C++ Problem Sets

Problem Set 1: Basic Input/Output and Variables

Problem 1.1: Personal Greeting

- **Description:** Write a C++ program that asks the user for their name and their favorite color. Then, it should print a greeting message that includes their name and favorite color.
- **Concepts to use:** `std::cout`, `std::cin`, `std::string`, `std::getline()`.

Problem 1.2: Simple Calculator

- **Description:** Write a C++ program that prompts the user to enter two integers. Then, it should calculate and display their sum, difference, product, and quotient.
- **Concepts to use:** `std::cout`, `std::cin`, `int`, arithmetic operators (`+`, `-`, `*`, `/`).

Problem Set 2: Conditionals (`if`, `else if`, `else`)

Problem 2.1: Positive, Negative, or Zero

- **Description:** Write a C++ program that asks the user to enter an integer. The program should then determine if the number is positive, negative, or zero, and print an appropriate message.
- **Concepts to use:** `std::cout`, `std::cin`, `int`, `if`, `else if`, `else`, comparison operators.

Problem 2.2: Voting Eligibility

- **Description:** Write a C++ program that asks the user for their age. If the age is 18 or greater, print "You are eligible to vote." Otherwise, print "You are not yet eligible to vote."
- **Concepts to use:** `std::cout`, `std::cin`, `int`, `if`, `else`, comparison operators.

Problem Set 3: Loops (`for`)

Problem 3.1: Countdown

- **Description:** Write a C++ program that uses a `for` loop to print a countdown from 10 down to 1, followed by "Blast Off!".
- **Concepts to use:** `std::cout`, `for` loop.

Problem 3.2: Sum of N Numbers

- **Description:** Write a C++ program that asks the user to enter a positive integer `N`. Then, use a `for` loop to calculate and display the sum of all integers from 1 to `N`.
- **Concepts to use:** `std::cout`, `std::cin`, `int`, `for` loop, arithmetic operator.

Problem Set 4: Star Patterns with `for` Loops

Problem 4.1: Solid Rectangle of Stars

- **Description:** Write a C++ program that asks the user for a `width` and `height`. Then, it should print a solid rectangle of stars (`*`) of the specified dimensions.
- **Example (width=5, height=3):**

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

```
*****
```

- **Concepts to use:** `std::cout`, `std::cin`, `int`, nested `for` loops.

Problem 4.2: Right-Angled Triangle of Stars (Right-aligned)

- **Description:** Write a C++ program that asks the user for the `height` of the triangle. Then, it should print a right-angled triangle of stars, where the hypotenuse is on the right side.
- **Example (height=5):**

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

- **Concepts to use:** `std::cout`, `std::cin`, `int`, nested `for` loops, understanding spaces.

Problem 4.3: Inverted Right-Angled Triangle of Stars (Left-aligned)

- **Description:** Write a C++ program that asks the user for the `height` of the triangle. Then, it should print an inverted right-angled triangle of stars, left-aligned.
- **Example (height=5):**

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

- **Concepts to use:** `std::cout`, `std::cin`, `int`, nested `for` loops.

Problem Set 5: Centered Triangle of Stars

Problem 5.1: Centered Pyramid of Stars

- **Description:** Write a C++ program that asks the user for the `height` of the pyramid. Then, it should print a perfectly centered pyramid of stars.
- **Example (height=5):**

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

- **Concepts to use:** `std::cout`, `std::cin`, `int`, nested `for` loops, understanding how the number of spaces and stars changes per row.

Solutions

(Try to solve them first before looking here!)

Solution for Problem Set 1

Solution 1.1: Personal Greeting

C++

```
#include <iostream> // Required for std::cout, std::cin
#include <string>   // Required for std::string, std::getline

int main() {
    std::string name;
```

```
    std::string favColor;

    std::cout << "What is your name? ";
    // Use getline for names that might contain spaces
    std::getline(std::cin, name);

    std::cout << "What is your favorite color? ";
    // Use getline for colors that might contain spaces (though
less common)
    std::getline(std::cin, favColor);

    std::cout << "Hello, " << name << "! Your favorite color is
" << favColor << "." << std::endl;

    return 0; // Indicates successful execution
}
```

Explanation 1.1:

- We include `<iostream>` for console I/O and `<string>` for `std::string` and `std::getline`.
- We declare two `std::string` variables: `name` and `favColor`.
- `std::getline(std::cin, name);` reads the entire line entered by the user, including spaces, until they press Enter, and stores it in `name`. This is crucial for names like "John Doe".
- Similarly for `favColor`.
- Finally, we use `std::cout` and the `<<` operator to construct and print the personalized greeting. `std::endl` adds a newline and flushes the output buffer.

