UNIVERSITY OF MILA

Faculty of Science and Technology

Department of Process Engineering

Level: 1st year ST - ENG & LMD

Introduction to Programming

Lecture 01 & 02:

Conditional and Iterative Control Structures

Loop Statements

---

by:

Dr. Farouk KECITA

Academic Year 2025/2026

1 Introduction to Loops

A loop is a block of statements that performs a set of instructions repeatedly.

In loops, we repeat a particular portion of the program either:

- A specified number of times, or
- Until a particular condition is satisfied

1.1 Types of Loops in C Programming

There are mainly two types of loops in C Programming:

Entry Controlled Loops:

- The test condition is checked before entering the main body of the loop.
- If the condition is false, the loop body will not execute.
- Examples: `for` loop and `while` loop

Exit Controlled Loops:

- The test condition is evaluated at the end of the loop body.
- The loop body will execute at least once, irrespective of whether the condition is true or false.
- Example: `do-while` loop

Loop Type	Description
`for` loop	First initializes, then condition check, then executes the body, and at last, the update is done
`while` loop	First initializes, then condition checks, and then executes the body; updating can be inside the body
`do-while` loop	First executes the body, and then the condition check is done

Table 1: Comparison of Loop Types in C

2 The for Loop

The `for` loop in C programming is a repetition control structure that allows programmers to write a loop that will be executed a specific number of times.

The `for` loop enables programmers to perform N number of steps together in a single line.

2.1 Syntax of for Loop

Syntax:

```
for (initialize expression; test expression; update expression) {
    body of for loop;
}
```

2.2 How for Loop Works

1. The initialization statement is executed only once.
2. Then, the test expression is evaluated.
3. If the test expression is evaluated to false, the for loop is terminated.
4. If the test expression is evaluated to true, statements inside the body of for loop are executed, and the update expression is updated.
5. Again the test expression is evaluated.
6. This process continues until the test expression becomes false.

2.3 Examples of for Loop

Example 1: C Program to Print "Hello World" 4 times using For Loop

Listing 1: Print "Hello World" 4 times

```c
#include <stdio.h>
int main() {
    int i;

    for(i = 1; i <= 4; i++) {
        printf("Hello World\n");
    }

    return 0;
}
```

```
Hello World
Hello World
Hello World
Hello World
```

Example 2: Write a C program to print all even numbers between 1 to 10.

Listing 2: Print even numbers from 1 to 10

```c
#include <stdio.h>
int main() {
    int i;

    printf("Even numbers from 1 to 10:\n");
    for(i = 2; i <= 10; i += 2) {
        printf("%d ", i);
    }
    printf("\n");

    return 0;
}
```

```
Even numbers from 1 to 10:
2 4 6 8 10
```

3 The while Loop

The `while` loop does not depend upon the number of iterations. In the `for` loop, the number of iterations was previously known, but in the `while` loop, the execution is terminated on the basis of the test condition.

If the test condition becomes false, it will break from the while loop; otherwise, the body will be executed.

3.1 Syntax of while Loop

Syntax:

```
initialization_expression;
while (test_expression) {
    body of the while loop;
    update_expression;
}
```

3.2 How while Loop Works

1. The condition may be any expression, and true is any non-zero value.
2. The loop iterates while the condition is true.
3. When the condition becomes false, the program control passes to the statement immediately following the loop.

Important Note A `while` loop might not execute at all. When the condition is tested and the result is false, the loop body will be skipped and the first statement after the while loop will be executed.

3.3 Examples of while Loop

Example 1: C Program to Print "Hello World" 4 times using while Loop

Listing 3: Print "Hello World" 4 times using while

```c
#include <stdio.h>
int main() {
    int i = 1;  // initialization

    while(i <= 4) {  // test condition
        printf("Hello World\n");
        i++;  // update
    }

    return 0;
}
```

```
Hello World
Hello World
Hello World
Hello World
```

Example 2: Write a C program to print all even numbers between 1 to 10 using while loop.

Listing 4: Print even numbers using while

```c
#include <stdio.h>
int main() {
    int i = 2;  // initialization

    printf("Even numbers from 1 to 10:\n");
    while(i <= 10) {  // test condition
        printf("%d ", i);
        i += 2;  // update
    }
    printf("\n");

    return 0;
}
```

```
Even numbers from 1 to 10:
2 4 6 8 10
```

4 The do...while Loop

The `do...while` in C is a loop statement used to repeat some part of the code until the given condition is fulfilled.

Key Characteristics:

- It is an exit-controlled or post-tested loop where the test condition is checked after executing the body of the loop.
- The statements in the `do...while` loop will always be executed at least once no matter what the condition is.
- Unlike the `while` loop, in which condition is checked at the top of the loop; in `do...while`, condition is checked at the bottom.

