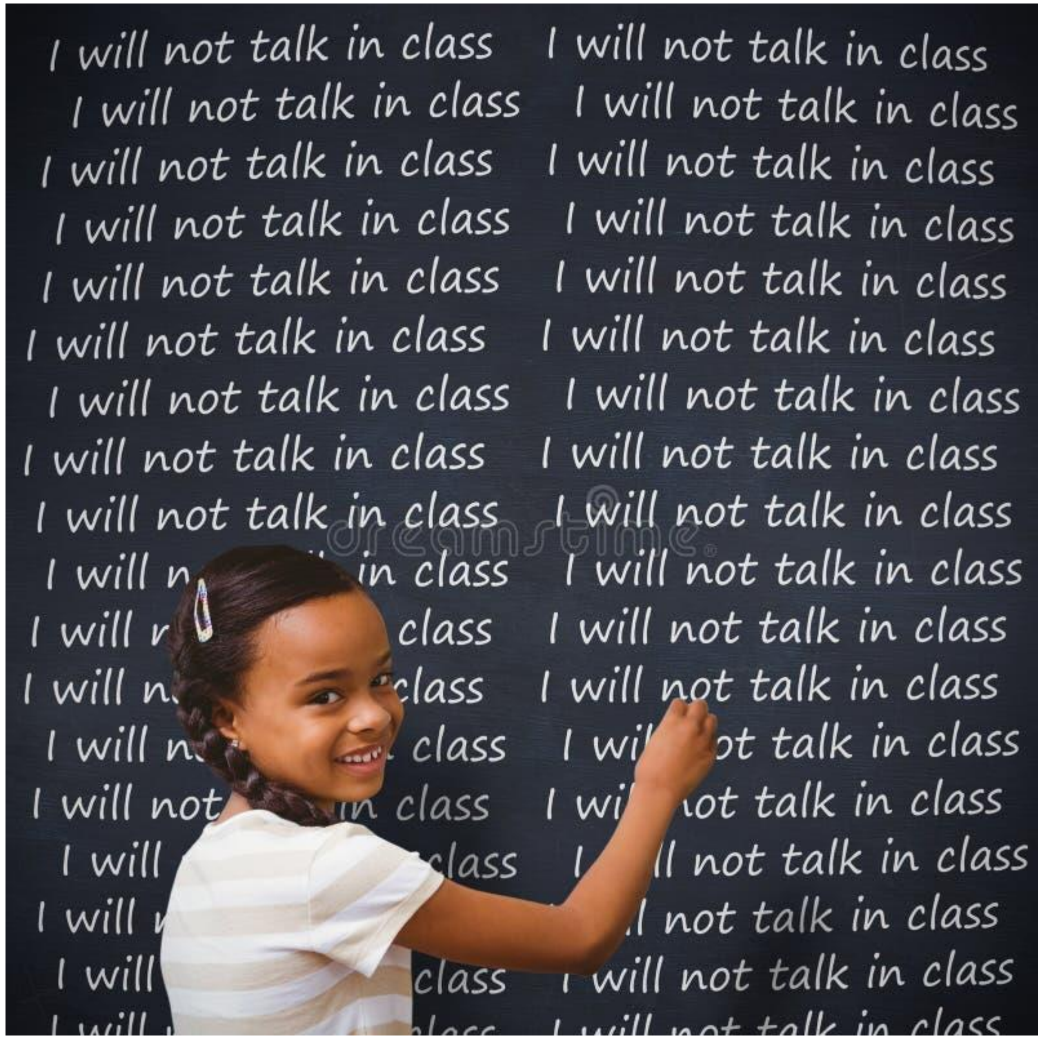
Fundamentals of Computer Programming

CS-110

Course Instructor: Dr. Momina Moetesum

Repetition Structures (Loops)

