4. Decision Making and Looping

Syeda Jannatul Naim
Lecturer | Dept. of CSE
World University of Bangladesh (WUB)

January, 2025

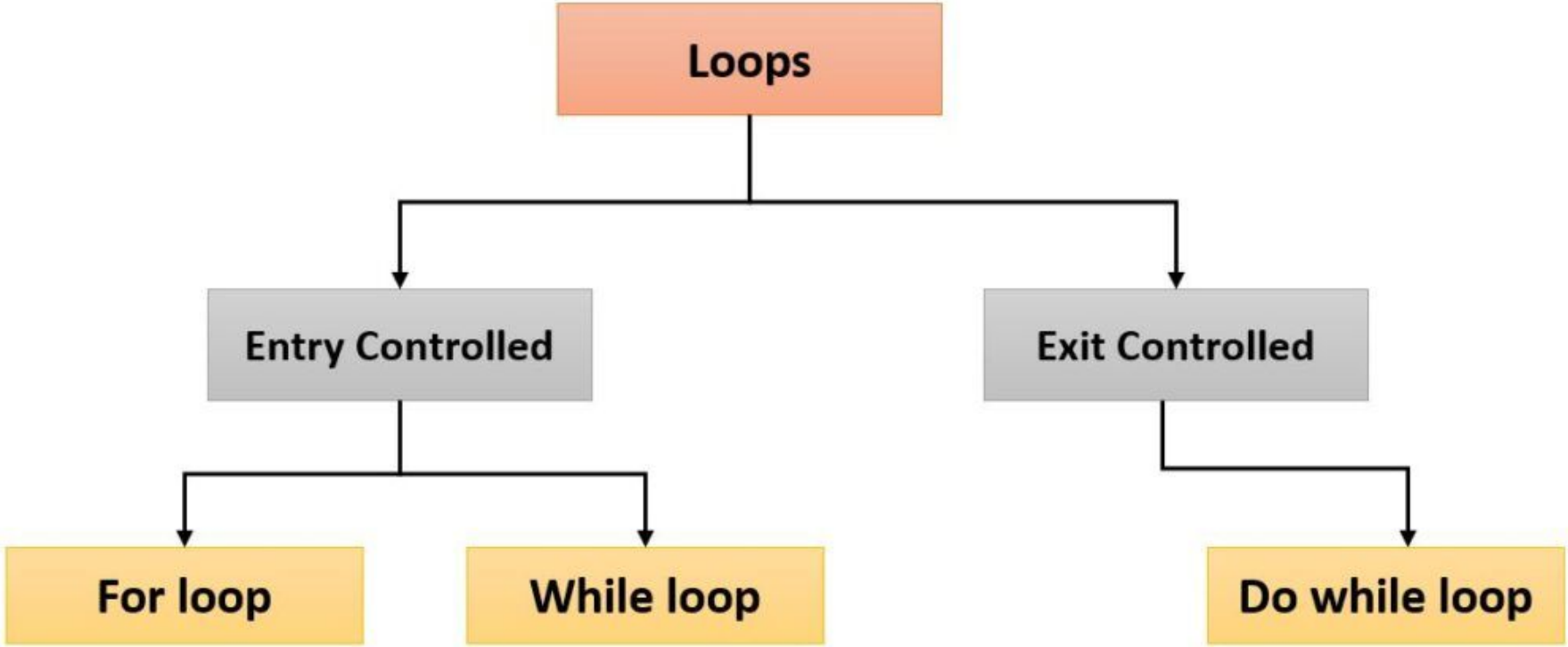
Content

1. **Introduction (Decision Making & Looping)**
2. **The FOR Statement**
3. **The WHILE Statement**
4. **The DO Statement**
5. **Jumps in Loops**
6. **Concise Test Expressions**

1. Introduction: Decision Making & Looping

- Looping allows programs to execute a block of code repeatedly. This is essential for processing collections of data, performing repetitive tasks, and creating interactive programs.
- **Why loops?**
  - Avoid writing repeated code
  - Reduce program size
  - Improve readability

1. Introduction: Decision Making & Looping



2.The FOR Statement

Compact loop structure with initialization, condition, and update all in one line.
Best for known number of iterations.

- Entry-controlled loop
- Most compact loop
- Used when number of iterations is known

Syntax:

```c
for (initialization; condition; update) {
    // code to repeat
}
```

Example

```c
#include <stdio.h>

int main() {
    int i;

    for (i = 1; i <= 5; i++) {
        printf("%d ", i);
    }
    return 0;
}
```

✔ Output:

```
1 2 3 4 5
```

3. The WHILE Statement

Executes a block of code as long as a condition is true. It's a pre-test loop (condition checked before execution).

- **Entry-controlled loop**
- Condition is checked **before** execution
- Loop may execute **zero or more times**

Syntax:

```c
while (condition) {
    // code to repeat
    // update condition variable
}
```

Example

```c
#include <stdio.h>

int main() {
    int i = 1;

    while (i <= 5) {
        printf("%d ", i);
        i++;
    }
    return 0;
}
```

✔ Output:

```
1 2 3 4 5
```

4. The DO Statement

Executes the block at least once, then repeats while condition is true. It's a post-test loop (condition checked after execution).

- **Exit-controlled loop**
- Condition is checked **after** execution
- Loop executes **at least once**

Syntax:

```c
do {
    // code to repeat
    // update condition variable
} while (condition);
```

Example

```c
#include <stdio.h>

int main() {
    int i = 1;

    do {
        printf("%d ", i);
        i++;
    } while (i <= 5);

    return 0;
}
```

✔ Output:

```
1 2 3 4 5
```

5. Jumps in Loops

Jump statements change normal loop execution. Some statements used to jump in the loop:

- breaks
- continue
- goto

These special statements to alter normal loop flow.

5.1 break statement

Terminates the loop immediately i.e. immediately exits the loop.

```c
for (int i = 1; i <= 5; i++) {
    if (i == 3)
        break;
    printf("%d ", i);
}
```

✔️ Output:

```
1 2
```

5.2 continue statement

Skips current iteration and moves to next.

```c
for (int i = 1; i <= 5; i++) {
    if (i == 3)
        continue;
    printf("%d ", i);
}
```

✔️ Output:

```
1 2 4 5
```

5.3 goto statement

Transfers control unconditionally to a labeled statement.

```
int i = 1;

start:
printf("%d ", i);
i++;

if (i <= 5)
    goto start;
```

6. Concise Test Expressions

Using **shorter forms of conditions** where non-zero means true. In that case, zero is treated as false and any non-zero value is treated as true, allowing conditions to be written in concise form.

a)

Long form

```c
while (i != 0) {
    printf("%d ", i);
    i--;
}
```

Short form

```c
while (i) {
    printf("%d ", i);
    i--;
}
```

Same meaning, shorter and cleaner.

b)

```c
if (!x)
    printf("x is zero");
```

- `!x` → TRUE when `x == 0`

c)

Using `for` without body

```c
for (i = 0; i < n; sum += i++);
```

Summary

Comparison Table: Loop Types

Loop Type	When to Use	Example Scenario	Entry Condition Check
WHILE	Unknown iterations, condition-based	User input validation	Before execution
DO-WHILE	At least one execution needed	Menu systems	After execution
FOR	Known iterations, counter-based	Array processing, patterns	Before execution

END
of
Chapter 4

***Reference**:* E. Balaguruswamy; Programming in ANSI C; *chapter-6;*