklmnopqrstuvwxyz"
        "0123456789";

    std::random_device rd;   // Random device for seeding
    std::mt19937 generator(rd()); // Mersenne Twister
random generator
    std::uniform_int_distribution<> distribution(0,
characters.size() - 1);

    std::string random_string;
    random_string.reserve(length);

    for (size_t i = 0; i < length; ++i) {
        random_string +=
characters[distribution(generator)];
    }

    return random_string;
}

int main() {
    const size_t STRING_LENGTH = 8;

    std::string random_string =
generate_random_string(STRING_LENGTH);
    std::cout << "Random String: " << random_string <<
std::endl;
```

```
    return 0;
}

/*
Random String: jQmNYWWC
*/
```

45. Check if a String Starts with Another String

This program checks whether a given string starts with another specified string. It uses `std::string` for safer and easier string manipulation in C++.

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

bool starts_with(const std::string &main_str, const
std::string &search_str) {
    if (main_str.size() < search_str.size()) return false;
    return main_str.substr(0, search_str.size()) ==
search_str;
}

int main() {
    std::string main_string = "Hello, World!";
    std::string search_string = "Hello";

    bool result = starts_with(main_string, search_string);
    std::cout << "Does the string start with '" <<
search_string << "'? "
              << (result ? "true" : "false") << std::endl;

    return 0;
}

/*
Does the string start with 'Hello'? true
*/
```

46. Trim a String

This program removes leading and trailing whitespace from a string using modern C++ facilities (`std::string` and standard library functions).

```
#include <iostream>
#include <string>
#include <cctype>

// Function to trim leading and trailing whitespace
std::string trim(const std::string &str) {
    size_t start = 0;
    size_t end = str.size();

    // Find first non-whitespace character
    while (start < end && std::isspace(static_cast<unsigned
char>(str[start]))) {
        ++start;
    }

    // Find last non-whitespace character
    while (end > start && std::isspace(static_cast<unsigned
char>(str[end - 1]))) {
        --end;
    }

    return str.substr(start, end - start);
}

int main() {
    std::string string_with_spaces = "   Hello, World!   ";

    std::string trimmed = trim(string_with_spaces);

    std::cout << "Original String: '" << string_with_spaces
<< "'" << std::endl;
    std::cout << "Trimmed String: '" << trimmed << "'" <<
std::endl;
```

```
    return 0;
}

/*
Original String: '   Hello, World!   '
Trimmed String: 'Hello, World!'
*/
```

47. Check Whether a String Contains a Substring

This program checks if a string contains a specified substring using `std::string::find` in C++.

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

int main() {
    std::string main_string = "Hello, World!";
    std::string substring_to_check = "World";

    // Check if main_string contains substring_to_check
    bool contains = main_string.find(substring_to_check) !=
std::string::npos;

    // Display the result
    std::cout << "Does the string contain '" <<
substring_to_check << "'? "
              << (contains ? "true" : "false") <<
std::endl;

    return 0;
}

/*
Does the string contain 'World'? true
*/
```

48. Compare Two Strings

This program compares two strings for both case-sensitive and case-insensitive equality using C++ `std::string` and algorithms.

```
#include <iostream>
#include <string>
#include <algorithm>

// Function to perform case-insensitive comparison
bool case_insensitive_compare(const std::string &str1,
const std::string &str2) {
    if (str1.size() != str2.size()) return false;

    for (size_t i = 0; i < str1.size(); ++i) {
        if (tolower(str1[i]) != tolower(str2[i])) return
false;
    }
    return true;
}

int main() {
    std::string string1 = "Hello";
    std::string string2 = "hello";

    // Case-sensitive comparison
    bool case_sensitive_comparison = (string1 == string2);

    // Case-insensitive comparison
    bool case_insensitive_comparison =
case_insensitive_compare(string1, string2);

    // Display the results
    std::cout << "Case-sensitive comparison: "
              << (case_sensitive_comparison ? "true" :
"false") << std::endl;
    std::cout << "Case-insensitive comparison: "
              << (case_insensitive_comparison ? "true" :
"false") << std::endl;
```

```
    return 0;
}

/*
Case-sensitive comparison: false
Case-insensitive comparison: true
*/
```

49. Encode a String to Base64

This program encodes a given string into Base64 format. It handles memory safely using `std::string` and supports standard Base64 characters.

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

// Base64 encoding table
const std::string base64_table =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz012345
6789+/";

// Function to encode a string to Base64
std::string base64_encode(const std::string& input) {
    std::string encoded;
    int val = 0, valb = -6;

    for (unsigned char c : input) {
        val = (val << 8) + c;
        valb += 8;
        while (valb >= 0) {
            encoded.push_back(base64_table[(val >> valb) &
0x3F]);
            valb -= 6;
        }
    }

    if (valb > -6) encoded.push_back(base64_table[((val <<
8) >> (valb + 8)) & 0x3F]);

    while (encoded.size() % 4) encoded.push_back('=');

    return encoded;
}

int main() {
    std::string original_string = "Hello, 你好!";
```

```
    // Encode the string to Base64
    std::string encoded_string =
base64_encode(original_string);

    // Display the results
    std::cout << "Original String: " << original_string <<
std::endl;
    std::cout << "Base64 Encoded String: " <<
encoded_string << std::endl;

    return 0;
}

/*
Original String: Hello, 你好!
Base64 Encoded String: SGVsbG8sIOS9oOWlvSEh
*/
```

50. Replace all Instances of a Character in a String

This program replaces all occurrences of a specified character in a string with another character, using C++'s `std::string` for safety and simplicity.

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

// Function to replace all instances of a character in a
string
std::string replace_char(const std::string& str, char
char_to_replace, char replacement_char) {
    std::string modified = str;
    for (char& c : modified) {
        if (c == char_to_replace) {
            c = replacement_char;
        }
    }
    return modified;
}

int main() {
    std::string original_string = "Hello, World!";
    char char_to_replace = 'l';
    char replacement_char = 'x';

    // Replace all instances of char_to_replace with
replacement_char
    std::string modified_string =
replace_char(original_string, char_to_replace,
replacement_char);

    // Display the result
    std::cout << "Original String: " << original_string <<
std::endl;
    std::cout << "Modified String: " << modified_string <<
std::endl;
```

```
    return 0;
}

/*
Original String: Hello, World!
Modified String: Hexxo, Worxd!
*/
```

51. Replace All Line Breaks with

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

// Function to replace all line breaks in a string
char* replace_line_breaks(const char* str, const char*
replacement) {
    // Calculate the size for the new string
    size_t str_len = strlen(str);
    size_t replacement_len = strlen(replacement);

    // Allocate enough space for the modified string
    // Worst case: every character is a line break, so we
need extra space
      char* modified_string = (char*)malloc(str_len *
replacement_len + 1);
    if (modified_string == NULL) {
        fprintf(stderr, "Memory allocation failed\n");
        exit(1);
    }

    size_t j = 0; // Index for modified string
    for (size_t i = 0; i < str_len; i++) {
        if (str[i] == '\n') {
           // Replace line break with the replacement string
            for (size_t k = 0; k < replacement_len; k++) {
                modified_string[j++] = replacement[k];
            }
        } else {
            modified_string[j++] = str[i];
        }
    }
   modified_string[j] = '\0'; // Null-terminate the modified
string

    return modified_string;
}
```

```c
int main() {
    // Example string with line breaks
            const     char*     string_with_line_breaks     =
"Hello,\nWorld!\nThis is a new line.";

    // Replacement string
    const char* replacement_string = "-";

    // Replace all line breaks with the replacement string
               char*       string_without_line_breaks       =
replace_line_breaks(string_with_line_breaks,
replacement_string);

    // Display the result
                    printf("Original         String:\n%s\n",
string_with_line_breaks);
        printf("\nString   without   Line   Breaks:\n%s\n",
string_without_line_breaks);

    // Free allocated memory
    free(string_without_line_breaks);

    return 0;
}

/*
Example Output:
Original String:
Hello,
World!
This is a new line.

String without Line Breaks:
Hello,-World!-This is a new line.
*/
```

52. Check Leap Year

This program replaces all newline characters (`\n`) in a string with a specified replacement string using C++ `std::string`.

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

// Function to replace all line breaks in a string
std::string replace_line_breaks(const std::string& str,
const std::string& replacement) {
    std::string modified;
    for (char c : str) {
        if (c == '\n') {
            modified += replacement;
        } else {
            modified += c;
        }
    }
    return modified;
}

int main() {
    std::string string_with_line_breaks =
"Hello,\nWorld!\nThis is a new line.";
    std::string replacement_string = "-";

    // Replace all line breaks with the replacement string
    std::string string_without_line_breaks =
replace_line_breaks(string_with_line_breaks,
replacement_string);

    // Display the result
    std::cout << "Original String:\n" <<
string_with_line_breaks << "\n\n";
    std::cout << "String without Line Breaks:\n" <<
string_without_line_breaks << std::endl;

    return 0;
}
```

```
/*
Original String:
Hello,
World!
This is a new line.

String without Line Breaks:
Hello,-World!-This is a new line.
*/
```

53. Format the Date

This program gets the current local date and formats it as a human-readable string, e.g., "Friday, September 20, 2024".

```
#include <iostream>
#include <iomanip>
#include <chrono>
#include <ctime>

int main() {
    // Get current time as time_t
    auto now = std::chrono::system_clock::now();
    std::time_t now_c =
std::chrono::system_clock::to_time_t(now);

    // Convert to local time
    std::tm local_time = *std::localtime(&now_c);

    // Format and display the date
    std::cout << "Formatted Date: "
              << std::put_time(&local_time, "%A, %B %d,
%Y")
              << std::endl;

    return 0;
}

/*
Formatted Date: Friday, September 20, 2024
*/
```

54. Display Current Date

This program retrieves the current date and displays it in `MM/DD/YYYY` format using modern C++ features.

```
#include <iostream>
#include <iomanip>
#include <chrono>
#include <ctime>

int main() {
    // Get current time as time_point
    auto now = std::chrono::system_clock::now();
    std::time_t now_c =
std::chrono::system_clock::to_time_t(now);

    // Convert to local time
    std::tm local_time = *std::localtime(&now_c);

    // Format and display the current date as MM/DD/YYYY
    std::cout << "Current Date: "
              << std::put_time(&local_time, "%m/%d/%Y")
              << std::endl;

    return 0;
}

/*
Current Date: 09/20/2024
*/
```

55. Compare The Value of Two Dates

This program compares two dates and determines whether the first date is earlier, later, or equal to the second date using modern C++ date/time facilities.

```
#include <iostream>
#include <chrono>

int main() {
    // Define two dates using std::tm
    std::tm date1_tm = {};
    std::tm date2_tm = {};

    // Initialize date1 to January 1, 2022
    date1_tm.tm_year = 2022 - 1900; // tm_year is years
since 1900
    date1_tm.tm_mon = 0;            // January (0-based)
    date1_tm.tm_mday = 1;           // 1st day

    // Initialize date2 to January 1, 2023
    date2_tm.tm_year = 2023 - 1900;
    date2_tm.tm_mon = 0;
    date2_tm.tm_mday = 1;

    // Convert std::tm to time_point for comparison
    std::time_t timestamp1 = std::mktime(&date1_tm);
    std::time_t timestamp2 = std::mktime(&date2_tm);

    // Compare the two dates
    if (timestamp1 < timestamp2) {
        std::cout << "Date1 ("
                  << date1_tm.tm_year + 1900 << "-" <<
date1_tm.tm_mon + 1 << "-" << date1_tm.tm_mday
                  << ") is earlier than Date2 ("
                  << date2_tm.tm_year + 1900 << "-" <<
date2_tm.tm_mon + 1 << "-" << date2_tm.tm_mday
                  << ")" << std::endl;
    } else if (timestamp1 > timestamp2) {
        std::cout << "Date1 ("
```

```
                  << date1_tm.tm_year + 1900 << "-" <<
date1_tm.tm_mon + 1 << "-" << date1_tm.tm_mday
                  << ") is later than Date2 ("
                  << date2_tm.tm_year + 1900 << "-" <<
date2_tm.tm_mon + 1 << "-" << date2_tm.tm_mday
                  << ")" << std::endl;
    } else {
        std::cout << "Date1 ("
                  << date1_tm.tm_year + 1900 << "-" <<
date1_tm.tm_mon + 1 << "-" << date1_tm.tm_mday
                  << ") is equal to Date2 ("
                  << date2_tm.tm_year + 1900 << "-" <<
date2_tm.tm_mon + 1 << "-" << date2_tm.tm_mday
                  << ")" << std::endl;
    }

    return 0;
}

/*
Date1 (2022-1-1) is earlier than Date2 (2023-1-1)
*/
```

56. Create Countdown Timer

This program creates a countdown timer that counts down from a specified duration (e.g., 60 seconds). It updates the remaining days, hours, minutes, and seconds in real-time until the countdown expires.

```
#include <iostream>
#include <chrono>
#include <thread>

int main() {
    // Set the countdown duration (in seconds)
    int countdown_seconds = 60;

    // Calculate the target time point
    auto target_time = std::chrono::system_clock::now() +
std::chrono::seconds(countdown_seconds);

    while (true) {
        // Calculate remaining time
        auto now = std::chrono::system_clock::now();
        auto time_diff =
std::chrono::duration_cast<std::chrono::seconds>(target_tim
e - now).count();

        // Break if the countdown has expired
        if (time_diff <= 0) {
            std::cout << "\nCountdown expired!" <<
std::endl;
            break;
        }

        // Calculate days, hours, minutes, and seconds
        int days = time_diff / 86400;
        int hours = (time_diff % 86400) / 3600;
        int minutes = (time_diff % 3600) / 60;
        int seconds = time_diff % 60;

        // Display the countdown
```

```
        std::cout << "\rCountdown: " << days << "d " <<
hours << "h "
                  << minutes << "m " << seconds << "s" <<
std::flush;

        // Sleep for 1 second
        std::this_thread::sleep_for(std::chrono::seconds(1)
);
    }

    return 0;
}

/*
Countdown: 0d 0h 0m 56s
Countdown: 0d 0h 0m 55s
...
Countdown expired!
*/
```

57. Remove Specific Item from an Array

This program removes all occurrences of a specific item from an integer array. It shifts remaining elements to maintain array order and updates the array size.

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    // Original array
    std::vector<int> arr = {1, 2, 3, 4, 5};
    int item_to_remove = 3;

    // Remove all occurrences of item_to_remove
    arr.erase(std::remove(arr.begin(), arr.end(),
item_to_remove), arr.end());

    // Display the updated array
    std::cout << "Updated List: ";
    for (int num : arr) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}

/*
Updated List: 1 2 4 5
*/
```

58. Check if an Array Contains a Specified Value

This program checks whether a given integer array contains a specific value. It returns true if the value is found, otherwise false.

```
#include <iostream>
#include <vector>
#include <algorithm>

int main() {
    // Example array
    std::vector<int> arr = {1, 2, 3, 4, 5};
    int value_to_check = 3;

    // Check if the array contains the value
    bool contains_value_result = std::find(arr.begin(),
arr.end(), value_to_check) != arr.end();

    // Display the result
    std::cout << "Does the list include " << value_to_check
<< "? "
              << (contains_value_result ? "true" : "false")
<< std::endl;

    return 0;
}

/*
Does the list include 3? true
*/
```

59. Insert Item in an Array

This program inserts an item at the end of an array (or vector in C++). It updates the array size dynamically and displays the modified array.

```cpp
#include <iostream>
#include <vector>

int main() {
    // Example array (vector)
    std::vector<int> arr = {1, 2, 3, 4, 5};
    int item_to_insert = 6;

    // Insert the item at the end
    arr.push_back(item_to_insert);

    // Display the result
    std::cout << "List after inserting: [";
    for (size_t i = 0; i < arr.size(); i++) {
        std::cout << arr[i];
        if (i < arr.size() - 1) std::cout << ", ";
    }
    std::cout << "]" << std::endl;

    return 0;
}

/*
List after inserting: [1, 2, 3, 4, 5, 6]
*/
```

60. Get Random Item from an Array

This program selects a random item from an array (or vector in C++) and displays it. It uses C++'s `<cstdlib>` and `<ctime>` to generate random numbers.

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>

int main() {
    // Example array (vector)
    std::vector<int> arr = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

    // Seed the random number generator
    std::srand(std::time(nullptr));

    // Get a random index
    int random_index = std::rand() % arr.size();

    // Get the random item
    int random_item = arr[random_index];

    // Display the result
    std::cout << "Random Item: " << random_item <<
std::endl;

    return 0;
}

/*
Random Item: 8
*/
```

61. Perform Intersection Between Two Arrays

This program finds common elements between two arrays (or vectors in C++) and stores the result in a new array/vector without duplicates.

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

int main() {
    // Example arrays
    std::vector<int> list1 = {1, 2, 3, 4, 5};
    std::vector<int> list2 = {3, 4, 5, 6, 7};

    std::vector<int> intersection;

    // Find intersection
    for (int i : list1) {
        for (int j : list2) {
            if (i == j) {
                // Check for duplicates
                bool already_exists = false;
                for (int val : intersection) {
                    if (val == i) {
                        already_exists = true;
                        break;
                    }
                }
                if (!already_exists) {
                    intersection.push_back(i);
                }
            }
        }
    }

    // Display the result
    std::cout << "Intersection: {";
    for (size_t i = 0; i < intersection.size(); i++) {
```

```
        std::cout << intersection[i];
        if (i < intersection.size() - 1) {
            std::cout << ", ";
        }
    }
    std::cout << "}" << std::endl;

    return 0;
}

/*
Intersection: {3, 4, 5}
*/
```

62. Split Array into Smaller Chunks

This program splits an array/vector into smaller sub-arrays (chunks) of a specified size and prints them.

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

void chunk_array(const std::vector<int>& array, int
chunk_size) {
    int size = array.size();
    int chunks = (size + chunk_size - 1) / chunk_size; //
Calculate number of chunks

    // Display the original array
    std::cout << "Original Array: {";
    for (size_t i = 0; i < array.size(); i++) {
        std::cout << array[i];
        if (i < array.size() - 1) std::cout << ", ";
    }
    std::cout << "}" << std::endl;

    // Display the chunks
    std::cout << "Chunks:" << std::endl;
    for (int i = 0; i < chunks; i++) {
        std::cout << "{";
        for (int j = 0; j < chunk_size && (i * chunk_size +
j) < size; j++) {
            std::cout << array[i * chunk_size + j];
            if (j < chunk_size - 1 && (i * chunk_size + j +
1) < size) {
                std::cout << ", ";
            }
        }
        std::cout << "}" << std::endl;
    }
}

int main() {
    // Example array
```

```
    std::vector<int> my_array = {1,2,3,4,5,6,7,8,9,10};

    // Split the array into chunks of size 3
    chunk_array(my_array, 3);

    return 0;
}

/*
Original Array: {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
Chunks:
{1, 2, 3}
{4, 5, 6}
{7, 8, 9}
{10}
*/
```

63. Get File Extension

This program retrieves the file extension from a given filename.

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

std::string get_file_extension(const std::string&
file_name) {
    size_t dot_pos = file_name.find_last_of('.');
    if (dot_pos != std::string::npos && dot_pos != 0) {
        return file_name.substr(dot_pos + 1); // Return
substring after the dot
    }
    return ""; // No extension found
}

int main() {
    // Example file name
    std::string file_name = "example.txt";

    // Get the file extension
    std::string extension = get_file_extension(file_name);

    // Display the result
    if (!extension.empty()) {
        std::cout << "File Extension: " << extension <<
std::endl;
    } else {
        std::cout << "No file extension found." <<
std::endl;
    }

    return 0;
}

/*
File Extension: txt
*/
```

64. Check If a Variable Is undefined or null

This program demonstrates how to check whether a pointer variable in C++ is `nullptr`, which indicates that it is uninitialized or has no assigned memory address.

```
#include <iostream>

int main() {
    // Define a pointer to an int (initially nullptr)
    int* variable = nullptr;

    // Check if the variable is nullptr
    if (variable != nullptr) {
        std::cout << "Variable has a value" << std::endl;
    } else {
        std::cout << "Variable is nullptr" << std::endl;
    }

    return 0;
}

/*
Variable is nullptr
*/
```

65. Generate a Random Number Between Two Numbers

This program generates a random integer within a specified range `[min_val, max_val]` using C++'s `<random>` library to provide better randomness than `rand()`.

```
#include <iostream>
#include <random>
#include <ctime>

// Function to generate a random number between min_val and
max_val (inclusive)
int get_random_number(int min_val, int max_val) {
    static std::mt19937 rng(static_cast<unsigned
int>(time(nullptr))); // Mersenne Twister RNG
    std::uniform_int_distribution<int> dist(min_val,
max_val);
    return dist(rng);
}

int main() {
    // Generate a random number between 1 and 100
    int random_num = get_random_number(1, 100);

    // Print the random number
    std::cout << "Random Number: " << random_num <<
std::endl;

    return 0;
}

/*
Expected output (will vary each time):
Random Number: 48
*/
```

66. Longest Daily Streak

This program calculates the longest consecutive daily streak of logins from an array representing daily login activity (`1` for logged in, `0` for not).

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

// Function to calculate the longest daily login streak
int daily_streak(const std::vector<int>& logins) {
    int max_streak = 0;
    int current_streak = 0;

    for (int status : logins) {
        if (status == 1) {  // Logged in
            current_streak++;
            if (current_streak > max_streak) {
                max_streak = current_streak; // Update max
streak
            }
        } else {
            current_streak = 0; // Reset streak if not
logged in
        }
    }

    return max_streak;
}

int main() {
    // Test cases
    std::vector<int> logins1 = {1, 1, 0, 1};
    std::vector<int> logins2 = {0, 0, 0};
    std::vector<int> logins3 = {1, 1, 1, 0, 1, 1};

    std::cout << daily_streak(logins1) << std::endl;  //
Outputs: 2
```

```
    std::cout << daily_streak(logins2) << std::endl;  //
Outputs: 0
    std::cout << daily_streak(logins3) << std::endl;  //
Outputs: 3

    return 0;
}

/*
2
0
3
*/
```

67. Validate an Email Address

This program validates an email address using a regular expression. It checks the format of the email to ensure it contains a username, an `@` symbol, a domain, and a top-level domain.