Week 4-B



Learnning Objectives

01

To understand the purpose of loops in programming

02

To differentiate between *for, while*, and *do-while* loop

03

To implement different loops to solve problems

04

To understand the use of *break* and *continue*

While Loop

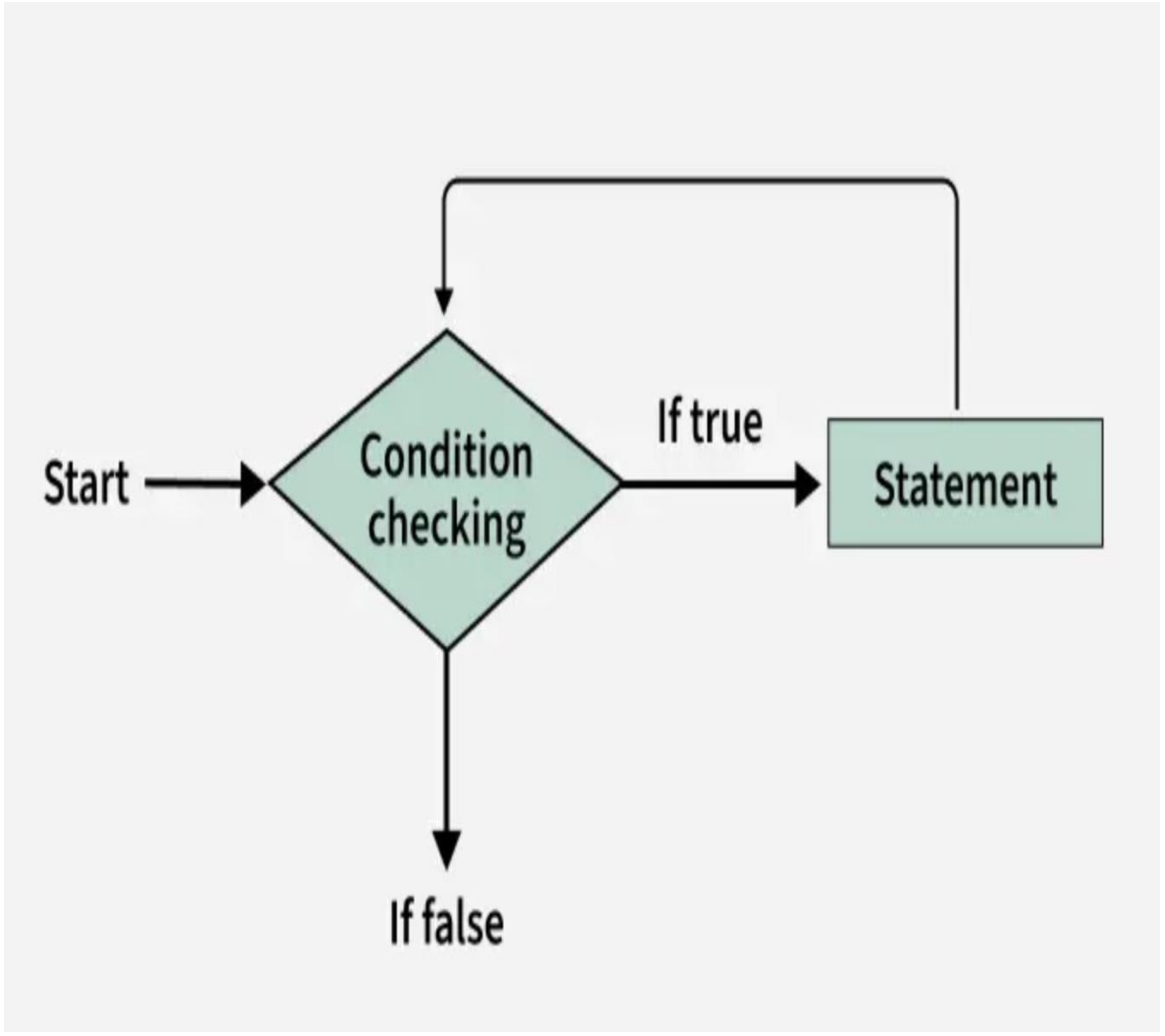
The while loop is also an **entry-controlled loop** which is used in situations where we do not know the exact number of iterations of the loop beforehand.

