3. Control Flow Statements

1. What are conditional statements in C++? Explain the if-else and switch statements.

- Conditional statements in C++ are control structures that allow a program to execute different blocks of code based on whether a specific condition is true or false. They are essential for creating dynamic and responsive software.
  - **if-else Statement:**
    The if-else statement is the most basic form of a conditional statement. It evaluates a condition, and if the condition is true, it executes a specific block of code. If the condition is false, it executes a different block of code.
    - Syntax:
      ```
      if (condition) {
          // Code to be executed if the condition is true
      } else {
          // Code to be executed if the condition is false
      }
      ```
    - Example:
      ```
      int age = 20;
      if (age >= 18) {
          std::cout << "You are an adult." << std::endl;
      } else {
          std::cout << "You are a minor." << std::endl;
      }
      ```
  - **switch Statement:**
    The switch statement is used to select one of several code blocks to execute based on the value of a variable. It is often used as an alternative to multiple if-else statements when dealing with multiple possible values for a single variable.

- Syntax:
```
switch (expression) {
    case value1:
        // Code to be executed if expression == value1
        break;
    case value2:
        // Code to be executed if expression == value2
        break;
    // ...
    default:
        // Code to be executed if no case matches
}
```
- Example:
```
int day = 3;
switch (day) {
    case 1:
        std::cout << "Monday" << std::endl;
        break;
    case 2:
        stdoba::cout << "Tuesday" << std::endl;
        break;
    case 3:
        std::cout << "Wednesday" << std::endl;
        break;
    default:
        std::cout << "Invalid day" << std::endl;
}
```

2. What is the difference between for, while, and do-while loops in C++?

- Here's a breakdown of the differences between for, while, and do-while loops in C++:

  1. for loop:

- Structure:

  for (initialization; condition; increment/decrement) {// code to be executed}.

- Use case:

  Best when you know the number of iterations in advance. It's often used for iterating through a sequence of numbers or elements in a container.

- Execution:
  - The initialization statement is executed only once at the beginning.
  - The condition is checked before each iteration. If it's true, the loop body is executed.
  - After each iteration,
    the increment/decrement statement is executed.
  - The loop continues as long as the condition remains true.

- Example:

```
for (int i = 0; i < 5; i++) {
    cout << i << " "; // Output: 0 1 2 3 4
}
```

2. while loop:

- Structure:
  while (condition) {// code to be executed}

- Use case:

  Best when you need to repeat a block of code as long as a condition is true, and you don't necessarily know the number of iterations beforehand.

- Execution**:**
    - The condition is checked before each iteration. If it's true, the loop body is executed.
    - The loop continues as long as the condition remains true.
    - If the condition is false from the beginning, the loop body is skipped entirely.
- Example:

```
int count = 0;
while (count < 3) {
    cout << "Count: " << count << endl; // Output: Count: 0, Count: 1, Count: 2
    count++;
}
```

3. do-while loop:

- Structure**:**
  do {// code to be executed} while (condition);

- Use case:
  Similar to while, but it guarantees that the loop body will execute at least once, regardless of the initial condition. Useful when you want to execute a block of code and then check if it should be repeated

- Execution**:**
    - The loop body is executed once.
    - The condition is checked after the first execution. If it's true, the loop body is executed again.
    - The loop continues as long as the condition remains true.

- Example:

```
int input;
do {
    cout << "Enter a positive number: ";
    cin >> input;
} while (input <= 0); // Loop continues until a positive number is entered.
```

3. How are break and continue statements used in loops? Provide examples.

break

- Purpose:

  Immediately exits the *innermost* loop (or switch), skipping all remaining code and stopping further iterations.

- Common use:

  Stop looping early when a condition is met—like finding a target in an array.

- Examples**:**

  i. **for loop** – stop at i == 3

```
for (int i = 1; i <= 5; ++i) {
    if (i == 3) {
```

```
        break;
    }
    std::cout << i << "\n";
}
```

ii. **while (true) loop** – exit on negative input

```
int sum = 0, x;
while (true) {
    std::cin >> x;
    if (x < 0) break;
    sum += x;
}
std::cout << sum;
```

iii. **Nested loops** – only inner loop exits.

```
for (int i = 1; i <= 3; ++i) {
    for (int j = 1; j <= 3; ++j) {
        if (i == 2) break;
        std::cout << i << "," << j << "\n";
    }
}
// i=2 loop exited immediately; i=3 runs full inner
loop :contentReference[oaicite:8]{index=8}
```

Continue

- Purpose:

  Skips the rest of the current iteration and immediately jumps to the loop's next iteration.

- Common use:

  Skip processing for specific cases without stopping the whole loop.

- Examples:

  i. **for loop** – skip i == 3

  ```
  for (int i = 1; i <= 5; ++i) {
    if (i == 3) continue;
    std::cout << i << "\n";
  }
  // Output: 1 2 4 5
  :contentReference[oaicite:14]{index=14}
  ```

  ii. **while loop** – ignore values > 50

  ```
  int sum = 0, x = 0;
  while (x >= 0) {
    sum += x;
    std::cin >> x;
    if (x > 50) continue;
  ```

```
}

std::cout << sum;

// Skips numbers >50
:contentReference[oaicite:15]{index=15}
```

iii. **Nested loops** – skip inner iteration when j == 2

```
for (int i = 1; i <= 3; ++i) {

   for (int j = 1; j <= 3; ++j) {

      if (j == 2) continue;

      std::cout << i << "," << j << "\n";

   }

}

// Omits j=2 each time
:contentReference[oaicite:16]{index=16}
```

4. Explain nested control structures with an example.

- A nested control structure is simply a control flow statement (like if, for, while) placed **inside** another. This allows you to model complex decision logic or multi-level iteration.

1. Nested if-else:

- Example: Check a Number's Sign and Parity

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

int main() {
   int num = 10;
```

```
    if (num > 0) {
        cout << "Number is positive.\n";
        if (num % 2 == 0) {
            cout << "Number is even.\n";
        } else {
            cout << "Number is odd.\n";
        }
    } else {
        cout << "Number is non-positive.\n";
    }
    return 0;
}
```

- Outer if checks if num is positive.
- Inner if-else then determines if it's even or odd.

2. **Nested Loops**

- Example: 3×3 Coordinate Grid using while
  -

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

int main() {
    int i = 0;
    while (i < 3) {
        int j = 0;
        while (j < 3) {
            cout << "(" << i << "," << j << ") ";
            j++;
        }
        cout << endl;
        i++;
    }
    return 0;
}
```

- Outer while controls rows (i).

    - Inner while controls columns (j), printing every (i, j) pair.

- Example: Nested for Loop - Print weeks and days:

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

int main() {
   int weeks = 3, days = 7;
   for (int i = 1; i <= weeks; ++i) {
      cout << "Week: " << i << "\n";
      for (int j = 1; j <= days; ++j) {
         cout << "  Day: " << j << "\n";
      }
   }
   return 0;
}
```

- Outer loop runs through each **week**.
- Inner loop lists **days** within that week.