Solution 1.2: Simple Calculator

C++

```
#include <iostream> // Required for std::cout, std::cin

int main() {
    int num1, num2;

    std::cout << "Enter the first integer: ";
    std::cin >> num1;

    std::cout << "Enter the second integer: ";
    std::cin >> num2;

    // Calculate and display results
    std::cout << "Sum: " << (num1 + num2) << std::endl;
    std::cout << "Difference: " << (num1 - num2) << std::endl;
    std::cout << "Product: " << (num1 * num2) << std::endl;

    // Check for division by zero before performing division
    if (num2 != 0) {
        std::cout << "Quotient: " << (static_cast<double>(num1)
/ num2) << std::endl;
    } else {
        std::cout << "Cannot divide by zero!" << std::endl;
    }

    return 0;
}
```

Explanation 1.2:

- We declare two `int` variables, `num1` and `num2`, to store the input numbers.
- We use `std::cin >> num1;` and `std::cin >> num2;` to read the integers.
- We perform the arithmetic operations directly within the `std::cout` statements. The parentheses `()` ensure the operation is done before the result is sent to `std::cout`.

- **Important:** For quotient, we added an `if (num2 != 0)` check. Dividing by zero leads to a runtime error, so it's good practice to prevent it.
- `static_cast<double>(num1)` is used to temporarily convert `num1` to a `double` before division. This ensures that the division results in a floating-point number (e.g., 5 / 2 becomes 2.5 instead of 2 due to integer division).

Solution for Problem Set 2

Solution 2.1: Positive, Negative, or Zero

C++

```
#include <iostream> // Required for std::cout, std::cin

int main() {
    int number;

    std::cout << "Enter an integer: ";
    std::cin >> number;

    if (number > 0) {
        std::cout << "The number is positive." << std::endl;
    } else if (number < 0) {
        std::cout << "The number is negative." << std::endl;
    } else { // This means number must be 0
        std::cout << "The number is zero." << std::endl;
    }

    return 0;
}
```

Explanation 2.1:

- We take an `int` input from the user.
- The `if` condition `(number > 0)` checks if it's positive.

- If not, `else if (number < 0)` checks if it's negative.
- If neither of those is true, the `else` block executes, meaning the number must be zero. This demonstrates the cascading nature of `if-else if-else` statements.

Solution 2.2: Voting Eligibility

C++

```
#include <iostream> // Required for std::cout, std::cin

int main() {
    int age;

    std::cout << "Please enter your age: ";
    std::cin >> age;

    if (age >= 18) {
        std::cout << "You are eligible to vote." << std::endl;
    } else {
        std::cout << "You are not yet eligible to vote." <<
std::endl;
    }

    return 0;
}
```

Explanation 2.2:

- We get the user's `age` as an integer.
- The `if` condition `(age >= 18)` directly checks the eligibility.
- If true, the first message is printed; otherwise, the `else` block is executed, printing the second message. This is a straightforward use of an `if-else` structure.

Solution for Problem Set 3

Solution 3.1: Countdown

C++

```
#include <iostream> // Required for std::cout

int main() {
    for (int i = 10; i >= 1; --i) {
        std::cout << i << std::endl;
    }
    std::cout << "Blast Off!" << std::endl;

    return 0;
}
```

Explanation 3.1:

- The `for` loop is set up as follows:
  - **Initialization:** `int i = 10;` starts the counter `i` at 10.
  - **Condition:** `i >= 1;` the loop continues as long as `i` is greater than or equal to 1.
  - **Iteration:** `--i;` decrements `i` by 1 after each iteration.
- Inside the loop, `std::cout << i << std::endl;` prints the current value of `i`.
- Once the loop finishes (when `i` becomes 0), "Blast Off!" is printed.

Solution 3.2: Sum of N Numbers

C++

```
#include <iostream> // Required for std::cout, std::cin

int main() {
    int N;
    int sum = 0; // Initialize sum to 0

    std::cout << "Enter a positive integer (N): ";
    std::cin >> N;

    // Optional: Input validation
    if (N <= 0) {
        std::cout << "N must be a positive integer." <<
std::endl;
        return 1; // Indicate an error
    }

    for (int i = 1; i <= N; ++i) {
        sum = sum + i; // Or sum += i; (shorthand operator)
    }

    std::cout << "The sum of numbers from 1 to " << N << " is: "
<< sum << std::endl;

    return 0;
}
```

Explanation 3.2:

- We declare an `int N` to store the user's input and `int sum = 0;` to accumulate the sum. It's crucial to initialize `sum` to 0.
- We added an `if (N <= 0)` check for basic input validation. If `N` isn't positive, it prints an error and exits. `return 1;` indicates an error, while `return 0;` indicates success.
- The `for` loop:
  - **Initialization:** `int i = 1;` starts counting from 1.
  - **Condition:** `i <= N;` the loop continues as long as `i` is less than or equal to `N`.

    - **Iteration:** `++i;` increments `i` by 1.
- Inside the loop, `sum = sum + i;` (or `sum += i;`) adds the current value of `i` to the `sum`.
- After the loop completes, the final `sum` is printed.

Solutions (Problem Set 4)

(Remember to try solving them yourself first!)