In for loop, we have seen that the number of iterations is **known beforehand**, i.e. the number of times the loop body is needed to be executed is known to us and we create the condition on the basis of it. But while loops execution is solely based on the condition.

While-Loop Syntax

- Only the **condition** is the part of while loop syntax, we have to **initialize** and **update loop** variable manually.

```
while (condition) {
   // Body of the loop
   // update expression
}
```



Example

Output

```
1 2 3 4 5
```

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

int main() {

        int i = 1;   // Initialization
        // while loop that starts with i = 1 and ends
        // when i is greater than 5.
        while (i <= 5) {
                cout << i << " ";
                i++;  // Updation
            }
   return 0;
}
```

Practice Problem:

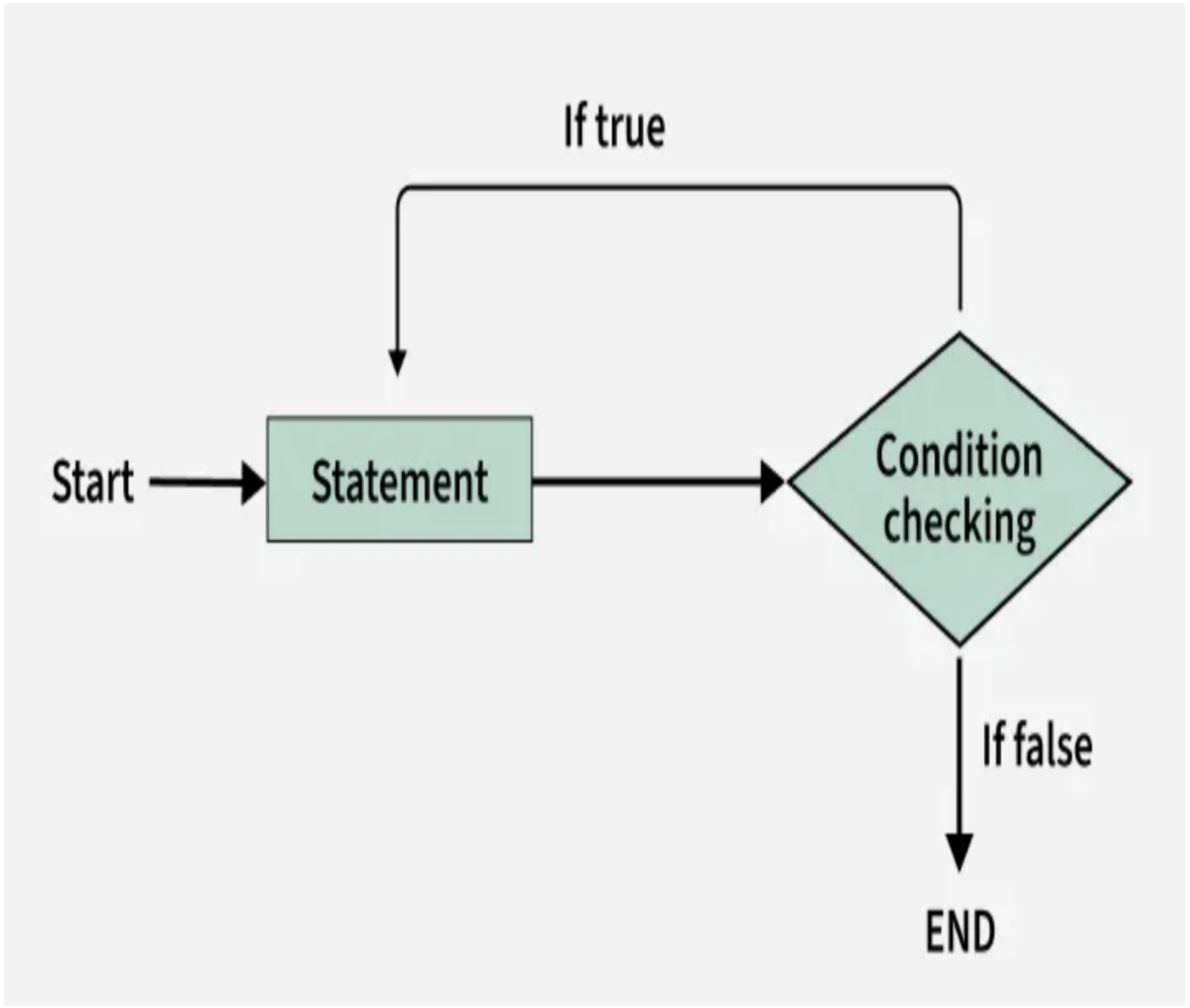
A **while loop** is best when the number of iterations is ***unknown*** and depends on **user input** or **some condition being met** (like guessing until correct)

Write a C++ program that generates a secret number between 1 and 100 (you can hardcode the number for simplicity, e.g., 42). The program should repeatedly ask the user to guess the number until they get it right.

- If the guess is too high, display **"Too high, try again."**
- If the guess is too low, display **"Too low, try again."**
- Stop only when the user guesses correctly, then print **"Congratulations! You found the number."**

Do-While Loop

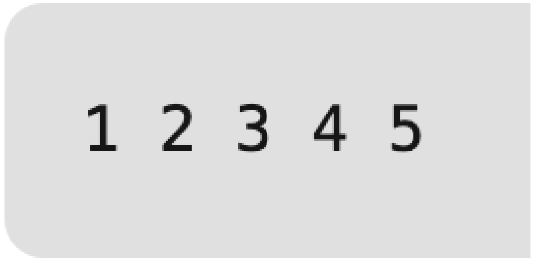
- The do-while loop is an **exit-controlled loop** which means the condition is checked after executing the body of the loop.
- In a do-while loop, the loop body will **execute at least once** irrespective of the test condition.



Do-While Loop Syntax