```
#include <iostream>
#include <regex>
#include <string>

// Function to validate an email address using regex
bool validate_email(const std::string& email) {
    // Define a basic regex pattern for email validation
    const std::regex
pattern(R"(^[^\s@]+@[^\s@]+\.[^\s@]+$)");

    // Check if the email matches the pattern
    return std::regex_match(email, pattern);
}

int main() {
    // Example usage
    std::string email_to_validate = "example@email.com";

    if (validate_email(email_to_validate)) {
        std::cout << "Email is valid" << std::endl;
    } else {
        std::cout << "Email is not valid" << std::endl;
    }

    return 0;
}

/*
Email is valid
*/
```

68. Record Temperatures

This program updates record low and high temperatures for a week by comparing the current week's temperatures with the existing records. It ensures that the record low and high are preserved or updated appropriately.

```
#include <iostream>
#include <array>

using namespace std;

// Function to update the record temperatures
void record_temps(array<array<int, 2>, 7>& record, const
array<array<int, 2>, 7>& current) {
    for (int i = 0; i < 7; i++) {
        // Update record low if the current daily low is
lower
        if (current[i][0] < record[i][0]) {
            record[i][0] = current[i][0];
        }
        // Update record high if the current daily high is
higher
        if (current[i][1] > record[i][1]) {
            record[i][1] = current[i][1];
        }
    }
}

int main() {
    // Initial record temperatures
    array<array<int, 2>, 7> records = {{
        {34, 82}, {24, 82}, {20, 89}, {5, 88}, {9, 88},
{26, 89}, {27, 83}
    }};

    // Current week's temperatures
    array<array<int, 2>, 7> current_week_temps = {{
        {44, 72}, {19, 70}, {40, 69}, {39, 68}, {33, 64},
{36, 70}, {38, 69}
```

```
    }};

    // Update the records with the current week's
temperatures
    record_temps(records, current_week_temps);

    // Print the updated record temperatures
    cout << "Updated Record Temperatures:" << endl;
    for (const auto& day : records) {
        cout << "[" << day[0] << ", " << day[1] << "]" <<
endl;
    }

    return 0;
}

/*
Updated Record Temperatures:
[34, 82]
[19, 82]
[20, 89]
[5, 88]
[9, 88]
[26, 89]
[27, 83]
*/
```

69. Kaprekar Numbers

This program checks whether a given number is a Kaprekar number. A Kaprekar number is a number whose square can be split into two parts that sum to the original number.

```
#include <iostream>
#include <string>
#include <cmath>

using namespace std;

// Function to check if a number is a Kaprekar number
bool is_kaprekar(unsigned long n) {
    if (n == 0 || n == 1) return true; // 0 and 1 are
Kaprekar numbers

    // Calculate the square of n
    unsigned long square = n * n;

    // Convert the square to a string to split into left
and right parts
    string square_str = to_string(square);
    int len = square_str.length();

    string left_str = "";
    string right_str = "";

    if (len == 1) {
        right_str = square_str; // Only right part if
length is 1
    } else {
        int split_index = len / 2;
        left_str = square_str.substr(0, len -
split_index);   // Left part
        right_str = square_str.substr(len -
split_index);     // Right part
    }

    // Convert left and right parts to numbers
```

```
    unsigned long left = !left_str.empty() ?
stoul(left_str) : 0;
    unsigned long right = !right_str.empty() ?
stoul(right_str) : 0;

    // Check if the sum of left and right equals n
    return (left + right == n);
}

int main() {
    // Test cases
    cout << "is_kaprekar(3) → " << (is_kaprekar(3) ?
"true" : "false") << endl;
    cout << "is_kaprekar(5) → " << (is_kaprekar(5) ?
"true" : "false") << endl;
    cout << "is_kaprekar(297) → " << (is_kaprekar(297) ?
"true" : "false") << endl;

    return 0;
}

/*
is_kaprekar(3) → false
is_kaprekar(5) → false
is_kaprekar(297) → true
*/
```

70. Pass Parameter to a threading.Timer Function

This C++ program demonstrates how to pass parameters to a function executed after a delay using `pthread`.

```
#include <iostream>
#include <thread>
#include <chrono>
#include <string>

using namespace std;

// Function to be executed after a delay
void my_function(const string& parameter) {
    cout << "Parameter received: " << parameter << endl;
}

// Function to execute after a delay
void delayed_execution(string parameter, int delay_seconds)
{
    this_thread::sleep_for(chrono::seconds(delay_seconds));
    my_function(parameter);
}

int main() {
    // Define the parameter
    string my_parameter = "Hello, world!";

    // Define the delay in seconds
    int delay = 1;

    // Create a thread to run the delayed_execution
function
    thread t(delayed_execution, my_parameter, delay);

    // Wait for the thread to finish
    t.join();
```

```
    return 0;
}

/*
Parameter received: Hello, world!
*/
```

71. Generate a Range of Numbers and Characters

This program generates a range of numbers or characters inclusively and stores them in a vector. It then prints the generated range.

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

using namespace std;

// Function to generate a range of numbers (inclusive)
vector<unsigned int> generate_number_range(unsigned int
start, unsigned int end) {
    vector<unsigned int> range;
    for (unsigned int i = start; i <= end; ++i) {
        range.push_back(i);
    }
    return range;
}

// Function to generate a range of characters (inclusive)
vector<char> generate_char_range(char start, char end) {
    vector<char> range;
    for (char c = start; c <= end; ++c) {
        range.push_back(c);
    }
    return range;
}

int main() {
    // Generate numbers from 1 to 5
    vector<unsigned int> number_range =
generate_number_range(1, 5);
    cout << "Number Range: ";
    for (auto n : number_range) {
        cout << n << " ";
    }
```

```
    cout << endl;

    // Generate characters from 'a' to 'e'
    vector<char> char_range = generate_char_range('a',
'e');
    cout << "Character Range: ";
    for (auto c : char_range) {
        cout << c << " ";
    }
    cout << endl;

    return 0;
}

/*
Number Range: 1 2 3 4 5
Character Range: a b c d e
*/
```

72. Perform Function Overloading

This program demonstrates function overloading in C++. Multiple functions have the same name but differ in parameter types or counts.

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

using namespace std;

// Function with no arguments
void example_function() {
    cout << "No arguments" << endl;
}

// Function with one integer argument
void example_function(int n) {
    cout << "One number argument: " << n << endl;
}

// Function with a string and an integer argument
void example_function(const string& s, int n) {
    cout << "String and number arguments: " << s << ", " <<
n << endl;
}

int main() {
    example_function();                     // Call
function with no arguments
    example_function(42);                   // Call
function with one integer argument
    example_function("Hello", 7);           // Call
function with string and integer arguments

    // Note: Calling example_function("world", 7.5) would
cause a compilation error due to no matching function

    return 0;
}
```

```
/*
No arguments
One number argument: 42
String and number arguments: Hello, 7
*/
```

73. Reverse Image

This program demonstrates how to invert a binary image represented as a 2D array. Each pixel value (0 or 1) is reversed to 1 or 0.

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

using namespace std;

// Function to reverse the image
void reverse_image(vector<vector<int>>& image) {
    for (auto& row : image) {
        for (auto& pixel : row) {
            pixel = 1 - pixel; // Invert pixel
        }
    }
}

// Function to print the image
void print_image(const vector<vector<int>>& image) {
    for (const auto& row : image) {
        for (const auto& pixel : row) {
            cout << pixel << " ";
        }
        cout << endl;
    }
    cout << endl;
}

int main() {
    // Define first image
    vector<vector<int>> image1 = {
        {1, 0, 0},
        {0, 1, 0},
        {0, 0, 1}
    };
    reverse_image(image1);
```

```
    print_image(image1); // Output:
[[0,1,1],[1,0,1],[1,1,0]]

    // Define second image
    vector<vector<int>> image2 = {
        {1, 1, 1},
        {0, 0, 0}
    };
    reverse_image(image2);
    print_image(image2); // Output: [[0,0,0],[1,1,1]]

    // Define third image
    vector<vector<int>> image3 = {
        {1, 0, 0},
        {1, 0, 0}
    };
    reverse_image(image3);
    print_image(image3); // Output: [[0,1,1],[0,1,1]]

    return 0;
}

/*
0 1 1
1 0 1
1 1 0

0 0 0
1 1 1

0 1 1
0 1 1
*/
```

74. No Yelling

This program removes excessive exclamation marks or question marks at the end of a sentence, leaving only a single punctuation mark if needed. It preserves any other punctuation or repeated symbols inside the sentence.

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

using namespace std;

string no_yelling(const string& sentence) {
    if (sentence.empty()) return "";

    size_t end = sentence.size() - 1;

    // Trim excessive '!' or '?' at the end
    while (end > 0 && (sentence[end] == '!' ||
sentence[end] == '?') && sentence[end] == sentence[end-1])
{
        end--;
    }

    string result = sentence.substr(0, end + 1);

    return result;
}

int main() {
    cout << no_yelling("What went wrong?????????") <<
endl;          // → "What went wrong?"
    cout << no_yelling("Oh my goodness!!!") <<
endl;                  // → "Oh my goodness!"
    cout << no_yelling("I just!!! can!!! not!!! believe!!!
it!!!") << endl; // → "I just!!! can!!! not!!! believe!!!
it!"
    cout << no_yelling("Oh my goodness!") <<
endl;                   // → "Oh my goodness!"
```

```
    cout << no_yelling("I just cannot believe it.") <<
endl;          // ➞ "I just cannot believe it."

    return 0;
}

/*
What went wrong?
Oh my goodness!
I just!!! can!!! not!!! believe!!! it!
Oh my goodness!
I just cannot believe it.
*/
```

75. Check if a Number is Float or Integer

This program checks whether a given string represents an integer, a floating-point number, or an invalid number.

```
#include <iostream>
#include <string>
#include <cctype>

using namespace std;

void check_number_type(const string& number) {
    bool is_integer = true;
    bool is_float = false;
    bool dot_seen = false;

    for (size_t i = 0; i < number.size(); i++) {
        if (number[i] == '.') {
            if (dot_seen) {
                is_integer = false;
                break;
            }
            dot_seen = true;
            is_float = true;
        } else if (!isdigit(number[i]) && number[i] != '-'
&& number[i] != '+') {
            is_integer = false;
            is_float = false;
            break;
        }
    }

    if (is_integer && !is_float) {
        cout << number << " is an integer." << endl;
    } else if (is_float) {
        cout << number << " is a float." << endl;
    } else {
        cout << number << " is not a valid number." <<
endl;
    }
```

```
}

int main() {
    check_number_type("5");        // Outputs: 5 is an
integer.
    check_number_type("3.14");     // Outputs: 3.14 is a
float.
    check_number_type("7.0");      // Outputs: 7.0 is a
float.
    check_number_type("-2.5");     // Outputs: -2.5 is a
float.
    check_number_type("abc");      // Outputs: abc is not a
valid number.

    return 0;
}

/*
5 is an integer.
3.14 is a float.
7.0 is a float.
-2.5 is a float.
abc is not a valid number.
*/
```

76. Pass a Function as Parameter

This program demonstrates how to pass a function as a parameter to another function in C++. It allows dynamic execution of different operations on numbers.

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

// Function type using std::function is optional in C++,
// here we'll just use regular function pointers

// Function that takes another function as a parameter
int operate_on_numbers(int a, int b, int (*operation)(int,
int)) {
    return operation(a, b); // Call the passed function
}

// Example functions to be passed
int add(int x, int y) {
    return x + y;
}

int multiply(int x, int y) {
    return x * y;
}

int main() {
    int result1 = operate_on_numbers(3, 5, add);
    cout << "Result of addition: " << result1 <<
endl;       // Outputs: 8

    int result2 = operate_on_numbers(3, 5, multiply);
    cout << "Result of multiplication: " << result2 <<
endl; // Outputs: 15

    return 0;
}

/*
```

```
Result of addition: 8
Result of multiplication: 15
*/
```

77. Slidey Numbers

This program checks whether a number is "slidey," meaning the absolute difference between every two consecutive digits is exactly 1. Single-digit numbers are always considered slidey.

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

// Function to check if a number is slidey
bool is_slidey(int n) {
    string digits = to_string(n); // Convert number to
string
    int len = digits.length();

    if (len <= 1) return true; // Single-digit numbers are
slidey

    for (int i = 0; i < len - 1; i++) {
        if (abs((digits[i] - '0') - (digits[i + 1] - '0'))
!= 1) {
            return false; // Difference is not 1
        }
    }

    return true;
}

int main() {
    // Test cases
    cout << is_slidey(123454321) << endl; // Outputs: 1
(true)
    cout << is_slidey(54345) << endl;     // Outputs: 1
(true)
    cout << is_slidey(987654321) << endl; // Outputs: 1
(true)
    cout << is_slidey(1123) << endl;      // Outputs: 0
(false)
```

```
    cout << is_slidey(1357) << endl;      // Outputs: 0
(false)

    return 0;
}

/*
1
1
1
0
0
*/
```

78. Remove All Whitespaces from a Text

This program removes all whitespace characters (spaces, tabs, newlines) from a given string and outputs a condensed version without any spaces.

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

// Function to remove whitespaces
string remove_whitespaces(const string& input_text) {
    string output_text;
    for (char c : input_text) {
        if (!isspace(c)) { // If not a whitespace
            output_text += c;
        }
    }
    return output_text;
}

int main() {
    string text_with_whitespaces = "This is a text with
spaces";

    // Remove whitespaces
    string text_without_whitespaces =
remove_whitespaces(text_with_whitespaces);

    cout << "Original Text: " << text_with_whitespaces <<
endl;
    cout << "Text without Whitespaces: " <<
text_without_whitespaces << endl;

    return 0;
}

// Original Text: This is a text with spaces
// Text without Whitespaces: Thisisatextwithspaces
```

79. Write to Console

This program demonstrates how to print messages and variables to the console using `std::cout` in C++.

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

int main() {
    // Print a message to the console
    cout << "Hello, world!" << endl;

    // You can also print variables or expressions
    int number = 42;
    cout << "The answer is: " << number << endl;

    // Multiple values can be printed in a single statement
    string first_name = "John";
    string last_name = "Doe";
    cout << "Full Name: " << first_name << " " << last_name
<< endl;

    return 0;
}

/*
Hello, world!
The answer is: 42
Full Name: John Doe
*/
```

80. Convert Date to Number

This program converts the current date and time into the number of milliseconds since the Unix Epoch (January 1, 1970).

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

int main() {
    // Get the current time as a time_point
    auto now = chrono::system_clock::now();

    // Convert to milliseconds since Unix Epoch
    auto milliseconds_since_epoch =
chrono::duration_cast<chrono::milliseconds>(now.time_since_
epoch()).count();

    // Print the current time in milliseconds since Unix
Epoch
    cout << "Current Time in milliseconds since Unix Epoch:
" << milliseconds_since_epoch << endl;

    return 0;
}

/*
Current Time in milliseconds since Unix Epoch:
1726202472472
*/
```

81. Find the Average of Two Numbers

This program calculates the average of two numbers by summing them and dividing by two.

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

// Function to find the average of two numbers
double find_average(double num1, double num2) {
    // Calculate the sum of the two numbers
    double total = num1 + num2;

    // Calculate the average by dividing the sum by 2
    double average = total / 2.0;

    return average;
}

int main() {
    double number1 = 10.0;
    double number2 = 20.0;

    double result = find_average(number1, number2);

    cout << "The average of " << number1 << " and " <<
number2 << " is: " << result << endl;

    return 0;
}

// The average of 10 and 20 is: 15
```

82. Calculate the Area of a Circle

This program calculates the area of a circle given its radius. It checks for valid input before performing the calculation.

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

// Function to calculate the area of a circle
double calculate_circle_area(double radius) {
    // Check if the radius is a valid number
    if (radius <= 0.0) {
        cout << "Invalid radius. Please provide a positive
number." << endl;
        return -1; // Return -1 to indicate an error
    }

    // Calculate the area
    double area = M_PI * radius * radius;
    return area;
}

int main() {
    double radius = 5.0;

    double area = calculate_circle_area(radius);

    // Check for error in area calculation
    if (area >= 0) {
        cout << "The area of a circle with radius " <<
radius << " is: " << area << endl;
    }

    return 0;
}

// The area of a circle with radius 5 is: 78.5398
```

83. Numbered Alphabet

This program converts letters of a string to their corresponding 0-based positions in the alphabet (A=0, B=1, ..., Z=25).

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

// Function to convert letters to their numbered positions
void alph_num(const string &s) {
    for (char c : s) {
        c = toupper(c); // Convert to uppercase
        if (c >= 'A' && c <= 'Z') {
            cout << (c - 'A') << " ";
        }
    }
    cout << endl;
}

int main() {
    string examples[] = {"XYZ", "ABCDEF", "JAVASCRIPT"};

    for (const string &ex : examples) {
        cout << ex << " → ";
        alph_num(ex);
    }

    return 0;
}

/*
XYZ → 23 24 25
ABCDEF → 0 1 2 3 4 5
JAVASCRIPT → 9 0 21 0 18 2 17 8 15 19
*/
```

84. Check if a String is Empty

This program checks whether a given string is empty or not.

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

// Function to check if a string is empty
bool is_empty_string(const string &str) {
    return str.empty(); // Returns true if the string is
empty
}

int main() {
    string empty_string = "";
    string non_empty_string = "Hello, world!";

    cout << "Is empty_string empty? "
         << (is_empty_string(empty_string) ? "true" :
"false") << endl;
    cout << "Is non_empty_string empty? "
         << (is_empty_string(non_empty_string) ? "true" :
"false") << endl;

    return 0;
}

/*
Is empty_string empty? true
Is non_empty_string empty? false
*/
```

85. Capitalize the First Letter of a String

This program capitalizes the first letter of a given string while keeping the rest of the string unchanged.

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

// Function to capitalize the first letter of a string
string capitalize_first_letter(const string &str) {
    if (str.empty()) {
        return "Empty string";
    }

    string result = str;
    result[0] = toupper(result[0]); // Capitalize the first
character
    return result;
}

int main() {
    string original_string = "hello, world!";
    string capitalized_string =
capitalize_first_letter(original_string);

    cout << "Original String: " << original_string << endl;
    cout << "Capitalized String: " << capitalized_string <<
endl;

    return 0;
}

/*
Original String: hello, world!
Capitalized String: Hello, world!
*/
```

86. Find the Maximum Element in an Array

This program finds the largest element in an array of integers. If the array is empty, it returns an indication value.

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

// Function to find the maximum element in an array
int find_max_element(const vector<int> &arr) {
    if (arr.empty()) {
        return -1; // Indicate empty array
    }

    int max_val = numeric_limits<int>::min();
    for (int num : arr) {
        if (num > max_val) {
            max_val = num;
        }
    }

    return max_val;
}

int main() {
    vector<int> numbers = {5, 2, 9, 1, 7};

    int max_number = find_max_element(numbers);

    if (max_number != -1) {
        cout << "Array: ";
        for (int num : numbers) {
            cout << num << " ";
        }
        cout << "\nMaximum Element: " << max_number <<
endl;
```

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

    return 0;
}

/*
Array: 5 2 9 1 7
Maximum Element: 9
*/
```

87. Reverse an Array

This program reverses the elements of an integer array and prints both the original and reversed arrays.

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

// Function to reverse an array
void reverse_array(vector<int> &arr) {
    int start = 0;
    int end = arr.size() - 1;

    while (start < end) {
        swap(arr[start], arr[end]);
        start++;
        end--;
    }
}

int main() {
    vector<int> original_array = {1, 2, 3, 4, 5};
    vector<int> reversed_array = original_array; // Copy
original array

    reverse_array(reversed_array);

    // Print original array
    cout << "Original Array: ";
    for (int num : original_array) {
        cout << num << " ";
    }
    cout << endl;

    // Print reversed array
    cout << "Reversed Array: ";
    for (int num : reversed_array) {
        cout << num << " ";
    }
```

```
    cout << endl;

    return 0;
}

/*
Original Array: 1 2 3 4 5
Reversed Array: 5 4 3 2 1
*/
```

88. Calculate the Power of a Number

This program calculates the power of a number using both the standard `pow()` function for floating-point numbers and a custom loop for integer exponents.

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

// Function to calculate power using pow() for floating-
point numbers
double calculate_power_with_pow(double base, double
exponent) {
    return pow(base, exponent);
}

// Function to calculate power with integer exponent using
a loop
int calculate_power_with_integer_exponent(int base,
unsigned int exponent) {
    int result = 1;
    for (unsigned int i = 0; i < exponent; i++) {
        result *= base;
    }
    return result;
}

int main() {
    double base_f = 2.0;
    double exponent_f = 3.0;
    double result_pow = calculate_power_with_pow(base_f,
exponent_f);
    cout << base_f << " to the power of " << exponent_f <<
" using pow(): " << result_pow << endl;

    int base_i = 2;
    unsigned int exponent_i = 3;
    int result_int_pow =
calculate_power_with_integer_exponent(base_i, exponent_i);
```

```
    cout << base_i << " to the power of " << exponent_i <<
" using custom integer power function: " << result_int_pow
<< endl;

    return 0;
}

/*
2 to the power of 3 using pow(): 8
2 to the power of 3 using custom integer power function: 8
*/
```

89. Find the Minimum Element in an Array

This program finds the minimum element in an array of integers by iterating through the array and keeping track of the smallest value.

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

// Function to find the minimum element in an array
int find_min_element(const vector<int>& arr) {
    if (arr.empty()) {
        return -1; // Indicating an empty array
    }

    int min_val = arr[0];
    for (size_t i = 1; i < arr.size(); i++) {
        if (arr[i] < min_val) {
            min_val = arr[i];
        }
    }
    return min_val;
}

int main() {
    vector<int> numbers = {5, 2, 9, 1, 7};

    if (numbers.empty()) {
        cout << "Empty list" << endl;
    } else {
        int min_number = find_min_element(numbers);
        cout << "List: ";
        for (int num : numbers) {
            cout << num << " ";
        }
        cout << "\nMinimum Element: " << min_number <<
endl;
```

```
    }

    return 0;
}

/*
List: 5 2 9 1 7
Minimum Element: 1
*/
```

90. Convert Minutes to Hours and Minutes

This program converts a total number of minutes into hours and remaining minutes using integer division and modulus operations.