Solution for Problem 4.1: Solid Rectangle of Stars

C++

```
#include <iostream>

int main() {
    int width, height;

    std::cout << "Enter the width of the rectangle: ";
    std::cin >> width;

    std::cout << "Enter the height of the rectangle: ";
    std::cin >> height;

    // Outer loop for rows
    for (int i = 0; i < height; ++i) {
        // Inner loop for columns (printing stars)
        for (int j = 0; j < width; ++j) {
            std::cout << "*";
        }
        std::cout << std::endl; // Move to the next line after
each row
    }
```

```
    return 0;
}
```

Explanation 4.1:

- We get `width` and `height` from the user.
- The **outer `for` loop** (`for (int i = 0; i < height; ++i)`) runs `height` number of times. Each iteration of this loop represents one row.
- The **inner `for` loop** (`for (int j = 0; j < width; ++j)`) runs `width` number of times for *each* row. It prints a `*` without a newline (`std::endl`) so that stars appear on the same line.
- After the inner loop finishes printing all stars for a row, `std::cout << std::endl;` is called. This moves the cursor to the next line, preparing for the stars of the next row.

Solution for Problem 4.2: Right-Angled Triangle of Stars (Right-aligned)

C++

```
#include <iostream>

int main() {
    int height;

    std::cout << "Enter the height of the triangle: ";
    std::cin >> height;

    // Outer loop for rows
    for (int i = 1; i <= height; ++i) {
        // Inner loop 1: Print spaces
        // Number of spaces decreases with each row
        // For row 'i', print (height - i) spaces
        for (int j = 1; j <= (height - i); ++j) {
            std::cout << " ";
        }
```

```
        // Inner loop 2: Print stars
        // Number of stars increases with each row
        // For row 'i', print 'i' stars
        for (int k = 1; k <= i; ++k) {
            std::cout << "*";
        }
        std::cout << std::endl; // Move to the next line
    }

    return 0;
}
```

Explanation 4.2:

- This pattern requires **two inner loops** for each row: one for spaces and one for stars.
- The **outer `for` loop** (`for (int i = 1; i <= height; ++i)`) iterates from 1 to `height`, representing each row.
- **First inner loop (for spaces):** `for (int j = 1; j <= (height - i); ++j)`
  - In the first row (`i=1`), it prints `height - 1` spaces.
  - In the second row (`i=2`), it prints `height - 2` spaces, and so on.
  - In the last row (`i=height`), it prints `height - height = 0` spaces. This correctly pushes the stars to the right.
- **Second inner loop (for stars):** `for (int k = 1; k <= i; ++k)`
  - In the first row (`i=1`), it prints 1 star.
  - In the second row (`i=2`), it prints 2 stars, and so on.
  - In the last row (`i=height`), it prints `height` stars.
- `std::cout << std::endl;` moves to the next line after spaces and stars for the current row are printed.

Solution for Problem 4.3: Inverted Right-Angled Triangle of Stars (Left-aligned)

C++

```
#include <iostream>

int main() {
    int height;

    std::cout << "Enter the height of the triangle: ";
    std::cin >> height;

    // Outer loop for rows
    // We can iterate downwards or upwards, depending on how we
calculate stars
    for (int i = height; i >= 1; --i) { // Start from height, go
down to 1
        // Inner loop for printing stars
        // For row 'i', print 'i' stars
        for (int j = 1; j <= i; ++j) {
            std::cout << "*";
        }
        std::cout << std::endl; // Move to the next line
    }

    // Alternative outer loop (iterating upwards):
    /*
    for (int i = 1; i <= height; ++i) {
        // Number of stars for current row = (height - i + 1)
        for (int j = 1; j <= (height - i + 1); ++j) {
            std::cout << "*";
        }
        std::cout << std::endl;
    }
    */

    return 0;
}
```

Explanation 4.3:

- This pattern can be solved by simply adjusting the `for` loop's iteration for the number of stars.
- **My preferred approach (in the solution):**
  - The **outer `for` loop** (`for (int i = height; i >= 1; --i)`) starts `i` at `height` and decrements it down to 1.
  - The **inner `for` loop** (`for (int j = 1; j <= i; ++j)`) simply prints `i` stars.
    - When `i` is `height`, it prints `height` stars.
    - When `i` is `height - 1`, it prints `height - 1` stars.
    - ...until `i` is 1, it prints 1 star.
- **Alternative approach (commented out in the solution):**
  - If you keep the outer loop `for (int i = 1; i <= height; ++i)`, then the number of stars needed for each row `i` is `(height - i + 1)`. This also works perfectly!

Solution (Problem Set 5)

(Seriously, try it yourself first! This one requires a bit more thought.)

Solution for Problem 5.1: Centered Pyramid of Stars

C++

```
#include <iostream>

int main() {
    int height;

    std::cout << "Enter the height of the pyramid: ";
    std::cin >> height;

    // Outer loop for rows
    for (int i = 1; i <= height; ++i) {
        // Inner loop 1: Print leading spaces
        // Number of spaces: height - current_row_number
```

```
        for (int j = 1; j <= (height - i); ++j) {
            std::cout << " ";
        }

        // Inner loop 2: Print stars
        // Number of stars: (2 * current_row_number) - 1
        for (int k = 1; k <= (2 * i - 1); ++k) {
            std::cout << "*";
        }
        std::cout << std::endl; // Move to the next line
    }

    return 0;
}
```