```
do {
   // Body of the loop
   // update expression
} while (condition);
```

Output



```
1 2 3 4 5
```

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

int main()
{
        int i = 1; // Initialization
        // while loop that starts with i = 1 and
ends
        // when i is greater than 5.
        do {
                cout << i << " ";   // Updation
                   i++;
              }while (i <= 5);
   return 0;
}
```

Practice Problem:

- Write a C++ program that displays a simple calculator menu to the user:
  - 1. Add two numbers
  - 2. Subtract two numbers
  - 3. Multiply two numbers
  - 4. Divide two numbers
  - 5. Exit
- The program should:
  - Ask the user to choose an option.
  - Perform the corresponding operation.
  - After showing the result, the menu should appear again.
  - Continue until the user selects **5 (Exit)**.

Infinite Loops

An **infinite loop** (sometimes called an endless loop) is a piece of coding that lacks a **functional exit** so that it repeats indefinitely.

An infinite loop occurs when a condition is always evaluated to be true.

Usually, this is an error.

We can manually create an infinite loop using all three loops.

Example:

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

int main()
{
   // This is an infinite for loop as the condition
   // expression is blank
   for (;;)
   {
      cout << "This loop will run forever.\n";
   }
       return 0;
}
```

Breaking out of the Loop

- The **break** statement when encountered, terminates the loop regardless of its test condition.
- This is useful when the execution of loop is dependent on multiple conditions.

Give Output of the Program

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

int main() {
   for (int i = 0; i < 5; i++) {
      // Terminating before reaching i = 4
      if (i == 2)
         break;
      cout << "Hi" << endl;
   }
   return 0;
}
```

Skipping an Iteration

- The **continue** statement, when encountered, skips the remaining code inside the current iteration of the loop and proceeds to the next iteration. This is useful when you want to skip specific iterations based on certain conditions but still want to continue the loop execution.

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

int main() {
    for (int i = 0; i < 5; i++) {
        // Skipping when i equals 2
        if (i == 2)
            continue;
        cout << "Hi" << endl;
    }
    return 0;
}
```

Give Output of the Program

Acknowledgment

- Content of these slides are taken from:
  - https://www.geeksforgeeks.org/
  - https://www.tutorialspoint.com/
  - https://www.programiz.com/