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

// Function to convert total minutes into hours and minutes
void convert_minutes_to_hours_and_minutes(unsigned int
total_minutes, unsigned int &hours, unsigned int &minutes)
{
    hours = total_minutes / 60;   // Calculate full hours
    minutes = total_minutes % 60; // Remaining minutes
}

int main() {
    unsigned int total_minutes = 135;
    unsigned int hours, minutes;

    convert_minutes_to_hours_and_minutes(total_minutes,
hours, minutes);

    cout << total_minutes << " minutes is equivalent to: "
         << hours << " hours and " << minutes << " minutes"
<< endl;

    return 0;
}


// 135 minutes is equivalent to: 2 hours and 15 minutes
```

91. Find the Sum of Digits in a Number

This program calculates the sum of all digits in a given positive integer using a loop and modulus/division operations.

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

// Function to calculate the sum of digits of a number
int sum_of_digits(unsigned int number) {
    int sum = 0;
    while (number > 0) {
        sum += number % 10; // Add the last digit
        number /= 10;       // Remove the last digit
    }
    return sum;
}

int main() {
    unsigned int input_number = 12345;
    int result = sum_of_digits(input_number);

    cout << "The sum of digits in " << input_number << "
is: " << result << endl;

    return 0;
}

// The sum of digits in 12345 is: 15
```

92. Like vs. Dislkes

This program simulates a "Like" and "Dislike" toggle system. Clicking the same button twice resets the state to "Nothing," and switching between "Like" and "Dislike" changes the state accordingly.

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

// Function to determine the final state after button
clicks
string like_or_dislike(const string buttons[], int size) {
    string state = "Nothing";

    for (int i = 0; i < size; i++) {
        if (state == "Nothing" && buttons[i] == "Like") {
            state = "Like";
        } else if (state == "Nothing" && buttons[i] ==
"Dislike") {
            state = "Dislike";
        } else if (state == "Like" && buttons[i] == "Like")
{
            state = "Nothing";
        } else if (state == "Like" && buttons[i] ==
"Dislike") {
            state = "Dislike";
        } else if (state == "Dislike" && buttons[i] ==
"Like") {
            state = "Like";
        } else if (state == "Dislike" && buttons[i] ==
"Dislike") {
            state = "Nothing";
        }
    }

    return state;
}
```

```
int main() {
    string test_cases[][3] = {
        {"Dislike"},
        {"Like", "Like"},
        {"Dislike", "Like"},
        {"Like", "Dislike", "Dislike"}
    };

    int sizes[] = {1, 2, 2, 3};

    for (int i = 0; i < 4; i++) {
        string result = like_or_dislike(test_cases[i],
sizes[i]);
        cout << "Final state: " << result << endl;
    }

    return 0;
}

/*
Final state: Dislike
Final state: Nothing
Final state: Like
Final state: Nothing
*/
```

93. Is a Valid Number?

```c
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <ctype.h>

bool is_valid_phone_number(const char* phone_number) {
    // Check length
    if (strlen(phone_number) != 14) {
        return false;
    }

    // Check format
    for (int i = 0; i < 14; i++) {
        if ((i == 0 && phone_number[i] != '(') ||
            (i >= 1 && i <= 3 && !isdigit(phone_number[i]))
||
            (i == 4 && phone_number[i] != ')') ||
            (i == 5 && phone_number[i] != ' ') ||
            (i >= 6 && i <= 8 && !isdigit(phone_number[i]))
||
            (i == 9 && phone_number[i] != '-') ||
           (i >= 10 && i <= 13 && !isdigit(phone_number[i])))
{
            return false;
        }
    }

    return true;
}

int main() {
    const char* test_numbers[] = {
        "(123) 456-7890",
        "1111)555 2345",
        "098) 123 4567"
    };

    for (int i = 0; i < 3; i++) {
```

```
            printf("Is  '%s'  a  valid  phone  number?  %s\n",
test_numbers[i],
                   is_valid_phone_number(test_numbers[i])  ?
"true" : "false");
    }

    return 0;
}
```

94. Calculate Simple Interest

This program validates a US-style phone number in the format `(XXX) XXX-XXXX`. It ensures the proper placement of parentheses, spaces, and hyphens, and that digits appear where expected.

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

// Function to validate phone number format
bool is_valid_phone_number(const string& phone_number) {
    if (phone_number.length() != 14) return false;

    for (int i = 0; i < 14; i++) {
        if ((i == 0 && phone_number[i] != '(') ||
            (i >= 1 && i <= 3 && !isdigit(phone_number[i]))
||
            (i == 4 && phone_number[i] != ')') ||
            (i == 5 && phone_number[i] != ' ') ||
            (i >= 6 && i <= 8 && !isdigit(phone_number[i]))
||
            (i == 9 && phone_number[i] != '-') ||
            (i >= 10 && i <= 13 &&
!isdigit(phone_number[i]))) {
            return false;
        }
    }

    return true;
}

int main() {
    string test_numbers[] = {
        "(123) 456-7890",
        "1111)555 2345",
        "098) 123 4567"
    };

    for (const auto& num : test_numbers) {
```

```
        cout << "Is '" << num << "' a valid phone number? "
             << (is_valid_phone_number(num) ? "true" :
"false") << endl;
    }

    return 0;
}

/*
Is '(123) 456-7890' a valid phone number? true
Is '1111)555 2345' a valid phone number? false
Is '098) 123 4567' a valid phone number? false
*/
```

95. Mini Sudoku

This program checks if a 3×3 mini Sudoku square is valid. A valid mini Sudoku contains all numbers from 1 to 9 without duplicates.

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

// Function to check if a 3x3 square is a valid mini Sudoku
bool is_mini_sudoku(int square[3][3]) {
    bool seen[10] = {false}; // Track numbers 1-9

    for (int i = 0; i < 3; i++) {
        for (int j = 0; j < 3; j++) {
            int num = square[i][j];

            // Number must be 1-9
            if (num < 1 || num > 9) return false;

            // Check for duplicates
            if (seen[num]) return false;

            seen[num] = true;
        }
    }
    return true;
}

int main() {
    int valid_sudoku[3][3] = {{1, 3, 2}, {9, 7, 8}, {4, 5,
6}};
    int invalid_sudoku_1[3][3] = {{1, 1, 3}, {6, 5, 4}, {8,
7, 9}};
    int invalid_sudoku_2[3][3] = {{0, 1, 2}, {6, 4, 5}, {9,
8, 7}};
    int valid_sudoku_2[3][3] = {{8, 9, 2}, {5, 6, 1}, {3,
7, 4}};

    cout << "is_mini_sudoku(valid_sudoku) = "
```

```
        << (is_mini_sudoku(valid_sudoku) ? "true" :
"false") << endl;
    cout << "is_mini_sudoku(invalid_sudoku_1) = "
         << (is_mini_sudoku(invalid_sudoku_1) ? "true" :
"false") << endl;
    cout << "is_mini_sudoku(invalid_sudoku_2) = "
         << (is_mini_sudoku(invalid_sudoku_2) ? "true" :
"false") << endl;
    cout << "is_mini_sudoku(valid_sudoku_2) = "
         << (is_mini_sudoku(valid_sudoku_2) ? "true" :
"false") << endl;

    return 0;
}

/*
is_mini_sudoku(valid_sudoku) = true
is_mini_sudoku(invalid_sudoku_1) = false
is_mini_sudoku(invalid_sudoku_2) = false
is_mini_sudoku(valid_sudoku_2) = true
*/
```

96. Check if a Number is a Perfect Number

This program checks whether a given number is a perfect number. A perfect number is a positive integer that is equal to the sum of its proper divisors (excluding itself).

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

// Function to check if a number is perfect
bool is_perfect_number(unsigned int number) {
    if (number < 2) return false; // 1 and below are not
perfect numbers

    unsigned int sum_of_divisors = 1; // 1 is always a
proper divisor
    unsigned int sqrt_val = static_cast<unsigned
int>(sqrt(number));

    for (unsigned int i = 2; i <= sqrt_val; i++) {
        if (number % i == 0) {
            sum_of_divisors += i;
            if (i != number / i) {
                sum_of_divisors += number / i;
            }
        }
    }

    return sum_of_divisors == number;
}

int main() {
    unsigned int test_number = 28;

    cout << "Is " << test_number << " a perfect number? "
         << (is_perfect_number(test_number) ? "true" :
"false") << endl;
```

```
    return 0;
}

// Is 28 a perfect number? true
```

97. Calculate the Volume of a Cylinder

This program calculates the volume of a cylinder given its radius and height. The formula used is $\text{Volume} = \pi \times r^2 \times h$

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

// Function to calculate the volume of a cylinder
double calculate_cylinder_volume(double radius, double
height) {
    if (radius <= 0.0 || height <= 0.0) {
        return -1.0; // Invalid input
    }
    const double PI = 3.141592653589793;
    return PI * pow(radius, 2) * height;
}

int main() {
    double cylinder_radius = 5.0;
    double cylinder_height = 10.0;

    double volume =
calculate_cylinder_volume(cylinder_radius,
cylinder_height);

    if (volume < 0) {
        cout << "Invalid inputs. Please provide valid
positive numbers." << endl;
    } else {
        cout << "Cylinder Volume: " << volume << " cubic
units" << endl;
    }

    return 0;
}

// Cylinder Volume: 785.398 cubic units
```

98. Get Student Top Notes

This program finds the highest (top) note for each student from a list of students, where each student can have multiple notes.

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

#define MAX_NOTES 10  // Maximum number of notes per
student

// Define a structure for Student
struct Student {
    int id;
    string name;
    int notes[MAX_NOTES];
    int note_count;  // Number of notes
};

// Function to get the top note for each student
void get_student_top_notes(Student students[], int
student_count) {
    for (int i = 0; i < student_count; i++) {
        int max_note = students[i].notes[0];  // Assume
first note is the max
        for (int j = 1; j < students[i].note_count; j++) {
            if (students[i].notes[j] > max_note) {
                max_note = students[i].notes[j];
            }
        }
        cout << "Top note for " << students[i].name << ": "
<< max_note << endl;
    }
}

int main() {
    // Define some students with their notes
    Student students[] = {
```

```
        {1, "Jacek", {5, 3, 4, 2, 5, 5}, 6},
        {2, "Ewa", {2, 3, 3, 3, 2, 5}, 6},
        {3, "Zygmunt", {2, 2, 4, 4, 3, 3}, 6}
    };
    int student_count = sizeof(students) /
sizeof(students[0]);

    // Get the top notes for each student
    get_student_top_notes(students, student_count);

    return 0;
}

/*
Top note for Jacek: 5
Top note for Ewa: 5
Top note for Zygmunt: 4
*/
```

99. Find the Intersection of Two Arrays

This program finds the common elements (intersection) between two arrays of integers.

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

// Function to find the intersection of two arrays
void find_intersection(int vec1[], int len1, int vec2[],
int len2) {
    cout << "Intersection of Arrays: ";

    // Check each element of vec1 if it exists in vec2
    for (int i = 0; i < len1; i++) {
        for (int j = 0; j < len2; j++) {
            if (vec1[i] == vec2[j]) {
                cout << vec1[i] << " ";
                vec2[j] = -1;  // Mark as visited to avoid
duplicates
                break;          // Stop checking further
elements of vec2
            }
        }
    }
    cout << endl;
}

int main() {
    int vec1[] = {1, 2, 3, 4, 5};
    int vec2[] = {3, 4, 5, 6, 7};
    int len1 = sizeof(vec1) / sizeof(vec1[0]);
    int len2 = sizeof(vec2) / sizeof(vec2[0]);

    find_intersection(vec1, len1, vec2, len2);

    return 0;
}
// Intersection of Arrays: 3 4 5
```

100. Convert Feet to Meters

This program converts a given length in feet to meters using the conversion factor $1 \text{ foot} = 0.3048 \text{ meters}$.

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

// Function to convert feet to meters
double feet_to_meters(double feet) {
    return feet * 0.3048;  // Conversion factor
}

int main() {
    double feet_value = 10.0;
    double meters_value = feet_to_meters(feet_value);

    cout << feet_value << " feet is equal to " <<
meters_value << " meters" << endl;

    return 0;
}

// 10 feet is equal to 3.048 meters
```

101. Convert Days to Years, Months, and Days

This program converts a total number of days into approximate years, months, and days, assuming 1 year = 365 days and 1 month = 30 days.

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

// Function to convert days to years, months, and days
void convert_days_to_years_months_days(int total_days, int
&years, int &months, int &days) {
    years = total_days / 365;
    int remaining_days_after_years = total_days % 365;
    months = remaining_days_after_years / 30;
    days = remaining_days_after_years % 30;
}

int main() {
    int total_days = 1000;
    int years, months, days;

    convert_days_to_years_months_days(total_days, years,
months, days);

    cout << total_days << " days is approximately "
         << years << " years, "
         << months << " months, and "
         << days << " days." << endl;

    return 0;
}

// #include <iostream>
using namespace std;

// Function to convert days to years, months, and days
```

```
void convert_days_to_years_months_days(int total_days, int
&years, int &months, int &days) {
    years = total_days / 365;
    int remaining_days_after_years = total_days % 365;
    months = remaining_days_after_years / 30;
    days = remaining_days_after_years % 30;
}

int main() {
    int total_days = 1000;
    int years, months, days;

    convert_days_to_years_months_days(total_days, years,
months, days);

    cout << total_days << " days is approximately "
         << years << " years, "
         << months << " months, and "
         << days << " days." << endl;

    return 0;
}

// 1000 days is approximately 2 years, 8 months, and 10
days.
```

102. Find the Median of an Array

This program calculates the median of an array of integers. It first sorts the array in ascending order and then finds the middle element if the number of elements is odd, or the average of the two middle elements if the number of elements is even.

```
#include <iostream>
#include <algorithm> // for sort
using namespace std;

// Function to find the median of an array
double findMedian(int arr[], int n) {
    if (n == 0) {
        cout << "Invalid input. Please provide a non-empty
list." << endl;
        exit(1); // Exit the program if the input is
invalid
    }

    // Sort the array
    sort(arr, arr + n);

    // Calculate the median
    if (n % 2 == 0) {
        // Even number of elements → average of two middle
elements
        return (arr[n / 2 - 1] + arr[n / 2]) / 2.0;
    } else {
        // Odd number of elements → middle element
        return arr[n / 2];
    }
}

int main() {
    int numbers[] = {5, 2, 8, 1, 7, 3};
    int n = sizeof(numbers) / sizeof(numbers[0]);

    double median = findMedian(numbers, n);
```

```
    cout << "Median: " << median << endl;

    return 0;
}

// Median: 4
```

103. Calculate the Distance Between Two Points

This program calculates the Euclidean distance between two points (x_1, y_1) and (x_2, y_2) in a 2D plane using the distance formula:

$$\text{distance} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

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

// Function to calculate the distance between two points
(x1, y1) and (x2, y2)
double calculateDistance(double x1, double y1, double x2,
double y2) {
    return sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
}

int main() {
    double x1 = 1.0, y1 = 2.0;
    double x2 = 4.0, y2 = 6.0;

    // Call the function to calculate the distance
    double result = calculateDistance(x1, y1, x2, y2);

    // Print the result
    cout << "The distance between (" << x1 << ", " << y1 <<
") and ("
         << x2 << ", " << y2 << ") is " << result << endl;

    return 0;
}

// The distance between (1.00, 2.00) and (4.00, 6.00) is 5
```

104. Check if a Number is a Perfect Square

This program checks whether a given number is a perfect square. A number is considered a perfect square if its square root is an integer.

$$n \text{ is a perfect square if } \sqrt{n} \in \mathbb{Z}$$

```
#include <iostream>
#include <cmath>

// Function to check if a number is a perfect square
bool is_perfect_square(double number) {
    // Check if the number is non-negative
    if (number < 0) {
        return false; // Negative numbers cannot be perfect
squares
    }

    // Calculate the square root of the number
    double square_root = std::sqrt(number);

    // Check if the square root is an integer
    return square_root == std::floor(square_root);
}

int main() {
    double test_number = 25.0;

    // Call the function to check if the number is a
perfect square
    bool result = is_perfect_square(test_number);

    // Print the result
    std::cout << "Is " << test_number
              << " a perfect square? "
              << (result ? "true" : "false") << std::endl;

    return 0;
```

```
}

// Is 25 a perfect square? true
```

105. Find the Area of a Rectangle

This program calculates the **area of a rectangle** using the formula:

$$\text{Area} = \text{Length} \times \text{Width}$$

It checks if the given inputs are valid positive numbers before performing the calculation.

```
#include <iostream>

// Function to calculate the area of a rectangle
double calculate_rectangle_area(double length, double
width) {
    // Check if the inputs are valid positive numbers
    if (length <= 0.0 || width <= 0.0) {
        std::cout << "Invalid inputs. Please provide valid
positive numbers for length and width." << std::endl;
        return 0.0; // Return 0 if inputs are invalid
    }

    // Calculate the area of the rectangle
    return length * width;
}

int main() {
    double rectangle_length = 5.0;
    double rectangle_width = 8.0;

    // Call the function to calculate the area of the
rectangle
    double result =
calculate_rectangle_area(rectangle_length,
rectangle_width);

    // Print the result
    std::cout << "The area of the rectangle with length "
              << rectangle_length
              << " and width "
```

```
                << rectangle_width
                << " is "
                << result
                << std::endl;

    return 0;
}

// The area of the rectangle with length 5 and width 8 is
40
```

106. Convert Binary to Decimal

This program converts a binary string (composed of '0's and '1's) into its decimal equivalent.

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

// Function to convert binary string to decimal
int binary_to_decimal(const std::string& binary_string) {
    int decimal_value = 0;

    // Check if the input string contains only '0' and '1'
    for (char c : binary_string) {
        if (c != '0' && c != '1') {
            std::cout << "Invalid input. Please provide a
valid binary string." << std::endl;
            return -1; // Return -1 to indicate invalid
input
        }
    }

    // Convert binary string to decimal
    for (char c : binary_string) {
        decimal_value = decimal_value * 2 + (c - '0');
    }

    return decimal_value;
}

int main() {
    std::string binary_number = "1101";  // Example binary
number
    int decimal_result = binary_to_decimal(binary_number);

    // Check if the conversion was successful
    if (decimal_result != -1) {
        std::cout << "The decimal equivalent of binary " <<
binary_number
                  << " is " << decimal_result << std::endl;
```

```
    }

    return 0;
}

// The decimal equivalent of binary 1101 is 13
```

107. Count the Number of Words in a Sentence

This program counts the number of words in a given sentence.

```
#include <iostream>
#include <string>
#include <cctype>

// Function to count words in a sentence
int count_words(const std::string& sentence) {
    if (sentence.empty()) {
        std::cout << "Invalid input. Please provide a valid
sentence." << std::endl;
        return -1; // Invalid input
    }

    int word_count = 0;
    bool in_word = false;

    // Iterate through the sentence to count words
    for (char c : sentence) {
        if (std::isspace(c)) {
            in_word = false; // End of a word
        } else if (!in_word) {
            word_count++;   // Start of a new word
            in_word = true;
        }
    }

    return word_count;
}

int main() {
    std::string sentence = "This is a sample sentence.";
    int word_count = count_words(sentence);

    if (word_count != -1) {
```

```
        std::cout << "The sentence \"" << sentence << "\"
has " << word_count << " words." << std::endl;
    }

    return 0;
}

// The sentence "This is a sample sentence." has 5 words.
```

108. Find the Union of Two Arrays

This program finds the union of two arrays, meaning it combines all unique elements from both arrays into a single result array without duplicates. It first inserts all elements of the first array, then checks elements of the second array and only inserts them if they are not already present.

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

// Function to find the union of two arrays
std::vector<int> find_union(const std::vector<int>& arr1,
const std::vector<int>& arr2) {
    std::vector<int> result;

    // Insert elements of the first array
    for (int num : arr1) {
        result.push_back(num);
    }

    // Insert elements of the second array if not already
present
    for (int num : arr2) {
        bool found = false;
        for (int r : result) {
            if (r == num) {
                found = true;
                break;
            }
        }
        if (!found) {
            result.push_back(num);
        }
    }

    return result;
}
```

```
int main() {
    std::vector<int> array1 = {1, 2, 3, 4, 5};
    std::vector<int> array2 = {3, 4, 5, 6, 7};

    std::vector<int> union_result = find_union(array1,
array2);

    std::cout << "Union of Arrays: ";
    for (int num : union_result) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    return 0;
}

// Union of Arrays: 1 2 3 4 5 6 7
```

109. Scoring System

This program calculates the scores of three players — Andy (A), Ben (B), and Charlotte (C) — based on a string of letters representing points.

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

using namespace std;

// Function to calculate scores for Andy (A), Ben (B), and
Charlotte (C)
void calculate_scores(const string& score_string, int&
andy, int& ben, int& charlotte) {
    // Initialize scores
    andy = 0;
    ben = 0;
    charlotte = 0;

    // Loop through the characters in the score string
    for (char ch : score_string) {
        if (ch == 'A') {
            andy++;
        } else if (ch == 'B') {
            ben++;
        } else if (ch == 'C') {
            charlotte++;
        }
    }
}

int main() {
    int andy, ben, charlotte;

    calculate_scores("A", andy, ben, charlotte);
    cout << "[" << andy << ", " << ben << ", " << charlotte
<< "]" << endl;  // [1, 0, 0]

    calculate_scores("ABC", andy, ben, charlotte);
```

```
    cout << "[" << andy << ", " << ben << ", " << charlotte
<< "]" << endl;  // [1, 1, 1]

    calculate_scores("ABCBACC", andy, ben, charlotte);
    cout << "[" << andy << ", " << ben << ", " << charlotte
<< "]" << endl;  // [2, 2, 3]

    return 0;
}


// [1, 0, 0]
// [1, 1, 1]
// [2, 2, 3]
```

110. Check if a Number is a Strong Number

A Strong Number is a number in which the sum of the factorials of its digits equals the number itself.

This program calculates the factorial of each digit and sums them to check if the number is strong.

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

// Function to calculate the factorial of a number
unsigned int calculate_factorial(unsigned int n) {
    unsigned int result = 1;
    for (unsigned int i = 2; i <= n; i++) {
        result *= i;
    }
    return result;
}

// Function to check if a number is a Strong Number
bool is_strong_number(unsigned int num) {
    unsigned int original_num = num;
    unsigned int sum_of_factorials = 0;

    while (num > 0) {
        unsigned int digit = num % 10;
        sum_of_factorials += calculate_factorial(digit);
        num /= 10;
    }

    return sum_of_factorials == original_num;
}

int main() {
    unsigned int test_number = 145;

    if (is_strong_number(test_number)) {
```

```
        cout << test_number << " is a strong number: true"
<< endl;
    } else {
        cout << test_number << " is a strong number: false"
<< endl;
    }

    return 0;
}

// 145 is a strong number: true
```

111. Check if a Number is a Narcissistic Number

A Narcissistic Number (also called an Armstrong Number) is a number in which the sum of its digits raised to the power of the number of digits equals the number itself.

