6. Looping in C

- **THEORY EXERCISE: Compare and contrast while loops, for loops, and do-while loops. Explain the scenarios in which each loop is most appropriate.**

While Loops in

- **How it works:** A while loop in C executes a block of code repeatedly as long as a given condition is true. The condition is evaluated *before* each iteration. [1]
- **Syntax:**

```
while (condition) {
   // Code to be executed as long as the condition is true
}
```

- **Key characteristic:** The loop might not execute even once if the initial condition is false. It's a **pre-test loop**.
- **Analogy:** Imagine a thermostat that keeps turning on the heater *while* the temperature is below a set point. If the room is already warm enough, the heater never starts.
- **Most appropriate scenarios in c:**
    - When the number of iterations is unknown beforehand.
    - When the loop's termination depends on a condition that changes within the loop's body (e.g., reading input until a specific value is entered, processing data until a flag is set).
    - Implementing iterative algorithms where the stopping point isn't fixed.

```
#include <stdio.h>


int main() {
   int i = 1;
   while (i <= 5) {
      printf("%d ", i);
      i++;
   }
   printf("\n");
```

```
    // Example: Reading input until the user enters -1
    int num;
    printf("Enter numbers (-1 to stop):\n");
    scanf("%d", &num);
    while (num != -1) {
        printf("You entered: %d\n", num);
        scanf("%d", &num);
    }

    return 0;
}
```

For Loops

- **How it works:** A for loop in C provides a concise way to iterate a specific number of times. It typically involves initialization, a condition check, and an update expression, all within the loop's header.
- **Syntax:**

```
for (initialization; condition; update) {
    // Code to be executed as long as the condition is true
}
```

    - initialization: Executed only once at the beginning of the loop (e.g., initializing a counter variable).
    - condition: Evaluated before each iteration. The loop continues as long as it's true.
    - update: Executed at the end of each iteration (e.g., incrementing or decrementing the counter).
- **Key characteristic:** The number of iterations is often known or can be easily determined before the loop starts. It's generally considered a **definite loop**.
- **Analogy:** Think of a baker placing exactly a dozen cookies on a tray. The action (placing a cookie) is repeated a fixed number of times (12).

- **Most appropriate scenarios in C:**
    - When you need to iterate a specific number of times.
    - When working with arrays or other data structures where you need to access elements by index.
    - Implementing algorithms that require a fixed number of steps.

```
#include <stdio.h>


int main() {
   // Looping 5 times
   for (int i = 0; i < 5; i++) {
      printf("Iteration %d\n", i);
   }


   // Iterating through an array
   int numbers[] = {10, 20, 30, 40, 50};
   int size = sizeof(numbers) / sizeof(numbers[0]);
   for (int i = 0; i < size; i++) {
      printf("Element at index %d: %d\n", i, numbers[i]);
   }


   return 0;
}
```

Do-While Loops

- **How it works:** A do-while loop in C executes a block of code at least once, and then continues to execute as long as a given condition is true. The condition is evaluated *after* each iteration.
- **Syntax:**

```
do {
   // Code to be executed
```

} while (condition); // Note the semicolon at the end

- **Key characteristic:** The loop is guaranteed to execute at least one time, even if the initial condition is false. It's a **post-test loop**.
- **Analogy:** Imagine a child who is told to eat at least one bite of their vegetables *and then* they can leave the table. The action (eating a bite) happens at least once before the permission is checked.
- **Most appropriate scenarios in C:**
    - When you need to execute a block of code once before checking a condition for further iterations.
    - Situations where you want to prompt the user for input and process it at least once.
    - Implementing menu-driven programs where you want to display the menu before the user makes a choice.

```
#include <stdio.h>


int main() {
   int choice;
   do {
      printf("\nMenu:\n");
      printf("1. Option A\n");
      printf("2. Option B\n");
      printf("3. Exit\n");
      printf("Enter your choice: ");
      scanf("%d", &choice);

      switch (choice) {
         case 1:
            printf("You selected Option A.\n");
            break;
```

```
            case 2:
                printf("You selected Option B.\n");
                break;
            case 3:
                printf("Exiting program.\n");
                break;
            default:
                printf("Invalid choice. Please try again.\n");
        }
    } while (choice != 3);

    return 0;
}
```

Comparison Table :

Feature	while Loop	for Loop	do-while Loop
Condition Check	Before each iteration	Typically before each iteration	After each iteration
Minimum Iterations	0	0 (if the condition is initially false)	1
Primary Use Case	Unknown number of iterations, condition-based termination	Known number of iterations, iterating over sequences	Guaranteed one-time execution, post-condition check
Syntax Structure	while (condition) { ... }	for (init; condition; update) { ... }	do { ... } while (condition);

Export to Sheets

In Summary:

- Choose while when you don't know how many times the loop needs to run and the termination depends on a condition.

- Choose for when you have a clear idea of the number of iterations required, often when working with sequences or ranges.
- Choose do-while when you need to ensure that the loop body executes at least once, regardless of the initial condition.