4.1 Syntax of do...while Loop

Syntax:

```
do {
    body of do...while loop;
} while (condition);
```

4.2 How do...while Loop Works

1. When the program control first comes to the `do...while` loop, the body of the loop is executed first.
2. Then the test condition/expression is checked.
3. When the test condition is evaluated as true, the program control goes to the start of the loop and the body is executed once more.
4. The above process repeats until the test condition becomes false.
5. When the test condition is evaluated as false, the program control moves on to the next statements after the `do...while` loop.

4.3 Examples of do...while Loop

Example 1: Write a C Program to Print ”Hello World” 5 times using do...while Loop.

Listing 5: Print "Hello World" 5 times using do...while

```c
#include <stdio.h>
int main() {
    int i = 1;  // initialization

    do {
        printf("Hello World\n");
        i++;  // update
    } while(i <= 5);  // test condition

    return 0;
}
```

```
Hello World
Hello World
Hello World
Hello World
Hello World
```

Example 2: Write a C program that asks the user to enter a positive number. The program should continue asking for input until the user enters a positive number (greater than zero).

Listing 6: Validate positive number input

```c
#include <stdio.h>
int main() {
    int number;

    do {
        printf("Enter a positive number: ");
        scanf("%d", &number);

        if (number <= 0) {
            printf("Invalid input! ");
            printf("Please enter a positive number.\n");
        }
    } while (number <= 0);  // Repeat until valid

    printf("Thank you! You entered: %d\n", number);
    printf("Square: %d\n", number * number);

    return 0;
}
```

```
Enter a positive number: -5
Invalid input! Please enter a positive number.
Enter a positive number: 10
Thank you! You entered: 10
Square: 100
```

5 Logical Operators and if Statement

Logical operators are used to combine multiple conditions in decision-making statements.

Logical Operators in C:

- AND (&&): Returns true if both conditions are true
- OR (||): Returns true if at least one condition is true
- NOT (!): Reverses the logical state of its operand

5.1 Logical AND (&&) Example

Example 3: Write a C program that checks if a person is eligible to drive legally based on two conditions:

1. The person must be 18 years or older
2. The person must have a valid driver's license

Listing 7: Logical AND operator example

```c
#include <stdio.h>
int main() {
    int age, hasLicense;

    printf("Enter your age: ");
    scanf("%d", &age);
    printf("Do you have a driver's license? (1 for Yes, 0 for No
       ): ");
    scanf("%d", &hasLicense);

    // Using AND operator
    if (age >= 18 && hasLicense == 1) {
        printf("You can drive legally.\n");
    } else {
        printf("You cannot drive legally.\n");
        if (age < 18) {
            printf("Reason: You are under 18.\n");
        }
```

```c
        if (hasLicense != 1) {
            printf("Reason: You don't have a license.\n");
        }
    }

    return 0;
}
```

```
Enter your age: 20
Do you have a driver's license? (1 for Yes, 0 for No): 1
You can drive legally.
```

5.2 Logical OR (||) Example

Example 4: Write a C program that provides weather alerts based on temperature readings.
Program Requirements:

1. Ask the user to enter the current temperature in Celsius.
2. Use the logical OR operator (||) to check if the temperature represents extreme weather conditions:
   - Temperature is $0°C$ or below (freezing)
   - OR temperature is $40°C$ or above (extremely hot)
3. If either extreme condition is true, display "Extreme weather warning!" and appropriate messages.
4. If neither extreme condition applies, display "Weather is comfortable."

Listing 8: Logical OR operator example

```c
#include <stdio.h>
int main() {
    int temperature;

    printf("Enter temperature in Celsius: ");
    scanf("%d", &temperature);

    // Using OR operator
    if (temperature <= 0 || temperature >= 40) {
        printf("Extreme weather warning!\n");

        if (temperature <= 0) {
            printf("It's freezing cold!\n");
            printf("Wear warm clothes.\n");
        }
        if (temperature >= 40) {
            printf("It's extremely hot!\n");
```

```c
            printf("Stay hydrated and avoid sun.\n");
        }
    } else {
        printf("Weather is comfortable.\n");
    }

    return 0;
}
```

```
Enter temperature in Celsius: 45
Extreme weather warning!
It's extremely hot!
Stay hydrated and avoid sun.
```

```
Enter temperature in Celsius: -5
Extreme weather warning!
It's freezing cold!
Wear warm clothes.
```

```
Enter temperature in Celsius: 25
Weather is comfortable.
```

Summary

Loop Type	When to Use	Execution Guarantee
`for`	Known number of iterations	May not execute
`while`	Unknown number of iterations, condition at start	May not execute
`do-while`	Need at least one execution	Executes at least once

Table 2: Summary of Loop Types

Key Takeaways:

- `for` loops are best when you know exactly how many times to iterate
- `while` loops are best when iteration depends on a condition that may be false initially
- `do-while` loops guarantee at least one execution of the loop body
- Logical operators (&&, ||, !) are essential for combining multiple conditions
- Always ensure loops have a proper exit condition to avoid infinite loops