This program counts the digits, calculates the sum of each digit raised to that power, and checks if it matches the original number.

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

// Function to calculate the number of digits in a number
int count_digits(int num) {
    int count = 0;
    while (num > 0) {
        count++;
        num /= 10;
    }
    return count;
}

// Function to check if a number is a Narcissistic Number
bool is_narcissistic_number(int num) {
    int original_num = num;
    int num_digits = count_digits(num);
    int sum_of_powers = 0;

    while (num > 0) {
        int digit = num % 10;
        sum_of_powers += pow(digit, num_digits);
        num /= 10;
    }

    return sum_of_powers == original_num;
}
```

```
int main() {
    int test_number = 1634;

    if (is_narcissistic_number(test_number)) {
        cout << test_number << " is a Narcissistic Number:
true" << endl;
    } else {
        cout << test_number << " is a Narcissistic Number:
false" << endl;
    }

    return 0;
}

// 1634 is a Narcissistic Number: true
```

112. Count the Number of Consonants in a String

This program counts the number of consonants in a given string. It iterates through each character, converts it to lowercase, checks if it is an alphabet character and not a vowel, and increments the consonant count accordingly.

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

// Function to count the number of consonants in a string
int count_consonants(const string &input_str) {
    int count = 0;

    for (char c : input_str) {
        c = tolower(c);
        if ((c >= 'a' && c <= 'z') && !(c == 'a' || c == 
'e' || c == 'i' || c == 'o' || c == 'u')) {
            count++;
        }
    }

    return count;
}

int main() {
    string test_string = "Hello World";

    int result = count_consonants(test_string);

    cout << "The number of consonants in '" << test_string 
<< "' is: " << result << endl;

    return 0;
}

// The number of consonants in 'Hello World' is: 7
```

113. Check if a Number is a Triangular Number

This program checks if a given number is a triangular number. A triangular number is a number that can be expressed as the sum of the first `n` natural numbers (e.g., 1, 3, 6, 10…).

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

// Function to check if a number is a triangular number
bool is_triangular_number(int num) {
    if (num <= 0) {
        return false; // Zero or negative numbers cannot be
triangular numbers
    }

    int total = 0;
    int n = 1;

    while (total < num) {
        total += n;
        n++;
    }

    return total == num;
}

int main() {
    int test_number = 10;

    bool result = is_triangular_number(test_number);

    if (result) {
        cout << test_number << " is a triangular number:
true" << endl;
    } else {
```

```
        cout << test_number << " is a triangular number:
false" << endl;
    }

    return 0;
}

// 10 is a triangular number: true
```

114. Find the Area of a Trapezoid

This program calculates the area of a trapezoid given the lengths of its two bases and its height.

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

// Function to calculate the area of a trapezoid
double trapezoid_area(double base1, double base2, double
height) {
    // Check if the inputs are valid positive numbers
    if (base1 <= 0.0 || base2 <= 0.0 || height <= 0.0) {
        cout << "Invalid input. Please provide valid
positive numbers." << endl;
        return -1.0; // Return a negative value to indicate
invalid input
    }

    // Calculate the area of the trapezoid
    return 0.5 * height * (base1 + base2);
}

int main() {
    double base1_length = 5.0;
    double base2_length = 9.0;
    double trapezoid_height = 4.0;

    // Calculate the area
    double result = trapezoid_area(base1_length,
base2_length, trapezoid_height);

    if (result >= 0) {
        cout << "The area of the trapezoid is: " << result
<< endl;
    }

    return 0;
} // The area of the trapezoid is: 28
```

115. Abbreviations Unique?

This program checks if a list of abbreviations for a list of words are unique.

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

// Function to check if abbreviations are unique for words
bool unique_abbrev(const vector<string>& abbreviations,
const vector<string>& words) {
    int n = abbreviations.size();

    for (int i = 0; i < n; i++) {
        for (int j = i + 1; j < n; j++) {
            if (abbreviations[i] == abbreviations[j]) {
                return false; // Duplicate abbreviation
found
            }
        }
    }

    // Check if abbreviation matches the beginning of the
corresponding word
    for (int i = 0; i < n; i++) {
        if (words[i].substr(0, abbreviations[i].size()) !=
abbreviations[i]) {
            return false; // Abbreviation does not match
the start of the word
        }
    }

    return true;
}

int main() {
    vector<string> abbreviations1 = {"ho", "h", "ha"};
    vector<string> words1 = {"house", "hope", "happy"};
```

```
    cout << boolalpha << unique_abbrev(abbreviations1,
words1) << endl;  // false

    vector<string> abbreviations2 = {"s", "t", "v"};
    vector<string> words2 = {"stamina", "television",
"vindaloo"};
    cout << boolalpha << unique_abbrev(abbreviations2,
words2) << endl;  // true

    vector<string> abbreviations3 = {"bi", "ba", "bat"};
    vector<string> words3 = {"big", "bard", "battery"};
    cout << boolalpha << unique_abbrev(abbreviations3,
words3) << endl;  // true

    vector<string> abbreviations4 = {"mo", "ma", "me"};
    vector<string> words4 = {"moment", "many", "mean"};
    cout << boolalpha << unique_abbrev(abbreviations4,
words4) << endl;  // true

    return 0;
}

// false
// true
// true
// true
```

116. Check if a Number is a Fibonacci Number

This program checks whether a given number is a Fibonacci number.

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

// Function to check if a number is a perfect square
bool is_perfect_square(unsigned long long n) {
    unsigned long long sqrt_val = static_cast<unsigned long
long>(sqrt(n));
    return (sqrt_val * sqrt_val == n);
}

// Function to check if a number is a Fibonacci number
bool is_fibonacci_number(unsigned long long num) {
    unsigned long long num_squared = num * num;
    return is_perfect_square(5 * num_squared + 4) ||
is_perfect_square(5 * num_squared - 4);
}

int main() {
    unsigned long long test_number = 8;
    bool result = is_fibonacci_number(test_number);

    cout << test_number << " is a Fibonacci number: " <<
boolalpha << result << endl;

    return 0;
}

// 8 is a Fibonacci number: true
```

117. Find the Perimeter of a Rectangle

This program calculates the perimeter of a rectangle given its length and width.

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

// Function to calculate the perimeter of a rectangle
double rectangle_perimeter(double length, double width) {
    if (length <= 0.0 || width <= 0.0) {
        cout << "Invalid input. Please provide valid
positive numbers." << endl;
        return -1;
    }
    return 2.0 * (length + width);
}

int main() {
    double length = 5.0;
    double width = 8.0;

    double perimeter = rectangle_perimeter(length, width);

    if (perimeter != -1) {
        cout << "The perimeter of the rectangle is: " <<
perimeter << endl;
    }

    return 0;
}

// The perimeter of the rectangle is: 26.00
```

118. Club Entry

This program checks if a given word contains consecutive (doubled) letters.

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

// Function to check for doubled letters and return the
result
int club_entry(const string &word) {
    for (size_t i = 0; i < word.length() - 1; i++) {
        if (word[i] == word[i + 1]) {
            int position = word[i] - 'a' + 1;
            return position * 4;
        }
    }
    return -1; // No doubled letter found
}

int main() {
    string words[] = {"hill", "apple", "bee"};

    for (const auto &w : words) {
        int result = club_entry(w);
        if (result != -1) {
            cout << "club_entry(\"" << w << "\") → " <<
result << endl;
        } else {
            cout << "club_entry(\"" << w << "\") → No
doubled letter found" << endl;
        }
    }

    return 0;
}
// club_entry("hill") → 48
// club_entry("apple") → 16
// club_entry("bee") → 8
```

119. Check if a String is Anagram of Another String

This program checks if two given strings are anagrams of each other.

An anagram means that both strings contain the same characters with the same frequencies, regardless of order and case. Non-alphabetic characters are ignored.

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

// Function to clean and count characters
void clean_and_count(const string &str, int count[26]) {
    for (char c : str) {
        c = tolower(c);
        if (isalpha(c)) {  // Only consider alphabetic
characters
            count[c - 'a']++;
        }
    }
}

// Function to check if two strings are anagrams
bool are_anagrams(const string &str1, const string &str2) {
    int count1[26] = {0};  // Count for str1
    int count2[26] = {0};  // Count for str2

    clean_and_count(str1, count1);
    clean_and_count(str2, count2);

    for (int i = 0; i < 26; i++) {
        if (count1[i] != count2[i]) {
            return false;
        }
```

```
    }
    return true;
}

int main() {
    string string1 = "listen";
    string string2 = "silent";

    bool result = are_anagrams(string1, string2);
    cout << "\"" << string1 << "\" and \"" << string2 <<
"\" are anagrams: "
         << (result ? "true" : "false") << endl;

    return 0;
}

// "listen" and "silent" are anagrams: true
```

120. Generate Pascal's Triangle

This program generates Pascal's Triangle up to a specified number of rows.

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

// Function to generate Pascal's Triangle
void generate_pascals_triangle(int num_rows) {
    if (num_rows == 0) return;

    vector<vector<int>> triangle(num_rows);

    for (int i = 0; i < num_rows; i++) {
        triangle[i].resize(i + 1);
        triangle[i][0] = 1;  // First element
        triangle[i][i] = 1;  // Last element

        for (int j = 1; j < i; j++) {
            triangle[i][j] = triangle[i - 1][j - 1] +
triangle[i - 1][j];
        }
    }

    // Print the triangle
    cout << "Pascal's Triangle with " << num_rows << "
rows:\n";
    for (auto &row : triangle) {
        for (int val : row) {
            cout << val << " ";
        }
        cout << endl;
    }
}

int main() {
    int number_of_rows = 5;
```

```
    generate_pascals_triangle(number_of_rows);
    return 0;
}

// Pascal's Triangle with 5 rows:
// 1
// 1 1
// 1 2 1
// 1 3 3 1
// 1 4 6 4 1
```

121. Convert Decimal to Roman Numerals

This program converts a given decimal number (1–3999) into its Roman numeral equivalent.

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

// Function to convert a decimal number to a Roman numeral
void decimal_to_roman(int num) {
    if (num <= 0 || num > 3999) {
        cout << "Invalid input. Please provide a valid
positive integer within the range 1 to 3999." << endl;
        return;
    }

    string roman_symbols[] = {"M", "CM", "D", "CD", "C",
"XC", "L", "XL", "X", "IX", "V", "IV", "I"};
    int values[] = {1000, 900, 500, 400, 100, 90, 50, 40,
10, 9, 5, 4, 1};

    string result = "";
    for (int i = 0; i < 13; i++) {
        while (num >= values[i]) {
            result += roman_symbols[i];
            num -= values[i];
        }
    }

    cout << "The Roman numeral representation is: " <<
result << endl;
}

int main() {
    int decimal_number = 1984;
    decimal_to_roman(decimal_number);
```

```
    return 0;
}

// The Roman numeral representation is: MCMLXXXIV
```

122. Find the Area of a Parallelogram

This program calculates the area of a parallelogram given its base and height.

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

// Function to calculate the area of a parallelogram
double parallelogram_area(double base, double height) {
    if (base > 0.0 && height > 0.0) {
        return base * height;
    } else {
        cout << "Invalid input. Please provide valid
positive numbers." << endl;
        return -1;
    }
}

int main() {
    double base = 6.0;
    double height = 8.0;

    double area = parallelogram_area(base, height);

    if (area != -1) {
        cout << "The area of the parallelogram is: " <<
area << endl;
    }

    return 0;
}

// The area of the parallelogram is: 48.00
```

123. Superheroes

This program filters superhero names that end with "man" from a given list, sorts them alphabetically, and displays the sorted list.

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

// Function to filter and sort superhero names ending with
"man"
void superheroes(vector<string> names) {
    vector<string> filtered;

    // Filter names that end with "man"
    for (auto &name : names) {
        int len = name.length();
        if (len >= 3 && name.substr(len - 3) == "man") {
            filtered.push_back(name);
        }
    }

    // Sort the filtered array alphabetically
    sort(filtered.begin(), filtered.end());

    // Print the sorted filtered names
    cout << "[";
    for (size_t i = 0; i < filtered.size(); i++) {
        cout << "\"" << filtered[i] << "\"";
        if (i < filtered.size() - 1) {
            cout << ", ";
        }
    }
    cout << "]" << endl;
}

int main() {
    // Test case 1
```

```
    vector<string> heroes1 = {"Batman", "Superman",
"Spider-man", "Hulk", "Wolverine", "Wonder-Woman"};
    superheroes(heroes1);

    // Test case 2
    vector<string> heroes2 = {"Catwoman", "Deadpool",
"Dr.Strange", "Captain-America", "Aquaman", "Hawkeye"};
    superheroes(heroes2);

    // Test case 3
    vector<string> heroes3 = {"Wonder-Woman", "Catwoman",
"Invisible-Woman"};
    superheroes(heroes3);

    return 0;
}

// ["Batman", "Spider-man", "Superman"]
// ["Aquaman", "Catwoman"]
// ["Catwoman", "Invisible-Woman", "Wonder-Woman"]
```

124. Applying Discounts

This program applies a given discount percentage to a list of prices.

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

// Function to apply discounts to the prices
void get_discounts(vector<double> prices, const string
&discount) {
    // Extract the discount percentage from the string
    int discount_percentage = stoi(discount);

    // Calculate the discount factor
    double discount_factor = discount_percentage / 100.0;

    // Calculate and print the discounted prices
    cout << "[";
    for (size_t i = 0; i < prices.size(); i++) {
        double discounted_price = prices[i] * (1.0 -
discount_factor);
        cout << fixed << setprecision(2) <<
discounted_price;
        if (i < prices.size() - 1) {
            cout << ", ";
        }
    }
    cout << "]" << endl;
}

int main() {
    // Test case 1
    vector<double> prices1 = {2.0, 4.0, 6.0, 11.0};
    string discount1 = "50";
    get_discounts(prices1, discount1); // [1.00, 2.00,
3.00, 5.50]
```

```
    // Test case 2
    vector<double> prices2 = {10.0, 20.0, 40.0, 80.0};
    string discount2 = "75";
    get_discounts(prices2, discount2); // [2.50, 5.00,
10.00, 20.00]

    // Test case 3
    vector<double> prices3 = {100.0};
    string discount3 = "45";
    get_discounts(prices3, discount3); // [55.00]

    return 0;
}

// [1.00, 2.00, 3.00, 5.50]
// [2.50, 5.00, 10.00, 20.00]
// [55.00]
```

125. Check if a Number is a Smith Number

This program checks whether a given number is a Smith Number.

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

// Function to check if a number is prime
bool is_prime(unsigned long num) {
    if (num < 2) return false;
    for (unsigned long i = 2; i <= sqrt(num); i++) {
        if (num % i == 0) return false;
    }
    return true;
}

// Function to sum the digits of a number
unsigned long sum_of_digits(unsigned long num) {
    unsigned long sum = 0;
    while (num > 0) {
        sum += num % 10;
        num /= 10;
    }
    return sum;
}

// Function to find prime factors of a number
vector<unsigned long> prime_factors(unsigned long num) {
    vector<unsigned long> factors;
    for (unsigned long i = 2; i <= sqrt(num); i++) {
        while (num % i == 0) {
            factors.push_back(i);
            num /= i;
        }
    }
```

```
    if (num > 1) factors.push_back(num);
    return factors;
}

// Function to check if a number is a Smith number
bool is_smith_number(unsigned long num) {
    if (is_prime(num)) return false; // Prime numbers can't
be Smith numbers
    unsigned long original_sum = sum_of_digits(num);
    vector<unsigned long> factors = prime_factors(num);

    unsigned long factor_sum = 0;
    for (auto f : factors) {
        factor_sum += sum_of_digits(f);
    }

    return original_sum == factor_sum;
}

int main() {
    unsigned long number = 728;
    if (is_smith_number(number)) {
        cout << "Is " << number << " a Smith number?
true\n";
    } else {
        cout << "Is " << number << " a Smith number?
false\n";
    }
    return 0;
}

// Is 728 a Smith number? true
```

126. Basic Chessboard

This program generates and prints a basic 8x8 chessboard pattern using text.

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

void generate_chessboard() {
    int size = 8; // Size of the chessboard (8x8)

    for (int row = 0; row < size; row++) {
        for (int col = 0; col < size; col++) {
            // Use 'X' for black squares and ' ' for white
squares
            if ((row + col) % 2 == 0) {
                cout << " ";
            } else {
                cout << "X";
            }
            cout << " "; // Space between squares
        }
        cout << "\n"; // Newline after each row
    }
}
int main() {
    generate_chessboard(); // Generate and print the
chessboard
    return 0;
}
/*
  X   X   X   X
X   X   X   X
  X   X   X   X
X   X   X   X
  X   X   X   X
X   X   X   X
  X   X   X   X
X   X   X   X
*/
```

127. Which Number Is Not Like The Others

This program identifies the unique number in an array, i.e., the number that occurs only once while all others appear multiple times.

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

int unique(const vector<int>& numbers) {
    vector<int> counts(numbers.size(), 0);

    // Count the frequency of each number
    for (size_t i = 0; i < numbers.size(); i++) {
        for (size_t j = 0; j < numbers.size(); j++) {
            if (numbers[i] == numbers[j]) {
                counts[i]++;
            }
        }
    }

    // Find and return the unique number
    for (size_t i = 0; i < numbers.size(); i++) {
        if (counts[i] == 1) {
            return numbers[i];
        }
    }
    return -1; // No unique number found
}

int main() {
    vector<int> numbers1 = {3, 3, 3, 7, 3, 3};
    vector<int> numbers2 = {0, 0, 77, 0, 0};
    vector<int> numbers3 = {0, 1, 1, 1, 1, 1, 1, 1};
```

```
    cout << "Unique number in array 1: " <<
unique(numbers1) << endl;
    cout << "Unique number in array 2: " <<
unique(numbers2) << endl;
    cout << "Unique number in array 3: " <<
unique(numbers3) << endl;

    return 0;
}

// Unique number in array 1: 7
// Unique number in array 2: 77
// Unique number in array 3: 0
```

128. Find the Discount

This program calculates the discounted price of an item.

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

double find_discount(double original_price, unsigned char
discount_percentage) {
    double discount = original_price * (discount_percentage
/ 100.0);
    return original_price - discount;
}

int main() {
    cout << find_discount(1500.0, 50) << endl;  // Output:
750.00
    cout << find_discount(89.0, 20) << endl;    // Output:
71.20
    cout << find_discount(100.0, 75) << endl;   // Output:
25.00

    return 0;
}
```

129. Check if a String is Pangram or Not

This program checks whether a given string is a pangram.

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

bool is_pangram(const string &input_str) {
    int alphabet[26] = {0};

    for (char ch : input_str) {
        ch = tolower(ch);
        if (ch >= 'a' && ch <= 'z') {
            alphabet[ch - 'a'] = 1;
        }
    }

    for (int i = 0; i < 26; i++) {
        if (alphabet[i] == 0) {
            return false;
        }
    }
    return true;
}

int main() {
    string input_string = "The quick brown fox jumps over
the lazy dog";

    if (is_pangram(input_string)) {
        cout << "The given string is a pangram! 🎉" <<
endl;
    } else {
        cout << "The given string is not a pangram. 😳" <<
endl;
    }

    return 0;
} // The given string is a pangram! 🎉
```

130. Coaxial Cable Impedance

This program calculates the characteristic impedance of a coaxial cable using its dimensions and the dielectric constant. The formula used is:

$$Z = \frac{60}{\sqrt{\varepsilon_r}} \cdot \log_{10}\left(\frac{d_d}{d_c}\right)$$

Where:

d_d = outer diameter

d_c = inner conductor diameter

ε_r = relative permittivity of the dielectric

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

double impedance_calculator(double d_d, double d_c, double
e_r) {
    double log_term = log10(d_d / d_c);
    double impedance = 60.0 / sqrt(e_r) * log_term;
    return impedance;
}

int main() {
    cout << fixed;
    cout.precision(1);
    cout << impedance_calculator(20.7, 2.0, 4.0) << endl;
    cout << impedance_calculator(5.3, 1.2, 2.2) << endl;
    cout << impedance_calculator(4.48, 1.33, 2.2) << endl;
    return 0;
}

// 30.4
// 26.1
// 21.3
```

131. Censor Words Longer Than Four Characters

This program censors words in a string that are longer than four characters by replacing them with asterisks (*).

