Control Structures in C

Control structures in C determine the flow of execution in a program. They enable the programmer to implement decisions, repeat code, and control the program's behavior under different conditions. In C, control structures are classified into **Sequential**, **Selection**, **Iteration**, and **Jump**.

Let's explore each control structure in more detail with theory, syntax, examples, and explanations.

---

1. Sequential Control Structure

- **Definition**: This is the simplest form of control flow where statements are executed in the sequence they appear in the program.
- **Explanation**: There is no branching or looping. Every statement is executed one after another.

Example:

```
#include <stdio.h>

int main() {
    int num1 = 10, num2 = 20;
    int sum = num1 + num2;  // This is executed sequentially.
    printf("The sum is: %d\n", sum);  // Print the result
    return 0;
}
```

- **Explanation**: The program will first compute the sum of num1 and num2, then print the sum. The execution is straightforward and in order.

---

2. Selection Control Structures

These control structures allow the program to make decisions based on conditions. Depending on the condition's result, different paths of execution are chosen.

2.1 if Statement

- **Definition**: Executes a block of code only if the specified condition is true.
- **Syntax**:

```
if (condition) {
    // Code to be executed if condition is true
}
```

Example:

```
#include <stdio.h>

int main() {
   int num = 10;
   if (num > 0) {
      printf("Number is positive\n");
   }
   return 0;
}
```

- **Explanation**: Since num is greater than 0, the condition evaluates to true, and the message "Number is positive" is printed.

2.2 if-else Statement

- **Definition**: This structure provides two alternatives: one if the condition is true and another if the condition is false.
- **Syntax**:

```
if (condition) {
   // Code executed if condition is true
} else {
   // Code executed if condition is false
}
```

Example:

```
#include <stdio.h>

int main() {
   int num = -5;
   if (num > 0) {
      printf("Positive number\n");
   } else {
      printf("Negative number\n");
   }
   return 0;
}
```

- **Explanation**: Since num is not greater than 0, the else part executes, printing "Negative number."

2.3 else-if Ladder

- **Definition**: Used when there are multiple conditions to check, and the code executes the first matching condition.
- **Syntax**:

```
if (condition1) {
   // Code if condition1 is true
} else if (condition2) {
   // Code if condition2 is true
} else {
   // Code if no conditions are true
}
```

Example:

```
#include <stdio.h>
int main() {
   int marks = 85;
   if (marks >= 90) {
      printf("Grade: A\n");
   } else if (marks >= 75) {
      printf("Grade: B\n");
   } else if (marks >= 60) {
      printf("Grade: C\n");
   } else {
      printf("Grade: F\n");
   }
   return 0;
}
```

- **Explanation**: Based on the value of marks, the corresponding grade is printed. Since marks is 85, the grade "B" will be printed.

2.4 switch Statement

- **Definition**: The switch statement is used to execute one out of many blocks of code based on the value of a variable or expression.
- **Syntax**:

```
switch (expression) {
   case value1:
      // Code if expression == value1
      break;
   case value2:
      // Code if expression == value2
      break;
   default:
      // Code if no case matches
}
```

Example:

```
#include <stdio.h>
int main() {
   int day = 3;
   switch (day) {
      case 1:
         printf("Monday\n");
         break;
      case 2:
         printf("Tuesday\n");
         break;
      case 3:
         printf("Wednesday\n");
         break;
      default:
         printf("Invalid day\n");
   }
```

```
    return 0;
}
```

- **Explanation**: The value of day is 3, so the program will print "Wednesday." If day had been something other than 1, 2, or 3, the default block would execute.

---

3. Iteration (Looping) Control Structures

These structures are used to repeat a block of code multiple times.

3.1 for Loop

- **Definition**: The for loop is used when the number of iterations is known beforehand.
- **Syntax**:

```
for (initialization; condition; increment/decrement) {
    // Code to execute
}
```

Example:

```
#include <stdio.h>
int main() {
    for (int i = 1; i <= 5; i++) {
        printf("%d\n", i);
    }
    return 0;
}
```

- **Explanation**: The loop runs 5 times, printing the numbers 1 to 5. It starts with i = 1, checks if i <= 5, and then increments i after each iteration.

3.2 while Loop

- **Definition**: The while loop repeats a block of code as long as the condition is true.
- **Syntax**:

```
while (condition) {
    // Code to execute
}
```

Example:

```
#include <stdio.h>
```

```
int main() {
    int i = 1;
    while (i <= 5) {
        printf("%d\n", i);
        i++;
    }
    return 0;
}
```

- **Explanation**: The loop will print numbers from 1 to 5. The condition i <= 5 is checked at the beginning of each iteration.

3.3 do-while Loop

- **Definition**: The do-while loop ensures that the code runs at least once, and continues to run as long as the condition is true.
- **Syntax**:

```
do {
    // Code to execute
} while (condition);
```

Example:

```
#include <stdio.h>
int main() {
    int i = 1;
    do {
        printf("%d\n", i);
        i++;
    } while (i <= 5);
    return 0;
}
```

- **Explanation**: The loop will execute at least once, printing the numbers from 1 to 5. The condition is checked after the first execution.

---

4. Jump Control Structures

These structures alter the normal flow of program execution.

4.1 break Statement

- **Definition**: The break statement is used to exit a loop or a switch statement prematurely.
- **Syntax**:

```
break;
```

Example:

```
#include <stdio.h>
int main() {
   for (int i = 1; i <= 5; i++) {
      if (i == 3) {
         break;  // Exit loop when i equals 3
      }
      printf("%d\n", i);
   }
   return 0;
}
```

- **Explanation**: The loop prints numbers 1 and 2, then exits when i == 3.

4.2 continue Statement

- **Definition**: The continue statement skips the remaining code in the current iteration and proceeds to the next iteration of the loop.
- **Syntax**:

```
continue;
```

Example:

```
#include <stdio.h>
int main() {
   for (int i = 1; i <= 5; i++) {
      if (i == 3) {
         continue;  // Skip when i equals 3
      }
      printf("%d\n", i);
```

```
    }
    return 0;
}
```

- **Explanation**: The loop will print numbers 1, 2, 4, and 5, skipping the number 3 due to the continue statement.

4.3 goto Statement

- **Definition**: The goto statement transfers control to a labeled statement. It is generally not recommended, as it can make the program difficult to read and maintain.
- **Syntax**:

```
goto label;
```

Example:

```
#include <stdio.h>
int main() {
    int num = 5;
    if (num < 10) {
        goto less_than_ten;  // Jump to label if num is less than 10
    }
    return 0;
 less_than_ten:
    printf("Number is less than ten\n");
    return 0;
}
```

- **Explanation**: When num is less than 10, the program jumps to the less_than_ten label and prints "Number is less than ten."

Summary of Control Structures

Type	Control Structure	Description	Example
Sequential	-	Executes statements one after another	int a = 5; int b = 10;
Selection	if, if-else, switch	Makes decisions based on conditions	if (num > 0)
Iteration	for, while, do-while	Repeats code based on a condition	for (int i = 0; i < 5; i++)

Type	Control Structure	Description	Example
Jump	break, continue, goto	Alters the flow of control in loops	break in loops

Control structures are essential for writing logical, efficient, and flexible C programs. They allow you to make decisions, repeat tasks, and control the program's execution dynamically