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

void censor(char *text) {
    char result[1024]; // Store the final censored string
    int result_index = 0;
    char word[100]; // Temporary buffer for words
    int word_index = 0;
    int i = 0;

    while (text[i] != '\0') {
        if (isspace(text[i]) || text[i] == '\0') {
            if (word_index > 0) {
                word[word_index] = '\0'; // End the word
                if (strlen(word) > 4) {
                    for (int j = 0; j < strlen(word); j++)
{
                        result[result_index++] = '*';
                    }
                } else {
                    for (int j = 0; j < word_index; j++) {
                        result[result_index++] = word[j];
                    }
                }
                word_index = 0; // Reset the word buffer
            }
            if (text[i] != '\0') {
                result[result_index++] = text[i]; // Add
space or end
            }
        } else {
```

```
            word[word_index++] = text[i]; // Collect the
word
        }
        i++;
    }
    result[result_index] = '\0';
    cout << result << endl;
}

int main() {
    char text1[] = "The code is fourty";
    char text2[] = "Two plus three is five";
    char text3[] = "aaaa aaaaa 1234 12345";

    censor(text1);
    censor(text2);
    censor(text3);

    return 0;
}

/*
The code is ******
Two plus ***** is five
aaaa ***** 1234 *****
*/
```

132. Find the Area of an Ellipse

This program calculates the area of an ellipse using the formula:

$$\text{Area} = \pi \times a \times b$$

where `a` is the semi-major axis and `b` is the semi-minor axis.

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

double calculate_ellipse_area(double semi_major_axis,
double semi_minor_axis) {
    return M_PI * semi_major_axis * semi_minor_axis;
}

int main() {
    double semi_major_axis = 5.0;
    double semi_minor_axis = 3.0;

    double ellipse_area =
calculate_ellipse_area(semi_major_axis, semi_minor_axis);
    cout << "The area of the ellipse is: " << ellipse_area
<< endl;

    return 0;
}

// The area of the ellipse is: 47.1239
```

133. Check if a Number is a Palindrome in Binary

This program checks whether a given number is a palindrome in its binary representation.

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

bool is_binary_palindrome(unsigned int number) {
    string binary = bitset<32>(number).to_string(); //
Convert to 32-bit binary string
    // Remove leading zeros
    size_t first_one = binary.find('1');
    if (first_one != string::npos) {
        binary = binary.substr(first_one);
    }

    string reversed_binary = string(binary.rbegin(),
binary.rend());
    return binary == reversed_binary;
}

int main() {
    unsigned int number_to_check = 9;

    if (is_binary_palindrome(number_to_check)) {
        cout << number_to_check << " is a binary
palindrome! 🎉" << endl;
    } else {
        cout << number_to_check << " is not a binary
palindrome. 😳" << endl;
    }

    return 0;
} // 9 is a binary palindrome! 🎉
```

134. Find the Area of a Rhombus

This program calculates the area of a rhombus given the lengths of its two diagonals.

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

double calculate_rhombus_area(double diagonal1, double
diagonal2) {
    return (diagonal1 * diagonal2) / 2.0;  // Formula: (d1
* d2) / 2
}

int main() {
    double diagonal1_length = 8.0; // Length of the first
diagonal
    double diagonal2_length = 6.0; // Length of the second
diagonal

    double rhombus_area =
calculate_rhombus_area(diagonal1_length, diagonal2_length);

    cout << "The area of the rhombus is: " << rhombus_area
<< endl;

    return 0;
}

// The area of the rhombus is: 24
```

135. Check if a Number is a Catalan Number

This program checks whether a given number is a Catalan number.

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

// Function to calculate the binomial coefficient C(n, k)
unsigned long long binomial_coefficient(unsigned long long
n, unsigned long long k) {
    unsigned long long result = 1;
    if (k > n - k) k = n - k;
    for (unsigned long long i = 0; i < k; i++) {
        result *= n;
        result /= i + 1;
        n -= 1;
    }
    return result;
}

// Function to check if a number is a Catalan number
bool is_catalan_number(unsigned long long num) {
    unsigned long long i = 0;
    unsigned long long catalan;
    while (true) {
        catalan = binomial_coefficient(2 * i, i) / (i + 1);
        if (catalan == num) return true;
        if (catalan > num) return false;
        i++;
    }
}

int main() {
    unsigned long long number_to_check = 42;

    if (is_catalan_number(number_to_check)) {
```

```
        cout << number_to_check << " is a Catalan number! 🎉" << endl;
    } else {
        cout << number_to_check << " is not a Catalan number. 😵" << endl;
    }

    return 0;
}

// 42 is a Catalan number! 🎉
```

136. Find the Luhn Algorithm Check Digit

This program calculates the Luhn check digit for a given partial number.

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

int calculate_luhn_check_digit(unsigned long long input) {
    int total_sum = 0;
    bool is_second_digit = false;

    while (input > 0) {
        int digit = input % 10;
        if (is_second_digit) {
            digit *= 2;
            if (digit > 9) digit -= 9;
        }
        total_sum += digit;
        input /= 10;
        is_second_digit = !is_second_digit;
    }

    int check_digit = (10 - (total_sum % 10)) % 10;
    return check_digit;
}

int main() {
    unsigned long long partial_number = 123456789; //
Example partial number
    int check_digit =
calculate_luhn_check_digit(partial_number);

    cout << "Partial Number: " << partial_number << endl;
    cout << "Check Digit: " << check_digit << endl;
    cout << "Full Number with Check Digit: " <<
partial_number * 10 + check_digit << endl;
```

```
    return 0;
}

// Partial Number: 123456789
// Check Digit: 3
// Full Number with Check Digit: 1234567893
```

137. ATM Pin Validator

This program checks whether a given PIN is valid for an ATM.

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

bool is_valid_pin(const string &pin) {
    int length = pin.length();
    if (length != 4 && length != 6) return false;

    for (char ch : pin) {
        if (!isdigit(ch)) return false;
    }
    return true;
}

int main() {
    string pins[] = {"1234", "12345", "a234", "",
"123456"};
    for (string pin : pins) {
        cout << "PIN '" << pin << "' is valid: "
             << (is_valid_pin(pin) ? "true" : "false") <<
endl;
    }
    return 0;
}

// PIN '1234' is valid: true
// PIN '12345' is valid: false
// PIN 'a234' is valid: false
// PIN '' is valid: false
// PIN '123456' is valid: true
```

138. Check if a Year is a Magic Year

This program determines whether a year is a "Magic Year."

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

bool is_magic_year(int year) {
    string year_str = to_string(year);
    if (year_str.length() != 4) return false; // Ensure 4-
digit year

    int month = stoi(year_str.substr(0, 2));
    int day = stoi(year_str.substr(2, 2));
    int result = stoi(year_str.substr(4, 2));

    return (month * day == result);
}

int main() {
    int year_to_check = 1978; // Example year
    if (is_magic_year(year_to_check)) {
        cout << year_to_check << " is a Magic Year! 🎉" <<
endl;
    } else {
        cout << year_to_check << " is not a Magic Year.
😵" << endl;
    }
    return 0;
}

// 1978 is not a Magic Year. 😵
```

139. Enharmonic Equivalents

This program returns the enharmonic equivalent of a given musical note.

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

std::string getEquivalent(const std::string& note) {
    if (note == "C#") return "Db";
    else if (note == "Db") return "C#";
    else if (note == "D#") return "Eb";
    else if (note == "Eb") return "D#";
    else if (note == "F#") return "Gb";
    else if (note == "Gb") return "F#";
    else if (note == "G#") return "Ab";
    else if (note == "Ab") return "G#";
    else if (note == "A#") return "Bb";
    else if (note == "Bb") return "A#";
    else if (note == "C") return "C";
    else if (note == "D") return "D";
    else if (note == "E") return "E";
    else if (note == "F") return "F";
    else if (note == "G") return "G";
    else if (note == "A") return "A";
    else if (note == "B") return "B";
    return "Invalid note";
}

int main() {
    std::cout << getEquivalent("D#") << std::endl; //
Output: Eb
    std::cout << getEquivalent("Gb") << std::endl; //
Output: F#
    std::cout << getEquivalent("Bb") << std::endl; //
Output: A#
    return 0;
}
```

140. Find the Area of a Regular Polygon

This program calculates the area of a regular polygon (all sides and angles equal) using the formula:

$$\text{Area} = \frac{n \cdot s^2}{4 \cdot \tan(\pi/n)}$$

Where:

- n = number of sides
- s = length of each side

```cpp
#include <iostream>
#include <cmath>

// Function to calculate the area of a regular polygon
double calculateRegularPolygonArea(int n, double s) {
    double numerator = (1.0 / 4.0) * n * pow(s, 2);
    double denominator = tan(M_PI / n);
    double area = numerator / denominator;
    return area;
}

int main() {
    int numberOfSides = 6;    // Number of sides of the
polygon
    double sideLength = 5.0;  // Length of each side

    double polygonArea =
calculateRegularPolygonArea(numberOfSides, sideLength);
    std::cout << "The area of the regular polygon is: " <<
polygonArea << std::endl;

    return 0;
}

// The area of the regular polygon is: 64.95
```

141. Check if a Number is an Abundant Number

This program checks if a given number is an **abundant number**.

```
#include <iostream>

// Function to calculate the sum of proper divisors
int getProperDivisorsSum(int number) {
    int divisorSum = 0;
    int limit = number / 2; // Proper divisors are less
than half the number
    for (int i = 1; i <= limit; i++) {
        if (number % i == 0) {
            divisorSum += i;
        }
    }
    return divisorSum;
}

// Function to check if a number is an abundant number
bool isAbundantNumber(int number) {
    int divisorsSum = getProperDivisorsSum(number);
    return divisorsSum > number; // Return true if it's
abundant
}

int main() {
    int numberToCheck = 12; // Replace with the number you
want to check

    if (isAbundantNumber(numberToCheck)) {
        std::cout << numberToCheck << " is an Abundant
Number! 🎉" << std::endl;
    } else {
        std::cout << numberToCheck << " is not an Abundant
Number. 😳" << std::endl;
    }
```

```
    return 0;
}

// 12 is an Abundant Number! 🎉
```

142. Wash Your Hands

This program calculates the **total time spent washing hands** based on:

- `n` → number of people
- `nm` → number of times each person washes hands
- Each wash takes **21 seconds**.

The program converts the total seconds into **minutes and seconds** for easier reading.

```
#include <iostream>

void washHands(int n, int nm) {
    // Total seconds spent washing hands (21 seconds per
wash)
    int totalSeconds = n * nm * 21;

    // Convert seconds to minutes and remaining seconds
    int minutes = totalSeconds / 60;
    int seconds = totalSeconds % 60;

    std::cout << minutes << " minutes and " << seconds << "
seconds" << std::endl;
}

int main() {
    // Example usage
    washHands(8, 7);  // 8 people, 7 washes each
    washHands(0, 0);  // 0 people, 0 washes
    washHands(7, 9);  // 7 people, 9 washes each

    return 0;
}

// 588 minutes and 36 seconds
// 0 minutes and 0 seconds
// 220 minutes and 33 seconds
```

143. Calculate the Euler's Totient Function

This program calculates **Euler's Totient Function** $\varphi(n$ for a given positive integer `n`.

```
#include <iostream>

unsigned int eulerTotientFunction(unsigned int n) {
    if (n == 0) {
        std::cout << "Input must be a positive integer." <<
std::endl;
        return 0; // Invalid input
    }

    unsigned int result = n;
    unsigned int p = 2;

    // Iterate through all prime factors of n
    while (p * p <= n) {
        if (n % p == 0) {
            while (n % p == 0) {
                n /= p;
            }
            result -= result / p;
        }
        p++;
    }

    // If n is a prime number greater than 1
    if (n > 1) {
        result -= result / n;
    }

    return result;
}

int main() {
```

```
    unsigned int n = 12;  // Number to calculate the
totient function for
    unsigned int result = eulerTotientFunction(n);
    std::cout << "Euler's Totient Function for " << n << "
is: " << result << std::endl;

    return 0;
}

// Euler's Totient Function for 12 is: 4
```

144. Who's The Oldest?

This program determines the **oldest person** in a given group.

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

struct Person {
    std::string name;
    unsigned int age;
};

// Function to find the oldest person
std::string oldest(const std::vector<Person>& people) {
    std::string oldest_name = "";
    unsigned int max_age = 0;

    for (const auto& person : people) {
        if (person.age > max_age) {
            max_age = person.age;
            oldest_name = person.name;
        }
    }

    return oldest_name;
}

int main() {
    // First group of people
    std::vector<Person> people1 = {
        {"Emma", 71},
        {"Jack", 45},
        {"Amy", 15},
        {"Ben", 29}
    };
    std::cout << "The oldest person is: " <<
oldest(people1) << std::endl;

    // Second group of people
```

```
    std::vector<Person> people2 = {
        {"Max", 9},
        {"Josh", 13},
        {"Sam", 48},
        {"Anne", 33}
    };
    std::cout << "The oldest person is: " <<
oldest(people2) << std::endl;

    return 0;
}

// The oldest person is: Emma
// The oldest person is: Sam
```

145. Digital Cipher

This program implements a **digital cipher**, which encodes a message by shifting each letter's value based on a numeric key.

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

std::vector<unsigned int> digital_cipher(const std::string&
message, unsigned int key) {
    std::string key_str = std::to_string(key);
    size_t key_length = key_str.size();
    size_t message_length = message.size();

    std::vector<unsigned int>
encoded_message(message_length);

    for (size_t i = 0; i < message_length; i++) {
        // Letter value: 'a' -> 1, 'b' -> 2, ..., 'z' -> 26
        unsigned int letter_value = message[i] - 'a' + 1;

        // Corresponding digit from the key (repeating if
necessary)
        unsigned int key_digit = key_str[i % key_length] -
'0';

        // Store the encoded value
        encoded_message[i] = letter_value + key_digit;
    }

    return encoded_message;
}

int main() {
    // Example 1
    std::string message1 = "scout";
    unsigned int key1 = 1939;
    auto encoded1 = digital_cipher(message1, key1);
    std::cout << "Encoded message 1: ";
```

```
    for (auto val : encoded1) std::cout << val << " ";
    std::cout << std::endl;

    // Example 2
    std::string message2 = "hernando";
    unsigned int key2 = 1990;
    auto encoded2 = digital_cipher(message2, key2);
    std::cout << "Encoded message 2: ";
    for (auto val : encoded2) std::cout << val << " ";
    std::cout << std::endl;

    // Example 3
    std::string message3 = "abella";
    unsigned int key3 = 100;
    auto encoded3 = digital_cipher(message3, key3);
    std::cout << "Encoded message 3: ";
    for (auto val : encoded3) std::cout << val << " ";
    std::cout << std::endl;

    return 0;
}

// Encoded message 1: 20 12 18 30 21
// Encoded message 2: 9 14 27 14 2 23 13 15
// Encoded message 3: 2 2 5 13 12 1
```

146. Chocolate Dilemma

This program checks whether two sisters, Agatha and Bertha, received **the same total area of chocolate pieces**.

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

// Function to calculate total area of chocolate pieces
int total_area(const std::vector<std::pair<int, int>>&
pieces) {
    int area = 0;
    for (auto& p : pieces) {
        area += p.first * p.second;
    }
    return area;
}

// Function to test fairness between two sisters
bool test_fairness(const std::vector<std::pair<int, int>>&
sister1,
                   const std::vector<std::pair<int, int>>&
sister2) {
    return total_area(sister1) == total_area(sister2);
}

int main() {
    // Example 1
    std::vector<std::pair<int,int>> agatha1 = {{4,3},
{2,4}, {1,2}};
    std::vector<std::pair<int,int>> bertha1 = {{6,2},
{4,2}, {1,1}, {1,1}};
    std::cout << test_fairness(agatha1, bertha1) <<
std::endl;  // 1 (true)

    // Example 2
    std::vector<std::pair<int,int>> agatha2 = {{1,2},
{2,1}};
    std::vector<std::pair<int,int>> bertha2 = {{2,2}};
```

```
    std::cout << test_fairness(agatha2, bertha2) <<
std::endl;  // 1 (true)

    // Example 3
    std::vector<std::pair<int,int>> agatha3 = {{1,2},
{2,1}};
    std::vector<std::pair<int,int>> bertha3 = {{2,2},
{4,4}};
    std::cout << test_fairness(agatha3, bertha3) <<
std::endl;  // 0 (false)

    // Example 4
    std::vector<std::pair<int,int>> agatha4 = {{2,2},
{2,2}, {2,2}, {2,2}};
    std::vector<std::pair<int,int>> bertha4 = {{4,4}};
    std::cout << test_fairness(agatha4, bertha4) <<
std::endl;  // 1 (true)

    // Example 5
    std::vector<std::pair<int,int>> agatha5 = {{1,5},
{6,3}, {1,1}};
    std::vector<std::pair<int,int>> bertha5 = {{7,1},
{2,2}, {1,1}};
    std::cout << test_fairness(agatha5, bertha5) <<
std::endl;  // 0 (false)

    return 0;
}


// 1
// 1
// 0
// 1
// 0
```

147. Calculate the Area of a Hexagon

This program calculates the **area of a regular hexagon** given the length of one of its sides.

```
#include <iostream>
#include <cmath>

// Function to calculate the area of a hexagon
double calculate_hexagon_area(double side_length) {
    return (3.0 * std::sqrt(3.0) / 2.0) *
std::pow(side_length, 2);
}

int main() {
    double side_length = 5.0; // Replace with your
hexagon's side length
    double hexagon_area =
calculate_hexagon_area(side_length);

    std::cout << "The area of the hexagon is: "
              << hexagon_area << std::endl;

    return 0;
}

// The area of the hexagon is: 64.9519
```

148. Check if a Number is a Pronic Number

A **Pronic number** is a number that is the product of two consecutive integers.

```
#include <iostream>

// Function to check if a number is a Pronic number
bool is_pronic_number(int number) {
    for (int i = 0; i <= number; i++) {
        if (i * (i + 1) == number) {
            return true;
        }
    }
    return false;
}

int main() {
    int number1 = 6;  // Example number
    int number2 = 7;  // Another example

    if (is_pronic_number(number1)) {
        std::cout << number1 << " is a Pronic Number!" <<
std::endl;
    } else {
        std::cout << number1 << " is not a Pronic Number."
<< std::endl;
    }

    if (is_pronic_number(number2)) {
        std::cout << number2 << " is a Pronic Number!" <<
std::endl;
    } else {
        std::cout << number2 << " is not a Pronic Number."
<< std::endl;
    }
```

```
    return 0;
}

// 6 is a Pronic Number!
// 7 is not a Pronic Number.
```

149. Virtual DAC

A **Digital-to-Analog Converter (DAC)** converts a digital number into a proportional analog voltage.

```
#include <iostream>

// Function to simulate a virtual DAC
double v_dac(unsigned short digital_value) {
    return (digital_value / 1023.0) * 5.0;
}

int main() {
    std::cout << v_dac(0) << std::endl;      // Expected
output: 0.00
    std::cout << v_dac(1023) << std::endl;   // Expected
output: 5.00
    std::cout << v_dac(400) << std::endl;    // Expected
output: 1.96

    return 0;
}

// 0
// 5
// 1.95695
```

150. Find the Area of a Pentagon

This program calculates the **area of a regular pentagon** (all sides and angles equal) given the length of one side.

```
#include <iostream>
#include <cmath>
#include <iomanip>

// Function to calculate the area of a regular pentagon
double calculate_pentagon_area(double side_length) {
    double area = (1.0 / 4.0) * sqrt(5.0 * (5.0 + 2.0 *
sqrt(5.0))) * side_length * side_length;
    return area;
}

int main() {
    double side_length = 4.0; // Example side length
    double pentagon_area =
calculate_pentagon_area(side_length);

    std::cout << std::fixed << std::setprecision(2);
    std::cout << "The area of the pentagon is: " <<
pentagon_area << std::endl;

    return 0;
}

// The area of the pentagon is: 27.53
```

151. Check if a Number is a Cube Number

This program checks whether a given number is a **perfect cube**.

```
#include <iostream>
#include <cmath>

// Function to check if a number is a cube number
bool is_cube_number(long long number) {
    long long cube_root =
std::round(std::cbrt(static_cast<double>(number)));
    return (cube_root * cube_root * cube_root == number);
}

int main() {
    long long number_to_check = 27; // Example number

    if (is_cube_number(number_to_check)) {
        std::cout << number_to_check << " is a Cube Number!
🎲" << std::endl;
    } else {
        std::cout << number_to_check << " is not a Cube
Number. 😵" << std::endl;
    }

    return 0;
}

// 27 is a Cube Number! 🎲
```

152. Weekly Salary Calculation

This program calculates the **weekly salary** of an employee based on hours worked each day.

```
#include <iostream>

unsigned int weekly_salary(unsigned int hours[7]) {
    unsigned int total_salary = 0;

    for (int i = 0; i < 7; i++) {
        unsigned int hour = hours[i];

        if (i < 5) { // Monday to Friday
            if (hour > 8) {
                total_salary += 8 * 10;           //
Regular hours
                total_salary += (hour - 8) * 15;  //
Overtime
            } else {
                total_salary += hour * 10;
            }
        } else { // Saturday and Sunday
            if (hour > 8) {
                total_salary += 8 * 20;           //
Regular hours
                total_salary += (hour - 8) * 30;  //
Overtime
            } else {
                total_salary += hour * 20;
            }
        }
    }

    return total_salary;
}

int main() {
    unsigned int hours[7] = {8, 8, 8, 8, 8, 0, 0}; //
Example hours
```

```
    unsigned int salary = weekly_salary(hours);
    std::cout << "Total weekly salary: $" << salary <<
std::endl;
    return 0;
}

// Total weekly salary: $400
```

153. Find the Area of a Cube

This program calculates the **surface area of a cube**.

```
#include <iostream>
#include <cmath>

// Function to calculate the surface area of a cube
double calculate_cube_surface_area(double side_length) {
    return 6.0 * std::pow(side_length, 2);
}

int main() {
    double side_length = 4.0; // Example side length
    double cube_surface_area =
calculate_cube_surface_area(side_length);

    std::cout << "The surface area of the cube is: " <<
cube_surface_area << std::endl;
    return 0;
}

// The surface area of the cube is: 96
```

154. Find the Area of a Cone

This program calculates the **surface area of a cone**.

```
#include <iostream>
#include <cmath>

// Function to calculate the surface area of a cone
double calculate_cone_surface_area(double radius, double
slant_height) {
    return M_PI * radius * (radius + slant_height);
}

int main() {
    double radius = 3.0;       // Radius of the cone's base
    double slant_height = 5.0; // Slant height of the cone

    double surface_area =
calculate_cone_surface_area(radius, slant_height);

    std::cout << "The surface area of the cone is: " <<
std::fixed << std::setprecision(2) << surface_area <<
std::endl;

    return 0;
}

// The surface area of the cone is: 75.40
```

155. Check if a Number is a Happy Number

This program checks whether a given number is a **Happy Number**.

```
#include <iostream>
#include <unordered_set>

// Function to calculate the sum of squares of digits
int sum_of_squared_digits(int number) {
    int sum = 0;
    while (number > 0) {
        int digit = number % 10;
        sum += digit * digit;
        number /= 10;
    }
    return sum;
}

// Function to check if a number is happy
bool is_happy_number(int number) {
    int slow = number;
    int fast = number;

    do {
        slow = sum_of_squared_digits(slow);
        fast =
sum_of_squared_digits(sum_of_squared_digits(fast));
    } while (slow != fast);

    return slow == 1;
}

int main() {
    int number = 19; // Number to check

    if (is_happy_number(number)) {
```

```
        std::cout << number << " is a Happy Number! 😀" <<
std::endl;
    } else {
        std::cout << number << " is not a Happy Number.
😵" << std::endl;
    }

    return 0;
}

// 19 is a Happy Number! 😀
```

156. Calculate the Area of a Triangular Prism

This program calculates the surface area of a triangular prism.

```
#include <iostream>
#include <cmath>
#include <iomanip> // for std::setprecision

// Function to calculate the area of a triangle using
Heron's formula
double triangle_area(double a, double b, double c) {
    double s = (a + b + c) / 2.0;  // semi-perimeter
    return std::sqrt(s * (s - a) * (s - b) * (s - c));
}

// Function to calculate the surface area of a triangular
prism
double triangular_prism_area(double a, double b, double c,
double height) {
    double base_area = triangle_area(a, b, c);
    double perimeter = a + b + c;
    return 2 * base_area + perimeter * height;
}

int main() {
    double a = 5.0, b = 6.0, c = 7.0; // sides of the
triangular base
    double height = 4.0;               // height of the
prism

    double surface_area = triangular_prism_area(a, b, c,
height);
    std::cout << "The surface area of the triangular prism
is: "
              << std::fixed << std::setprecision(2) <<
surface_area << std::endl;
    return 0;} // The surface area of the triangular prism
is: 78.00
```

157. Find ASCII Charcode of Inverse Case Character

This program determines the ASCII code of a character with its **inverse case**.

```
#include <iostream>
#include <cctype> // for std::toupper and std::tolower

// Function to find the ASCII code of the inverse case
character
unsigned int counterpart_char_code(char c) {
    if (std::isupper(c)) {
        // Convert uppercase to lowercase
        return static_cast<unsigned int>(std::tolower(c));
    } else if (std::islower(c)) {
        // Convert lowercase to uppercase
        return static_cast<unsigned int>(std::toupper(c));
    } else {
        // Return ASCII code if not a letter
        return static_cast<unsigned int>(c);
    }
}

int main() {
    char test_cases[] = {'A', 'a', '1', '#'};
    int num_cases = sizeof(test_cases) /
sizeof(test_cases[0]);

    for (int i = 0; i < num_cases; i++) {
        char test = test_cases[i];
        std::cout << "Char: " << test
                  << ", ASCII Code: " <<
counterpart_char_code(test)
                  << std::endl;
    }

    return 0;
```

```
}

// Char: A, ASCII Code: 97
// Char: a, ASCII Code: 65
// Char: 1, ASCII Code: 49
// Char: #, ASCII Code: 35
```

158. Solve a Linear Equation

This program solves simple linear equations in one variable of the form: x + a = b or x - a = b

```
#include <iostream>
#include <string>
#include <cstdlib>

// Function to solve a simple linear equation of the form
"x + a = b" or "x - a = b"
int solve_equation(const std::string& equation) {
    size_t equal_pos = equation.find('=');
    if (equal_pos == std::string::npos) {
        std::cerr << "Invalid equation format!\n";
        std::exit(1);
    }

    std::string left_side = equation.substr(0, equal_pos -
1);  // Exclude space before '='
    int rhs = std::stoi(equation.substr(equal_pos +
2));        // Skip "= "

    size_t plus_pos = left_side.find('+');
    size_t minus_pos = left_side.find('-');

    if (plus_pos != std::string::npos) {
        int num = std::stoi(left_side.substr(plus_pos +
2));    // Skip "x + "
        return rhs - num;
    } else if (minus_pos != std::string::npos) {
        int num = std::stoi(left_side.substr(minus_pos +
2));   // Skip "x - "
        return rhs + num;
    } else {
        std::cerr << "Unsupported operation\n";
        std::exit(1);
    }
}
```

```cpp
int main() {
    std::cout << solve_equation("x + 43 = 50") <<
"\n";    // Output: 7
    std::cout << solve_equation("x - 9 = 10") <<
"\n";     // Output: 19
    std::cout << solve_equation("x + 300 = 100") <<
"\n";  // Output: -200

    return 0;
}

// 7
// 19
// -200
```

159. Factorize a Number

This program finds all the factors of a given number and prints them in ascending order.

```
#include <iostream>
#include <vector>
#include <cmath>
#include <algorithm>

// Function to factorize a number and print the factors
void factorize(int n) {
    std::vector<int> factors;

    // Find factors
    for (int i = 1; i <= static_cast<int>(std::sqrt(n));
i++) {
        if (n % i == 0) {
            factors.push_back(i);
            if (i != n / i) {
                factors.push_back(n / i);
            }
        }
    }

    // Sort factors
    std::sort(factors.begin(), factors.end());

    // Print factors
    std::cout << "[";
    for (size_t i = 0; i < factors.size(); i++) {
        std::cout << factors[i];
        if (i < factors.size() - 1) {
            std::cout << ", ";
        }
    }
    std::cout << "]\n";
}
```

```
int main() {
    int num = 12;
    factorize(num);  // Output: [1, 2, 3, 4, 6, 12]

    return 0;
}
```

160. Check if a Number is an Automorphic Number

This program checks whether a number is **Automorphic**. A number is Automorphic if its square ends with the number itself.

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

bool is_automorphic_number(unsigned long long number) {
    unsigned long long square = number * number;

    // Convert numbers to strings
    std::string num_str = std::to_string(number);
    std::string square_str = std::to_string(square);

    // Check if the square ends with the number
    int num_len = num_str.length();
    return square_str.substr(square_str.length() - num_len)
== num_str;
}

int main() {
    unsigned long long number = 25; // Example number

    if (is_automorphic_number(number)) {
        std::cout << number << " is an Automorphic
Number!\n";
    } else {
        std::cout << number << " is not an Automorphic
Number.\n";
    }

    unsigned long long number2 = 7; // Another example
    if (is_automorphic_number(number2)) {
        std::cout << number2 << " is an Automorphic
Number!\n";
    } else {
```

```
        std::cout << number2 << " is not an Automorphic
Number.\n";
    }

    return 0;
}

// 25 is an Automorphic Number!
// 7 is not an Automorphic Number.
```

161. Calculate the Area of a Pyramid

This program calculates the surface area of a square-based pyramid.

```
#include <iostream>
#include <cmath>

// Function to calculate the surface area of a square-based
pyramid
double calculate_pyramid_surface_area(double side_length,
double slant_height) {
    double base_area = std::pow(side_length, 2); // Area of
the square base
    double lateral_area = 4.0 * (side_length *
slant_height) / 2.0; // Lateral area from 4 triangular
faces
    return base_area + lateral_area;
}

int main() {
    double side_length = 4.0;   // Length of the pyramid's
base side
    double slant_height = 5.0;  // Slant height of the
pyramid

    double pyramid_surface_area =
calculate_pyramid_surface_area(side_length, slant_height);
    std::cout << "The surface area of the pyramid is: " <<
pyramid_surface_area << std::endl;

    return 0;
}

// The surface area of the pyramid is: 56
```

162. Check if a Number is a Smith–Morra Gambit Number

This program checks if a number is a **Smith–Morra Gambit Number** (commonly known as a **Smith Number**).

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

// Function to calculate the sum of digits of a number
int digit_sum(int n) {
    int sum = 0;
    while (n > 0) {
        sum += n % 10;
        n /= 10;
    }
    return sum;
}

// Function to perform prime factorization and store
factors in a vector
std::vector<int> prime_factorization(int n) {
    std::vector<int> factors;
    int i = 2;
    while (i * i <= n) {
        while (n % i == 0) {
            factors.push_back(i);
            n /= i;
        }
        i++;
    }
    if (n > 1) {
        factors.push_back(n);
    }
    return factors;
}

// Function to check if a number is a Smith–Morra Gambit
Number
```

```
bool is_smith_morra_gambit_number(int number) {
    if (number <= 1) return false; // Exclude 1 and primes

    std::vector<int> factors = prime_factorization(number);

    // If the number is prime, it's not a Smith number
    if (factors.size() == 1 && factors[0] == number) return
false;

    int sum_of_digits_of_factors = 0;
    for (int factor : factors) {
        sum_of_digits_of_factors += digit_sum(factor);
    }

    return sum_of_digits_of_factors == digit_sum(number);
}

int main() {
    int number = 22; // Example number

    if (is_smith_morra_gambit_number(number)) {
        std::cout << "Is " << number << " a Smith-Morra
Gambit Number? true\n";
    } else {
        std::cout << "Is " << number << " a Smith-Morra
Gambit Number? false\n";
    }

    return 0;
}

// Is 22 a Smith-Morra Gambit Number? true
```

163. Check if a Number is a Solitary Number

A Solitary Number is a number that does not share any common factors with the sum of its proper divisors, other than 1.

```
#include <iostream>

// Function to calculate the sum of proper divisors of a
number
int get_proper_divisors_sum(int number) {
    int sum = 1; // Start with 1, since every number is
divisible by 1

    for (int i = 2; i * i <= number; i++) {
        if (number % i == 0) {
            sum += i;
            if (i != number / i) {
                sum += number / i;
            }
        }
    }

    return sum;
}

// Function to check if a number is a Solitary Number
bool is_solitary_number(int number) {
    int sum_of_divisors = get_proper_divisors_sum(number);
    return number != sum_of_divisors;
}

int main() {
    int solitary_number = 28; // Example number

    if (is_solitary_number(solitary_number)) {
        std::cout << solitary_number << " is a Solitary
Number!\n";
    } else {
```

```
        std::cout << solitary_number << " is not a Solitary
Number.\n";
    }

    return 0;
}

// 28 is a Solitary Number!
```

164. Basic Tower of Hanoi Puzzle

This program recursively prints the steps to solve the puzzle.

```
#include <iostream>

// Function to solve the Tower of Hanoi problem
void tower_of_hanoi(int n, char source, char auxiliary,
char target) {
    if (n == 1) {
        std::cout << "Move disk 1 from " << source << " to
" << target << "\n";
        return;
    }

    // Move n-1 disks from source to auxiliary using target
as temporary
    tower_of_hanoi(n - 1, source, target, auxiliary);

    // Move the nth disk from source to target
    std::cout << "Move disk " << n << " from " << source <<
" to " << target << "\n";

    // Move n-1 disks from auxiliary to target using source
as temporary
    tower_of_hanoi(n - 1, auxiliary, source, target);
}

int main() {
    int number_of_disks = 3;   // Number of disks
    char source_peg = 'A';     // Source peg
    char auxiliary_peg = 'B';  // Auxiliary peg
    char target_peg = 'C';     // Target peg

    std::cout << "Tower of Hanoi solution for " <<
number_of_disks << " disks:\n";
    tower_of_hanoi(number_of_disks, source_peg,
auxiliary_peg, target_peg);

    return 0;
```

```
}

// Tower of Hanoi solution for 3 disks:
// Move disk 1 from A to C
// Move disk 2 from A to B
// Move disk 1 from C to B
// Move disk 3 from A to C
// Move disk 1 from B to A
// Move disk 2 from B to C
// Move disk 1 from A to C
```

165. Calculate the Area of a Frustum

A **frustum** is a portion of a cone that remains after cutting the top off with a plane parallel to the base.

```
#include <iostream>
#include <cmath>

// Function to calculate the surface area of a frustum
double surface_area_of_frustum(double r1, double r2, double
l) {
    double lateral_area = M_PI * (r1 + r2) * l;
    double top_area = M_PI * r1 * r1;
    double bottom_area = M_PI * r2 * r2;
    return lateral_area + top_area + bottom_area;
}

int main() {
    double r1 = 4.0; // Top radius
    double r2 = 8.0; // Bottom radius
    double l = 6.0;  // Slant height

    double area = surface_area_of_frustum(r1, r2, l);
    std::cout << "Surface Area of the frustum is: " << area
<< "\n";

    return 0;
}

// Surface Area of the frustum is: 477.52
```

166. Check if a Number is a Motzkin Number

A **Motzkin number** counts the number of ways of drawing non-intersecting chords between points on a circle, among other combinatorial interpretations.

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

// Function to check if a number is a Motzkin number
bool is_motzkin_number(int number) {
    if (number == 0) return true;

    std::vector<long long> motzkin_numbers(number + 1);
    motzkin_numbers[0] = 1;
    motzkin_numbers[1] = 1;

    for (int n = 2; n <= number; n++) {
        motzkin_numbers[n] = ((2LL * n + 1LL) *
motzkin_numbers[n - 1] +
                              (3LL * n - 3LL) *
motzkin_numbers[n - 2]) / (n + 2LL);
    }

    for (int i = 0; i <= number; i++) {
        if (motzkin_numbers[i] == number) return true;
    }

    return false;
}

int main() {
    int test_numbers[] = {1, 2, 5, 6, 11};
    int n = sizeof(test_numbers) / sizeof(test_numbers[0]);

    for (int i = 0; i < n; i++) {
        if (is_motzkin_number(test_numbers[i])) {
```

```
            std::cout << test_numbers[i] << " is a Motzkin
Number!\n";
        } else {
            std::cout << test_numbers[i] << " is not a
Motzkin Number.\n";
        }
    }

    return 0;
}

// 1 is a Motzkin Number!
// 2 is a Motzkin Number!
// 5 is not a Motzkin Number.
// 6 is not a Motzkin Number.
// 11 is not a Motzkin Number.
```

167. Swapping Two by Two

This program swaps characters in a string **two by two in blocks of four**.

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

// Function to swap two characters at a time in a string
void swap_two(std::string &s) {
    int len = s.length();
    for (int i = 0; i + 3 < len; i += 4) {
        std::swap(s[i], s[i + 2]);
        std::swap(s[i + 1], s[i + 3]);
    }
}

int main() {
    std::string examples[] = {
        "ABCDEFGH",
        "AABBCCDDEEFF",
        "munchkins",
        "FFGGHHI"
    };

    for (auto &str : examples) {
        std::string original = str;
        swap_two(str);
        std::cout << "Original: " << original << ",
Swapped: " << str << std::endl;
    }

    return 0;
}

// Original: ABCDEFGH, Swapped: CDABGHEF
// Original: AABBCCDDEEFF, Swapped: BBAADDCCFFEE
// Original: munchkins, Swapped: ncmuinhks
// Original: FFGGHHI, Swapped: GGFFHHI
```

168. Extract a Word From a Sentence

This program removes a specific word from a sentence.

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

// Function to remove a word from a sentence
std::string remove_word(const std::string& sentence, const
std::string& word_to_remove) {
    std::string result;
    size_t pos = 0, word_len = word_to_remove.length();
    while (pos < sentence.length()) {
        if (sentence.substr(pos, word_len) ==
word_to_remove &&
            (pos + word_len == sentence.length() ||
sentence[pos + word_len] == ' ')) {
            // Skip the word and following space if present
            pos += word_len;
            if (pos < sentence.length() && sentence[pos] ==
' ') pos++;
        } else {
            result += sentence[pos++];
        }
    }
    return result;
}

int main() {
    std::string sentence1 = "One two three four";
    std::string word1 = "two";
    std::cout << remove_word(sentence1, word1) <<
std::endl; // Output: "One three four"

    std::string sentence2 = "Bob has a kid";
    std::string word2 = "kid";
    std::cout << remove_word(sentence2, word2) <<
std::endl; // Output: "Bob has a"

    return 0;
```

```
}

// One three four
// Bob has a
```

169. Basic Chatbot

This program simulates a basic chatbot that responds to user input with predefined messages.

```
#include <iostream>
#include <string>
#include <algorithm>

// Function to convert a string to lowercase
std::string to_lowercase(const std::string& str) {
    std::string lower = str;
    std::transform(lower.begin(), lower.end(),
lower.begin(),
                   [](unsigned char c){ return
std::tolower(c); });
    return lower;
}

// Function to simulate chatbot responses
std::string chatbot(const std::string& message) {
    std::string msg = to_lowercase(message);

    if (msg.find("hello") != std::string::npos) {
        return "Hello! How can I help you?";
    } else if (msg.find("how are you") !=
std::string::npos) {
        return "I am just a computer program, but thanks
for asking!";
    } else if (msg.find("bye") != std::string::npos) {
        return "Goodbye!";
    } else {
        return "I didn't understand that. Can you please
rephrase?";
    }
}

int main() {
    std::string user_input;
```

```
    std::cout << "Chatbot: Hello! How can I help you
today?" << std::endl;

    while (true) {
        std::cout << "You: ";
        std::getline(std::cin, user_input);

        if (user_input.empty()) {
            continue;
        }

        if (to_lowercase(user_input) == "bye") {
            std::cout << "Chatbot: Goodbye!" << std::endl;
            break;
        }

        std::cout << "Chatbot: " << chatbot(user_input) <<
std::endl;
    }

    return 0;
}

// Chatbot: Hello! How can I help you today?
// You: Hello
// Chatbot: Hello! How can I help you?
// You: How are you?
// Chatbot: I am just a computer program, but thanks for
asking!
// You: What's your name?
// Chatbot: I didn't understand that. Can you please
rephrase?
// You: bye
// Chatbot: Goodbye!
```

170. Backspace Attack

This program simulates a "backspace" key inside a string. The character `#` acts like a backspace: when encountered it removes the previous character (if any).

```
#include <iostream>
#include <string>
#include <array>

// Process input where '#' means backspace
std::string erase_backspaces(const std::string &input) {
    std::string output;
    output.reserve(input.size());
    for (char c : input) {
        if (c == '#') {
            if (!output.empty()) output.pop_back();
        } else {
            output.push_back(c);
        }
    }
    return output;
}

int main() {
    std::array<std::string,4> examples = {
        "he##l#hel#llo",
        "major# spar##ks",
        "si###t boy",
        "####"
    };

    for (const auto &ex : examples) {
        std::string out = erase_backspaces(ex);
        std::cout << "Input: '" << ex << "', Output: '" <<
out << "'\n";
    }

    return 0;
}
```

```
// Input: 'he##l#hel#llo', Output: 'hello'
// Input: 'major# spar##ks', Output: 'majo spks'
// Input: 'si###t boy', Output: 't boy'
// Input: '####', Output: ''
```

171. Counter

This program implements a simple counter using a `struct`. It provides functions to increment, decrement, and reset the counter, and prints the counter value after each operation.

```
#include <iostream>

// Counter class
class Counter {
private:
    int count;
public:
    Counter() : count(0) {} // Constructor initializes
count to 0

    void increment() {
        count++;
        std::cout << "Counter: " << count << std::endl;
    }

    void decrement() {
        count--;
        std::cout << "Counter: " << count << std::endl;
    }

    void reset() {
        count = 0;
        std::cout << "Counter reset to 0" << std::endl;
    }
};

int main() {
    Counter counter;  // Initialize counter to 0

    counter.increment();  // Counter: 1
    counter.increment();  // Counter: 2
    counter.decrement();  // Counter: 1
    counter.reset();      // Counter reset to 0
```

```
    return 0;
}

// Counter: 1
// Counter: 2
// Counter: 1
// Counter reset to 0
```

172. Phone Number Word Decoder

This program converts letters in a phone number to their corresponding digits based on a standard phone keypad mapping. Non-letter characters remain unchanged.

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

char char_to_num(char c) {
    if (c >= 'A' && c <= 'C') return '2';
    if (c >= 'D' && c <= 'F') return '3';
    if (c >= 'G' && c <= 'I') return '4';
    if (c >= 'J' && c <= 'L') return '5';
    if (c >= 'M' && c <= 'O') return '6';
    if (c >= 'P' && c <= 'S') return '7';
    if (c >= 'T' && c <= 'V') return '8';
    if (c >= 'W' && c <= 'Z') return '9';
    return c; // Non-letter characters remain unchanged
}

void text_to_num(const std::string& phone) {
    std::string result;
    for (char c : phone) {
        result += char_to_num(c);
    }
    std::cout << result << std::endl;
}

int main() {
    std::string examples[] = {
        "123-647-EYES",
        "(325)444-TEST",
        "653-TRY-THIS",
        "435-224-7613"
    };

    for (const auto& phone : examples) {
        text_to_num(phone);
    }
```

```
    return 0;
}

// 123-647-3937
// (325)444-8378
// 653-879-8447
// 435-224-7613
```

173. Coin Co-Operation

This program simulates a simple iterative “share or steal” coin game between two players.

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

void get_coin_balances(const std::vector<std::string>&
p1_choices,
                       const std::vector<std::string>&
p2_choices,
                       int& p1_balance, int& p2_balance) {
    p1_balance = 3;  // Initial balance for player 1
    p2_balance = 3;  // Initial balance for player 2

    int n = p1_choices.size();
    for (int i = 0; i < n; i++) {
        if (p1_choices[i] == "share" && p2_choices[i] ==
"share") {
            p1_balance += 2;
            p2_balance += 2;
        } else if (p1_choices[i] == "share" &&
p2_choices[i] == "steal") {
            p1_balance -= 1;
            p2_balance += 3;
        } else if (p1_choices[i] == "steal" &&
p2_choices[i] == "share") {
            p1_balance += 3;
            p2_balance -= 1;
        }
        // Both steal → no change
    }
}

int main() {
    std::vector<std::string> p1_choices = {"share",
"share", "share"};
```

```
    std::vector<std::string> p2_choices = {"steal",
"share", "steal"};
    int p1_balance, p2_balance;

    get_coin_balances(p1_choices, p2_choices, p1_balance,
p2_balance);

    std::cout << "Person 1 balance: " << p1_balance <<
std::endl;
    std::cout << "Person 2 balance: " << p2_balance <<
std::endl;

    return 0;
}

// Person 1 balance: 3
// Person 2 balance: 11
```

174. Validate PIN

This program checks whether a given PIN code is valid.

```
#include <iostream>
#include <string>
#include <cctype>

bool validate(const std::string& pin) {
    int len = pin.length();

    // Check if length is either 4 or 6
    if (len != 4 && len != 6) {
        return false; // Invalid length
    }

    // Check if all characters are digits
    for (char c : pin) {
        if (!isdigit(c)) {
            return false; // Contains non-digit characters
        }
    }

    return true; // Valid PIN
}

int main() {
    // Test cases
    std::cout << validate("121317") << std::endl; // 1
(true)
    std::cout << validate("1234") << std::endl;   // 1
(true)
    std::cout << validate("45135") << std::endl;  // 0
(false)
    std::cout << validate("89abc1") << std::endl; // 0
(false)
    std::cout << validate("900876") << std::endl; // 1
(true)
    std::cout << validate(" 4983") << std::endl;  // 0
(false)
```

```
    return 0;
}

// 1
// 1
// 0
// 0
// 1
// 0
```

175. Random Number Generator

This program generates a random number within a specified range.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

int generate_random_number(int min_val, int max_val) {
    return rand() % (max_val - min_val + 1) + min_val;
}

int main() {
    // Seed the random number generator
    std::srand(static_cast<unsigned
int>(std::time(nullptr)));

    // Generate and print a random number between 1 and 10
    int random_number = generate_random_number(1, 10);
    std::cout << "Random Number: " << random_number <<
std::endl;

    return 0;
}

// Random Number: 7
```

176. Seven Boom!

This program checks an array of numbers to see if any of the numbers contain the digit '7'. If any number contains '7', it returns "Boom!"; otherwise, it returns "there is no 7 in the array".

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

std::string seven_boom(const std::vector<int>& numbers) {
    for (int num : numbers) {
        std::string str = std::to_string(num);
        if (str.find('7') != std::string::npos) {
            return "Boom!";
        }
    }
    return "there is no 7 in the array";
}

int main() {
    std::vector<int> example1 = {1, 2, 3, 4, 5, 6, 7};
    std::vector<int> example2 = {8, 6, 33, 100};
    std::vector<int> example3 = {2, 55, 60, 97, 86};

    std::cout << seven_boom(example1) << std::endl;
    std::cout << seven_boom(example2) << std::endl;
    std::cout << seven_boom(example3) << std::endl;

    return 0;
}

// Boom!
// there is no 7 in the array
// Boom!
```

177. Capitalize the Last Letter

This program takes a string and capitalizes the last letter of each word while preserving the rest of the word. It then prints the transformed words.

```
#include <iostream>
#include <sstream>
#include <string>
#include <cctype>

void cap_last(std::string input) {
    std::istringstream iss(input);
    std::string word;

    while (iss >> word) {
        if (!word.empty()) {
            word[word.size() - 1] =
std::toupper(word[word.size() - 1]);
        }
        std::cout << word << " ";
    }
    std::cout << std::endl;
}

int main() {
    std::string example1 = "hello";
    std::string example2 = "My Name Is Example";
    std::string example3 = "HELp THe LASt LETTERs
CAPITALISe";

    cap_last(example1);
    cap_last(example2);
    cap_last(example3);

    return 0;
}
// hellO
// MY NamE IS ExamplE
// HELP THE LAST LETTERS CAPITALISE
```

178. Dice Rolling Simulator

This program simulates rolling a standard six-sided dice by generating a random number between 1 and 6 and displaying the result.

```
#include <iostream>
#include <cstdlib>
#include <ctime>

void roll_dice() {
    int result = std::rand() % 6 + 1; // Generate a random
number between 1 and 6
    std::cout << "You rolled a " << result << std::endl;
}

int main() {
    std::srand(std::time(nullptr)); // Seed the random
number generator

    roll_dice(); // Simulate rolling the dice

    return 0;
}

// You rolled a 4
```

179. Seconds to Time Converter

This program converts a given number of seconds into hours, minutes, and seconds, providing a human-readable format for time duration.

```
#include <iostream>

void convert_seconds_to_time(unsigned int seconds) {
    unsigned int hours = seconds / 3600;
    unsigned int minutes = (seconds % 3600) / 60;
    unsigned int remaining_seconds = seconds % 60;

    std::cout << "Time: " << hours << " hours, "
              << minutes << " minutes, "
              << remaining_seconds << " seconds" <<
std::endl;
}

int main() {
    unsigned int input_seconds = 3665;
    convert_seconds_to_time(input_seconds);

    return 0;
}

// Time: 1 hours, 1 minutes, 5 seconds
```

180. Bar Chart Generator

This program generates a simple text-based bar chart from an array of integers.

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

void generate_bar_chart(const std::vector<int>& data) {
    if (data.empty()) {
        std::cout << "No data to display." << std::endl;
        return;
    }

    // Find the maximum value
    int max_value = data[0];
    for (int val : data) {
        if (val > max_value) {
            max_value = val;
        }
    }

    // Generate the bar chart
    for (int val : data) {
        int bar_length =
static_cast<int>((static_cast<float>(val) / max_value) *
20.0f);
        std::cout << val << " | ";
        for (int j = 0; j < bar_length; j++) {
            std::cout << "";
        }
        for (int j = bar_length; j < 20; j++) {
            std::cout << " ";
        }
        std::cout << std::endl;
    }
}

int main() {
```

```cpp
    std::vector<int> chart_data = {5, 8, 12, 4, 6};
    generate_bar_chart(chart_data);
    return 0;
}

// 5  | ████████
// 8  | ████████████
// 12 | ████████████████████
// 4  | ███████
// 6  | ██████████
```

181. Right-Angled Triangle Pattern

This program prints a right-angled triangle pattern of a specified height using asterisks (*). Each row has a number of asterisks equal to its row number.

```
#include <iostream>

void generate_right_angled_triangle(int height) {
    for (int i = 1; i <= height; i++) {
        for (int j = 1; j <= i; j++) {
            std::cout << "*";
        }
        std::cout << std::endl;
    }
}

int main() {
    int triangle_height = 5; // Example height
    generate_right_angled_triangle(triangle_height);
    return 0;
}

// *
// **
// ***
// ****
// *****
```

182. Positive Count / Negative Sum

This program processes an array of integers and outputs two values: the count of positive numbers and the sum of negative numbers. If the array is empty or contains no positives/negatives, it prints an empty array `[]`.

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

void count_positives_sum_negatives(const std::vector<int>&
numbers) {
    int count_positives = 0;
    int sum_negatives = 0;

    for (int num : numbers) {
        if (num > 0) {
            count_positives++;
        } else if (num < 0) {
            sum_negatives += num;
        }
    }

    if (numbers.empty() || (count_positives == 0 &&
sum_negatives == 0)) {
        std::cout << "[]" << std::endl;
    } else {
        std::cout << "[" << count_positives << ", " <<
sum_negatives << "]" << std::endl;
    }
}

int main() {
    std::vector<int> example1 = {1,2,3,4,5,6,7,8,9,10,-11,-
12,-13,-14,-15};
    std::vector<int> example2 = {92,6,73,-77,81,-90,99,8,-
85,34};
    std::vector<int> example3 = {91,-4,80,-73,-28};
    std::vector<int> example4 = {};
```

```
    count_positives_sum_negatives(example1); // [10, -65]
    count_positives_sum_negatives(example2); // [7, -252]
    count_positives_sum_negatives(example3); // [2, -105]
    count_positives_sum_negatives(example4); // []

    return 0;
}

// [10, -65]
// [7, -252]
// [2, -105]
// []
```

183. Number Pyramid Generator

This program generates a number pyramid of a given height. Each row contains numbers starting from 1 up to the current row number, aligned to form a centered pyramid shape.

```
#include <iostream>

void generate_number_pyramid(int height) {
    for (int i = 1; i <= height; i++) {
        // Print spaces
        for (int j = 0; j < height - i; j++) {
            std::cout << " ";
        }

        // Print numbers
        for (int j = 1; j <= i; j++) {
            std::cout << j;
            if (j < i) {
                std::cout << " ";  // Space between numbers
            }
        }

        std::cout << std::endl; // Move to next line
    }
}

int main() {
    int pyramid_height = 4;
    generate_number_pyramid(pyramid_height);
    return 0;
}

//    1
//   1 2
//  1 2 3
// 1 2 3 4
```

184. Diamond Pattern Generator

This program generates a diamond pattern of a specified odd height. The pattern consists of stars (*) centered with spaces, forming a symmetric diamond shape. The height must be a positive odd integer.

```
#include <iostream>
#include <cmath>

void generate_diamond_pattern(int height) {
    if (height % 2 == 0 || height < 1) {
        std::cout << "Height must be an odd positive
integer." << std::endl;
        return;
    }

    int midpoint = (height + 1) / 2;

    for (int i = 1; i <= height; i++) {
        int spaces = std::abs(midpoint - i); // Number of
spaces
        int stars = height - 2 * spaces;     // Number of
stars

        for (int j = 0; j < spaces; j++) {
            std::cout << " ";
        }

        for (int j = 0; j < stars; j++) {
            std::cout << "*";
        }

        std::cout << std::endl;
    }
}

int main() {
    generate_diamond_pattern(5);
```

```
    return 0;
}
```

185. Check If the Brick Fits through the Hole

This program checks whether a rectangular brick can fit through a rectangular hole.

```
#include <iostream>
#include <algorithm> // for std::min and std::max

bool fits(int brick_w, int brick_h, int hole_w, int hole_h)
{
    return (brick_w <= hole_w && brick_h <= hole_h) ||
(brick_w <= hole_h && brick_h <= hole_w);
}

bool does_brick_fit(int a, int b, int c, int w, int h) {
    return fits(a, b, w, h) || fits(a, c, w, h) || fits(b,
c, w, h);
}

int main() {
    std::cout << does_brick_fit(1, 1, 1, 1, 1) <<
std::endl; // true
    std::cout << does_brick_fit(1, 2, 1, 1, 1) <<
std::endl; // true
    std::cout << does_brick_fit(1, 2, 2, 1, 1) <<
std::endl; // false

    return 0;
}

// 1
// 1
// 0
```

186. Countdown Timer

This program implements a simple countdown timer in seconds.

```
#include <iostream>
#include <thread>   // For std::this_thread::sleep_for
#include <chrono>   // For std::chrono::seconds

int main() {
    unsigned int time;
    std::cout << "Enter countdown time in seconds: ";
    std::cin >> time;

    while (time > 0) {
        std::cout << "Time Remaining: " << time << "
seconds" << std::endl;
        std::this_thread::sleep_for(std::chrono::seconds(1)
); // Sleep for 1 second
        time--;
    }

    std::cout << "Countdown completed!" << std::endl;
    return 0;
}

// Enter countdown time in seconds: 3
// Time Remaining: 3 seconds
// Time Remaining: 2 seconds
// Time Remaining: 1 seconds
// Countdown completed!
```

187. State Names and Abbreviations

This program filters a list of U.S. state inputs to separate full state names from their two-letter abbreviations.

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

bool is_full_state_name(const std::string& state) {
    const std::vector<std::string> full_state_names = {
        "Alabama","Alaska","Arizona","Arkansas","California
","Colorado","Connecticut",
        "Delaware","Florida","Georgia","Hawaii","Idaho","Il
linois","Indiana","Iowa",
        "Kansas","Kentucky","Louisiana","Maine","Maryland",
"Massachusetts","Michigan",
        "Minnesota","Mississippi","Missouri","Montana","Neb
raska","Nevada","New Hampshire",
        "New Jersey","New Mexico","New York","North
Carolina","North Dakota","Ohio","Oklahoma",
        "Oregon","Pennsylvania","Rhode Island","South
Carolina","South Dakota","Tennessee",
        "Texas","Utah","Vermont","Virginia","Washington","W
est Virginia","Wisconsin","Wyoming"
    };

    for (const auto& s : full_state_names) {
        if (state == s) return true;
    }
    return false;
}

void filter_state_names(const std::vector<std::string>&
states, const std::string& category) {
    for (const auto& state : states) {
        if (category == "abb" && state.length() == 2) {
            std::cout << "\"" << state << "\" ";
```

```
        } else if (category == "full" &&
is_full_state_name(state)) {
            std::cout << "\"" << state << "\" ";
        }
    }
    std::cout << std::endl;
}

int main() {
    std::vector<std::string> states1 = {"Arizona", "CA",
"NY", "Nevada"};
    std::vector<std::string> states2 = {"MT", "NJ", "TX",
"ID", "IL"};

    std::cout << "Abbreviations in states1: ";
    filter_state_names(states1, "abb");  // "CA" "NY"

    std::cout << "Full state names in states1: ";
    filter_state_names(states1, "full"); // "Arizona"
"Nevada"

    std::cout << "Abbreviations in states2: ";
    filter_state_names(states2, "abb");  // "MT" "NJ" "TX"
"ID" "IL"

    std::cout << "Full state names in states2: ";
    filter_state_names(states2, "full"); // None

    return 0;
}

// Abbreviations in states1: "CA" "NY"
// Full state names in states1: "Arizona" "Nevada"
// Abbreviations in states2: "MT" "NJ" "TX" "ID" "IL"
// Full state names in states2:
```

188. Functioninator 8000

The "Functioninator 8000" takes a word and transforms it into a "fancy" word by adding a suffix and a numeric length indicator:

```
#include <iostream>
#include <string>
#include <cctype>

// Helper function: check if a character is a consonant
bool isConsonant(char c) {
    c = std::tolower(c);
    return std::isalpha(c) && std::string("aeiou").find(c)
== std::string::npos;
}

// Functioninator 8000 logic
void inatorInator(const std::string &word) {
    char lastChar = word.back(); // Last character of the
word
    std::string suffix = isConsonant(lastChar) ? "inator" :
"-inator";
    std::cout << word << suffix << " " << word.length() <<
"000\n";
}

int main() {
    std::string testCases[] = {"Shrink", "Doom",
"EvilClone"};

    for (const auto &word : testCases) {
        inatorInator(word);
    }

    return 0;
}
// Shrinkinator 6000
// Doominator 4000
// EvilClone-inator 9000
```

189. Pages in a Book

Given a total number of pages in a book, determine if it is possible to number the pages…

```
#include <iostream>
#include <cmath>

// Check if x is a perfect square
bool isPerfectSquare(long long x) {
    long long s = static_cast<long long>(std::sqrt(x));
    return s * s == x;
}

// Check if total pages form a triangular number
bool pagesInBook(long long total) {
    if (total <= 0) return false;

    long long discriminant = 1 + 8 * total;
    if (!isPerfectSquare(discriminant)) return false;

    long long sqrtDisc = static_cast<long
long>(std::sqrt(discriminant));
    long long n = (sqrtDisc - 1) / 2;

    return n > 0 && n * (n + 1) / 2 == total;
}

int main() {
    long long testCases[] = {5, 4005, 9453};

    for (long long total : testCases) {
        std::cout << "Total pages " << total << ": "
                  << (pagesInBook(total) ? "true" :
"false") << "\n";
    }

    return 0;
}
```

```
// Total pages 5: false
// Total pages 4005: true
// Total pages 9453: true
```

190. Highest Digit

Given a positive integer `n`, find the **largest single digit** in that number.

```cpp
#include <iostream>

// Function to find the highest digit in a number
int highestDigit(int n) {
    int maxDigit = 0;

    while (n > 0) {
        int digit = n % 10; // Extract last digit
        if (digit > maxDigit) {
            maxDigit = digit; // Update max digit
        }
        n /= 10; // Remove last digit
    }

    return maxDigit;
}

int main() {
    int testCases[] = {4666, 544, 379, 2, 377401};

    for (int n : testCases) {
        std::cout << "Highest digit in " << n << ": "
                  << highestDigit(n) << std::endl;
    }

    return 0;
}

// Highest digit in 4666: 6
// Highest digit in 544: 5
// Highest digit in 379: 9
// Highest digit in 2: 2
// Highest digit in 377401: 7
```

191. Video Length in Seconds

Convert a video length given in `MM:SS` format (minutes:seconds) into total seconds.

```
#include <iostream>
#include <sstream>
#include <string>

int minutesToSeconds(const std::string& videoLength) {
    std::istringstream iss(videoLength);
    std::string minuteStr, secondStr;

    if (!std::getline(iss, minuteStr, ':') ||
!std::getline(iss, secondStr)) {
        return -1; // Invalid format
    }

    int minutes = std::stoi(minuteStr);
    int seconds = std::stoi(secondStr);

    if (seconds >= 60) {
        return -1; // Invalid seconds
    }

    return minutes * 60 + seconds;
}

int main() {
    std::string testCases[] = {"01:00", "13:56", "10:60",
"121:49", "invalid"};

    for (const auto& video : testCases) {
        std::cout << "Video Length '" << video << "': "
                  << minutesToSeconds(video) << " seconds"
<< std::endl;
    }

    return 0;
```

```
}

// Video Length '01:00': 60 seconds
// Video Length '13:56': 836 seconds
// Video Length '10:60': -1 seconds
// Video Length '121:49': 7309 seconds
// Video Length 'invalid': -1 seconds
```

192. Count Letters in a Word Search

Given a 2D grid of letters, count how many times a specific letter appears.

```
#include <iostream>
#include <array>

size_t letterCounter(const std::array<std::array<char, 6>,
5>& grid, char letter) {
    size_t count = 0;

    for (const auto& row : grid) {
        for (char cell : row) {
            if (cell == letter) {
                count++;
            }
        }
    }

    return count;
}

int main() {
    std::array<std::array<char, 6>, 5> grid = {{
        {'D', 'E', 'Y', 'H', 'A', 'D'},
        {'C', 'B', 'Z', 'Y', 'J', 'K'},
        {'D', 'B', 'C', 'A', 'M', 'N'},
        {'F', 'G', 'G', 'R', 'S', 'R'},
        {'V', 'X', 'H', 'A', 'S', 'S'}
    }};

    char letter = 'D';
    std::cout << "The letter '" << letter << "' appears "
              << letterCounter(grid, letter) << " times."
<< std::endl;

    letter = 'H';
    std::cout << "The letter '" << letter << "' appears "
```

```
              << letterCounter(grid, letter) << " times."
<< std::endl;

    return 0;
}

// The letter 'D' appears 3 times.
// The letter 'H' appears 2 times.
```

193. Find the Other Two Side Lengths

Given the shortest side of a 30°-60°-90° triangle:

- The **longest side (hypotenuse)** is twice the shortest side.
- The **medium side** (opposite 60°) is `shortest * √3`.

```
#include <iostream>
#include <cmath>
#include <iomanip>

const double SQRT_3 = 1.7320508075688772;

void otherSides(double shortest, double& longest, double&
medium) {
    longest = 2.0 * shortest;
    medium = shortest * SQRT_3;

    // Round to 2 decimal places
    longest = std::round(longest * 100.0) / 100.0;
    medium = std::round(medium * 100.0) / 100.0;
}

int main() {
    double testCases[] = {1.0, 12.0, 2.0, 3.0};
    size_t n = sizeof(testCases) / sizeof(testCases[0]);

    for (size_t i = 0; i < n; i++) {
        double shortest = testCases[i];
        double longest, medium;

        otherSides(shortest, longest, medium);

        std::cout << "Shortest Side: " << std::fixed <<
std::setprecision(1) << shortest
                  << ", Longest Side: " << std::fixed <<
std::setprecision(2) << longest
                  << ", Medium Side: " << medium <<
std::endl;
    }
```

```
    return 0;
}

// Shortest Side: 1.0, Longest Side: 2.00, Medium Side:
1.73
// Shortest Side: 12.0, Longest Side: 24.00, Medium Side:
20.78
// Shortest Side: 2.0, Longest Side: 4.00, Medium Side:
3.46
// Shortest Side: 3.0, Longest Side: 6.00, Medium Side:
5.20
```

194. War of Numbers

Given an array of integers:

1. Compute the sum of **even numbers**.
2. Compute the sum of **odd numbers**.
3. Return the **absolute difference** between these sums.

```
#include <iostream>
#include <cstdlib> // for std::abs
#include <vector>

int warOfNumbers(const std::vector<int>& numbers) {
    int evenSum = 0;
    int oddSum = 0;

    for (int num : numbers) {
        if (num % 2 == 0)
            evenSum += num;
        else
            oddSum += num;
    }

    return std::abs(evenSum - oddSum);
}
int main() {
    std::vector<int> example1 = {2, 8, 7, 5};
    std::vector<int> example2 = {12, 90, 75};
    std::vector<int> example3 = {5, 9, 45, 6, 2, 7, 34, 8,
6, 90, 5, 243};

    std::cout << warOfNumbers(example1) << std::endl;
    std::cout << warOfNumbers(example2) << std::endl;
    std::cout << warOfNumbers(example3) << std::endl;

    return 0;
}
// 2
// 27
// 168
```

195. Make a Circle with OOP

Create a `Circle` class that:

1. Stores the **radius** of a circle.
2. Can compute the **area** (`π * r²`).
3. Can compute the **perimeter/circumference** (`2 * π * r`).

Demonstrate it with multiple circle objects.

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

class Circle {
private:
    double radius;

public:
    // Constructor to initialize radius
    Circle(double r) : radius(r) {}

    // Function to calculate area
    double getArea() const {
        return M_PI * radius * radius;
    }

    // Function to calculate perimeter
    double getPerimeter() const {
        return 2 * M_PI * radius;
    }

    // Getter for radius (optional)
    double getRadius() const {
        return radius;
    }
};

int main() {
    double circles[] = {11.0, 4.44};
```

```
    for (double radius : circles) {
        Circle circle(radius);
          cout << "Radius: " << radius << " - Area: " <<
circle.getArea() << ", Perimeter: " << circle.getPerimeter()
<< endl;
    }

    return 0;
}

// Radius: 11.00 - Area: 380.132711, Perimeter: 69.115038
// Radius: 4.44 - Area: 61.932101, Perimeter: 27.897343
```

196. Find the nth Tetrahedral Number

A **tetrahedral number** represents the number of spheres that can form a tetrahedron.

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

// Function to calculate the nth Tetrahedral Number
unsigned long long tetra(unsigned long long n) {
    return n * (n + 1) * (n + 2) / 6;
}

int main() {
    cout << tetra(2) << endl;  // Output: 4
    cout << tetra(5) << endl;  // Output: 35
    cout << tetra(6) << endl;  // Output: 56

    return 0;
}
```

197. Joke Teller

This program randomly selects a joke category and tells a joke to the user.

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <ctime>

using namespace std;

void prompt_user(const string& prompt) {
    cout << prompt << " ";
    string input;
    getline(cin, input);  // Waits for user input
}

void joke_teller_program() {
      vector<string> categories = {"knock-knock", "dad",
"animal", "puns"};

    // Randomly select a category
    srand(time(0));  // Seed for random number generation
          string category = categories[rand() %
categories.size()];

    string setup, punchline1, punchline2;

    if (category == "knock-knock") {
        setup = "Knock, knock.";
        punchline1 = "Tank.";
        punchline2 = "You're welcome!";
    } else if (category == "dad") {
        setup = "Why did the scarecrow win an award?";
        punchline1 = "";
          punchline2 = "Because he was outstanding in his
field!";
    } else if (category == "animal") {
```

```
        setup = "Why don't scientists trust atoms?";
        punchline1 = "";
        punchline2 = "Because they make up everything!";
    } else {  // Default to "puns"
         setup = "I used to be a baker because I kneaded
dough.";
        punchline1 = "";
        punchline2 = "";
    }

    if (!setup.empty()) cout << setup << endl;
    if (!punchline1.empty()) prompt_user(punchline1);
    cout << punchline2 << endl;
}

int main() {
    joke_teller_program();
    return 0;
}


// Why did the scarecrow win an award?
// Because he was outstanding in his field!
```

198. Which Generation Are You?

This program determines a person's family relationship based on:

1. **Generation number (`x`):**
   - Negative numbers indicate ancestors.
   - Positive numbers indicate descendants.
   - Zero indicates yourself.
2. **Gender (`y`):**
   - `'m'` for male.
   - `'f'` for female.

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

using namespace std;

string generation(int x, char y) {
    switch (x) {
        case -3:
            return (y == 'm') ? "great grandfather" : "great
grandmother";
        case -2:
           return (y == 'm') ? "grandfather" : "grandmother";
        case -1:
            return (y == 'm') ? "father" : "mother";
        case 0:
            return "me!";
        case 1:
            return (y == 'm') ? "son" : "daughter";
        case 2:
           return (y == 'm') ? "grandson" : "granddaughter";
        case 3:
             return (y == 'm') ? "great grandson" : "great
granddaughter";
        default:
            return "unknown";
    }
```

```
}

int main() {
    cout << generation(2, 'f') << endl;  // granddaughter
    cout << generation(-3, 'm') << endl; // great grandfather
    cout << generation(1, 'f') << endl;  // daughter
    return 0;
}

// granddaughter
// great grandfather
// daughter
```

199. FizzBuzz Game

The **FizzBuzz Game** is a classic programming exercise:

1. Count from **1 to 100**.
2. For each number:
   - If divisible by **3**, print `"Fizz"`.
   - If divisible by **5**, print `"Buzz"`.
   - If divisible by **both 3 and 5**, print `"FizzBuzz"`.
   - Otherwise, print the number itself.

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

using namespace std;

void fizz_buzz_game() {
    cout << "Welcome to the FizzBuzz Game!" << endl;
    cout << "Let's play FizzBuzz!" << endl;

    for (int i = 1; i <= 100; ++i) {
        string output = "";

        if (i % 3 == 0) {
            output += "Fizz";
        }

        if (i % 5 == 0) {
            output += "Buzz";
        }

        // Print the output or the number itself
        if (output.empty()) {
            cout << i << endl;
        } else {
            cout << output << endl;
        }
    }
}
```

```c
int main() {
    fizz_buzz_game();
    return 0;
}

// Welcome to the FizzBuzz Game!
// Let's play FizzBuzz!
// 1
// 2
// Fizz
// 4
// Buzz
// Fizz
```

200. Swap Pairs of Adjacent Digits

This program swaps every pair of adjacent digits in an integer **with an even number of digits**:

1. Convert the integer to a string for easy manipulation.
2. Loop through the string two characters at a time.
3. Swap each pair of adjacent digits.
4. Convert the result back to an integer and display it.

```
#include <iostream>
#include <string>
#include <algorithm>

using namespace std;

void swap_pairs_of_adjacent_digits(unsigned int number) {
    string number_str = to_string(number);
    int length = number_str.length();

    // Check if the number has an even length
    if (length % 2 != 0) {
       cout << "Please enter an integer with an even length."
<< endl;
        return;
    }

    // Swap pairs of adjacent digits
    string swapped_number = "";
    for (int i = 0; i < length; i += 2) {
        // Swap adjacent digits
        swapped_number += number_str[i + 1];
        swapped_number += number_str[i];
    }

    // Convert the result back to an integer
    unsigned int result = stoul(swapped_number);
    cout << "Original Number: " << number << endl;
    cout << "Number with Swapped Pairs of Adjacent Digits:
" << result << endl;
```

```
}

int main() {
    unsigned int number = 123456;
    swap_pairs_of_adjacent_digits(number);
    return 0;
}


// Original Number: 123456
// Number with Swapped Pairs of Adjacent Digits: 214365
```

201. Capitalization Changer

This program **changes the capitalization** of each character in a string:

```
#include <iostream>
#include <string>
#include <cctype>

using namespace std;

string change_capitalization(const string &input_string) {
    string result_string = "";

    for (char ch : input_string) {
        if (isupper(ch)) {
            // Convert uppercase to lowercase
            result_string += tolower(ch);
        } else {
            // Convert lowercase to uppercase
            result_string += toupper(ch);
        }
    }

    cout << "Original String: " << input_string << endl;
     cout << "String with Changed Capitalization: " <<
result_string << endl;

    return result_string;
}

int main() {
    string input = "Hello World";
    change_capitalization(input);
    return 0;
}

// Original String: Hello World
// String with Changed Capitalization: hELLO wORLD
```

202. Array Halves Swapper

This program swaps the two halves of an array:

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

void swap_array_halves(vector<int>& arr) {
    int length = arr.size();

    // Check if the array has an even length
    if (length % 2 != 0) {
       cout << "Please provide an array with an even length."
<< endl;
        return;
    }

    // Calculate the midpoint of the array
    int midpoint = length / 2;

    // Swap the two halves of the array
    for (int i = 0; i < midpoint; ++i) {
        swap(arr[i], arr[midpoint + i]);
    }

    cout << "Array with Swapped Halves: ";
    for (int num : arr) {
        cout << num << " ";
    }
    cout << endl;
}

int main() {
    vector<int> array = {1, 2, 3, 4, 5, 6};
    swap_array_halves(array);
    return 0;
}

// Array with Swapped Halves: [4, 5, 6, 1, 2, 3]
```

203. Sum of Digits in String

This program calculates the **sum of all digits** in a given string:

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

void sum_of_digits_in_string(const string& input_string) {
    int digit_sum = 0;

    for (char ch : input_string) {
        // Check if the character is a digit
        if (isdigit(ch)) {
            // Add the digit value to the sum
            digit_sum += ch - '0';
        }
    }

    cout << "Original String: " << input_string << endl;
    cout << "Sum of Digits in the String: " << digit_sum <<
endl;
}

int main() {
    sum_of_digits_in_string("abc123xyz456");
    return 0;
}

// Original String: abc123xyz456
// Sum of Digits in the String: 21
```

204. Sum of Cubes

This program calculates the **sum of cubes** of all integers from 1 up to a given number:

- It iterates from `1` to `n`.
- For each integer, it calculates its cube (i^3) and adds it to a running total.
- Uses `unsigned long long` to handle large sums safely.
- Finally, it prints the total sum.

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

void sum_of_cubes(unsigned int up_to_integer) {
    unsigned long long cubes_sum = 0;  // Use long long to
handle large sums

    for (unsigned int i = 1; i <= up_to_integer; ++i) {
        // Calculate the cube of each integer and add to sum
        cubes_sum += static_cast<unsigned long long>(i) * i
* i;
    }

    cout << "Sum of Cubes from 1 to " << up_to_integer << ":
" << cubes_sum << endl;
}

int main() {
    sum_of_cubes(5);
    return 0;
}

// Sum of Cubes from 1 to 5: 225
```

205. Maximum Integer for Sum

This program finds the **largest integer `n`** such that the sum of all integers from 1 to `n` does **not exceed a given target sum**.

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

void find_max_integer_for_sum(unsigned int target_sum) {
    unsigned int current_sum = 0;
    unsigned int max_integer = 0;

    while (current_sum + max_integer + 1 <= target_sum) {
        max_integer += 1;
        current_sum += max_integer;
    }

    cout << "Maximum Integer (n) for Sum <= " << target_sum
<< ": " << max_integer << endl;
}

int main() {
    find_max_integer_for_sum(15);
    return 0;
}

// Maximum Integer (n) for Sum <= 15: 5
```

206. URL Breakdown

This program parses a URL into its **scheme**, **domain**, and **path** using **C++ regex**:

1. **Scheme** – e.g., `http`, `https`, `ftp`.
2. **Domain** – e.g., `www.example.com`.
3. **Path** – e.g., `/page/subpage`.

```
#include <iostream>
#include <regex>
#include <string>
#include <unordered_map>

void break_url(const std::string& url) {
    // Regular expression to capture scheme, domain, and path
              std::regex      url_regex(R"(^(\w+):\/\/([\w.-
]+)(\/.*)?$)");
    std::smatch matches;

    if (std::regex_match(url, matches, url_regex)) {
        // Create a map to store the parts of the URL
              std::unordered_map<std::string,   std::string>
url_parts;

        url_parts["scheme"] = matches[1].str();
        url_parts["domain"] = matches[2].str();
        url_parts["path"] = matches[3].str().empty() ? "" :
matches[3].str();

        // Output the parts of the URL
        std::cout << "URL Parts:\n";
        for (const auto& pair : url_parts) {
             std::cout << pair.first << ": " << pair.second
<< std::endl;
        }
    } else {
        std::cout << "Invalid URL format." << std::endl;
    }
```

```
}

int main() {
    break_url("https://www.example.org/page");
    return 0;
}


// URL Parts:
// path: /page
// scheme: https
// domain: www.example.org
```

207. Sort Strings by Length

This program sorts an array of strings in ascending order based on the **length of each string**.

```
#include <iostream>
#include <vector>
#include <algorithm>

void       sort_strings_by_length(std::vector<std::string>&
strings) {
    // Sort the vector of strings by length
       std::sort(strings.begin(),  strings.end(),  [](const
std::string& a, const std::string& b) {
        return a.size() < b.size();
    });

    // Print the original array and the sorted array
    std::cout << "Original Array of Strings:\n";
    for (const auto& str : strings) {
        std::cout << str << " ";
    }
    std::cout << "\n";

    std::cout << "Array of Strings Sorted by Length:\n";
    for (const auto& str : strings) {
        std::cout << str << " ";
    }
    std::cout << "\n";
}

int main() {
     std::vector<std::string> strings = {"apple", "banana",
"orange", "kiwi", "grape"};
    sort_strings_by_length(strings);
    return 0;
}


// Original Array of Strings:
```

```
// ["apple", "banana", "orange", "kiwi", "grape"]
// Array of Strings Sorted by Length:
// ["kiwi", "apple", "grape", "banana", "orange"]
```

208. Simplify Absolute Path

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

void simplify_absolute_path(const std::string& path) {
    std::vector<std::string> parts;
    std::stringstream ss(path);
    std::string part;

    // Split the path into parts using '/' as a delimiter
    while (std::getline(ss, part, '/')) {
        if (part == "..") {
            if (!parts.empty()) {
                  parts.pop_back(); // Go up one directory
level
            }
        } else if (part != "." && !part.empty()) {
            parts.push_back(part); // Add valid part
        }
    }

    // Construct the simplified path
    std::string simplified_path = "/";
    for (size_t i = 0; i < parts.size(); ++i) {
        simplified_path += parts[i];
        if (i != parts.size() - 1) {
            simplified_path += "/";
        }
    }

     std::cout << "Original Absolute Path: " << path <<
std::endl;
       std::cout << "Simplified Absolute Path: " <<
simplified_path << std::endl;
}

int main() {
```

```
    simplify_absolute_path("/home/user/../documents/./file.
txt");
    return 0;
}

// Original Absolute Path:
/home/user/../documents/./file.txt
// Simplified Absolute Path: /home/documents/file.txt
// Original Absolute Path: /a/./b/../../c/
// Simplified Absolute Path: /c
// Original Absolute Path: /../
// Simplified Absolute Path: /
```

209. Count Common Elements in Arrays

This program takes **two arrays of integers** and counts the number of **distinct elements that are common** to both arrays.

```
#include <iostream>
#include <vector>
#include <unordered_set>

void count_common_elements(const std::vector<int>& arr1,
const std::vector<int>& arr2) {
    std::unordered_set<int> set1(arr1.begin(), arr1.end());
    std::unordered_set<int> set2(arr2.begin(), arr2.end());

    // Find common elements by using the intersection of the
two sets
    std::unordered_set<int> common_elements;
    for (const int& elem : set1) {
        if (set2.find(elem) != set2.end()) {
            common_elements.insert(elem);
        }
    }

    // Display results
    std::cout << "Array 1: ";
    for (const int& num : arr1) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    std::cout << "Array 2: ";
    for (const int& num : arr2) {
        std::cout << num << " ";
    }
    std::cout << std::endl;

    std::cout << "Common Elements: ";
    for (const int& num : common_elements) {
        std::cout << num << " ";
    }
```

```
    std::cout << std::endl;

      std::cout << "Number of Common Elements: " <<
common_elements.size() << std::endl;
}

int main() {
    count_common_elements({1, 2, 3, 4, 5}, {3, 4, 5, 6, 7});
    return 0;
}


// Array 1: [1, 2, 3, 4, 5]
// Array 2: [3, 4, 5, 6, 7]
// Common Elements: [5, 3, 4]
// Number of Common Elements: 3
```

210. Check Same Digits in a Number

This program checks whether all digits in a given integer are the same.

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

void are_all_digits_same(int number) {
    int original_number = number;  // Keep the original
number
    int first_digit = number % 10;
    number /= 10;

    while (number > 0) {
        int digit = number % 10;
        if (digit != first_digit) {
            cout << "Digits in " << original_number << "
are not all the same." << endl;
            return;
        }
        number /= 10;
    }

    cout << "Digits in " << original_number << " are all
the same." << endl;
}

int main() {
    are_all_digits_same(22222);
    are_all_digits_same(12345);
    are_all_digits_same(7777);
    return 0;
}

// Digits in 22222 are all the same.
// Digits in 12345 are not all the same.
// Digits in 7777 are all the same.
```

211. Rightmost Round Number Position

This program finds the **rightmost "round number"** in an array (a number divisible by 10) and prints its **position** (1-based index). If no round numbers exist, it prints position 0.

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

void rightmost_round_number_position(const vector<int>&
arr) {
    // Loop through the array from the end to the beginning
    for (int i = arr.size() - 1; i >= 0; i--) {
        if (arr[i] % 10 == 0) {
            cout << "Rightmost Round Number: " << arr[i]
                 << ", Position: " << (i + 1) << endl;
            return;
        }
    }
    cout << "No round numbers found in the array. Position:
0" << endl;
}

int main() {
    vector<int> arr = {123, 450, 678, 900};

    rightmost_round_number_position(arr);

    return 0;
}

// Rightmost Round Number: 900, Position: 4
```

212. Reverse Bits of 16-Bit Unsigned Short Integer

This program reverses the **bits** of a 16-bit unsigned short integer. It also prints both the **original and reversed binary representations** and the corresponding decimal values.

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

// Function to reverse bits of a 16-bit unsigned short
unsigned short reverse_bits_16_bit_unsigned_short(unsigned
short integer) {
    unsigned short reversed_integer = 0;

    for (int i = 0; i < 16; i++) {
        unsigned short bit = (integer >> i) &
1;           // Get i-th bit
        reversed_integer |= (bit << (15 -
i));            // Place it in reversed position
    }

    return reversed_integer;
}

int main() {
    unsigned short num = 5678;

    cout << "Original Integer: " << num << endl;
    cout << "Binary Representation: " << bitset<16>(num) <<
endl;

    unsigned short reversed_num =
reverse_bits_16_bit_unsigned_short(num);

    cout << "Reversed Binary: " << bitset<16>(reversed_num)
<< endl;
    cout << "Reversed Integer: " << reversed_num << endl;
```

```
    return 0;
}

// Original Integer: 5678
// Binary Representation: 0001011000101110
// Reversed Binary: 0111010001101000
// Reversed Integer: 29800
```

213. Greater Than 15 Checker

This program checks if a given integer is **greater than 15**. If it is, it returns the number itself; otherwise, it returns 15. It prints both the **original number** and the **result**.

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

void greater_than_15_checker(int number) {
    int result = (number > 15) ? number : 15;
    cout << "Given Number: " << number << endl;
    cout << "Result: " << result << endl;
}

int main() {
    greater_than_15_checker(20);  // Test with number
greater than 15
    greater_than_15_checker(10);  // Test with number less
than or equal to 15
    return 0;
}

// Given Number: 20
// Result: 20
// Given Number: 10
// Result: 15
```

214. Replace First Digit with $

This program replaces **the first digit** in a string with a `$` sign. It traverses the string and stops after replacing the first digit.

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

void replace_first_digit_with_dollar(string &input_string)
{
    bool found_digit = false;  // Flag to check if the
first digit is found

    for (char &ch : input_string) {
        if (isdigit(ch) && !found_digit) {
            ch = '$';  // Replace first digit with '$'
            found_digit = true;  // Mark that a digit has
been replaced
        }
    }

    cout << "Modified String: " << input_string << endl;
}

int main() {
    string str = "abc123xyz456";
    cout << "Original String: " << str << endl;
    replace_first_digit_with_dollar(str);
    return 0;
}

// Original String: abc123xyz456
// Modified String: abc$23xyz456
```

215. Prefix Sums

This program computes the **prefix sums** of an integer array. The prefix sum at position `i` is the sum of all elements from index `0` to `i`. It prints both the original array and the prefix sums array.

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

void prefix_sums(const vector<int>& input_array) {
    int prefix_sum = 0;

    cout << "Original Array: [";
    for (size_t i = 0; i < input_array.size(); i++) {
        cout << input_array[i];
        if (i < input_array.size() - 1) cout << ", ";
    }
    cout << "]" << endl;

    cout << "Prefix Sums Array: [";
    for (size_t i = 0; i < input_array.size(); i++) {
        prefix_sum += input_array[i];
        cout << prefix_sum;
        if (i < input_array.size() - 1) cout << ", ";
    }
    cout << "]" << endl;
}

int main() {
    vector<int> input_array = {1, 2, 3, 4, 5};
    prefix_sums(input_array);
    return 0;
}

// Original Array: [1, 2, 3, 4, 5]
// Prefix Sums Array: [1, 3, 6, 10, 15]
```

216. Next Prime Number

This program finds the **next prime number** after a given integer.

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

// Function to check if a number is prime
bool is_prime(int num) {
    if (num < 2) return false;
    for (int i = 2; i <= sqrt(num); i++) {
        if (num % i == 0) return false;
    }
    return true;
}

// Function to find the next prime number after a given
number
void next_prime_number(int given_number) {
    int next_number = given_number + 1;
    while (!is_prime(next_number)) {
        next_number++;
    }

    cout << "Given Number: " << given_number << endl;
    cout << "Next Prime Number: " << next_number << endl;
}

int main() {
    next_prime_number(10);   // Test case
    next_prime_number(17);   // Another test
    return 0;
}

// Given Number: 10
// Next Prime Number: 11
// Given Number: 17
// Next Prime Number: 19
```

217. Reverse Order of Bits

This program reverses the bits of an **8-bit unsigned integer**.

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

// Function to reverse the bits of an 8-bit unsigned
integer
void reverse_order_of_bits(unsigned char integer) {
    unsigned char reversed_integer = 0;
    unsigned char original_integer = integer;

    for (int i = 0; i < 8; i++) {
        reversed_integer <<= 1;
        if (integer & 1) {
            reversed_integer |= 1;
        }
        integer >>= 1;
    }

    cout << "Original Integer: " <<
static_cast<int>(original_integer) << endl;
    cout << "Binary Representation: " <<
bitset<8>(original_integer) << endl;
    cout << "Reversed Binary: " <<
bitset<8>(reversed_integer) << endl;
    cout << "Reversed Integer: " <<
static_cast<int>(reversed_integer) << endl << endl;
}

int main() {
    reverse_order_of_bits(14);
    reverse_order_of_bits(56);
    reverse_order_of_bits(234);
    return 0;
}

// Original Integer: 14
```

```
// Binary Representation: 00001110
// Reversed Binary: 01110000
// Reversed Integer: 112

// Original Integer: 56
// Binary Representation: 00111000
// Reversed Binary: 00011100
// Reversed Integer: 28

// Original Integer: 234
// Binary Representation: 11101010
// Reversed Binary: 01010111
// Reversed Integer: 87
```

218. Pyramid Pattern

This program prints a **pyramid (triangle) pattern** of stars (*) in the console.

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

// Function to generate an ASCII pyramid
void generate_ascii_triangle(int height) {
    for (int i = 1; i <= height; i++) {
        // Print leading spaces
        for (int j = 0; j < height - i; j++) {
            cout << " ";
        }

        // Print stars
        for (int j = 0; j < (i * 2 - 1); j++) {
            cout << "*";
        }

        // Move to the next line
        cout << endl;
    }
}

int main() {
    int triangle_height = 5;
    generate_ascii_triangle(triangle_height);
    return 0;
}

//     *
//    ***
//   *****
//  *******
// *********
```

Congratulations!

In very **line of code**, they have woven a story of innovation and creativity.

This book has been your compass in the vast world of **C++**.

Close this chapter knowing that every challenge overcome is an achievement, and every solution is a step toward mastery.

Thank you for allowing me to be part of your journey.

With gratitude, **Hernando Abella**…

Author of **200+ C++ Programs for Beginners**

Get your bonus books here:

www.hernandoabella.com