Control Structures in C

Control structures are an essential part of any programming language, allowing developers to control the flow of the program based on certain conditions. In C, control structures include decision-making statements, loops, and jump statements. These help in executing specific code blocks based on conditions or repeatedly executing a code block.

1. Introduction to Decision Control Statements

Decision control statements allow the program to make decisions based on certain conditions. These conditions evaluate to true or false, and based on the result, the program decides which block of code to execute.

2. Conditional Statements

Conditional statements in C include `if`, `if-else`, `if-else-if`, and `switch`. They are used to perform different actions based on different conditions.

2.1. `if` Statement

The `if` statement is the simplest form of decision-making statement. It executes a block of code only if the specified condition is true.

Syntax:

```
if (condition) {
    // code to execute if condition is true
}
```

Example:

Let's say you want to check if a person is old enough to vote.

```
#include <stdio.h>
int main() {
    int age = 18;

    if (age >= 18) {
        printf("You are eligible to vote.\n");
    }

    return 0;
}
```

In this example, the program checks if the age is greater than or equal to 18. If true, it prints a message stating that the person is eligible to vote.

2.2. `if-else` Statement

The `if-else` statement allows the execution of one block of code when the condition is true and another block when the condition is false.

Syntax:

```
if (condition) {
    // code to execute if condition is true
} else {
    // code to execute if condition is false
}
```

Example:

Check if a number is positive or negative.

```
#include <stdio.h>
int main() {
    int number = -5;

    if (number >= 0) {
        printf("The number is positive.\n");
    } else {
        printf("The number is negative.\n");
    }

    return 0;
}
```

2.3. `if-else-if` Ladder

The `if-else-if` ladder is used when multiple conditions need to be checked sequentially. It executes the code block of the first condition that evaluates to true.

Syntax:

```
if (condition1) {
    // code if condition1 is true
} else if (condition2) {
    // code if condition2 is true
} else if (condition3) {
    // code if condition3 is true
} else {
    // code if none of the above conditions are true
}
```

Example:

Determine a student's grade based on their marks.

```
#include <stdio.h>
int main() {
    int marks = 75;

    if (marks >= 90) {
        printf("Grade A\n");
    } else if (marks >= 80) {
        printf("Grade B\n");
    } else if (marks >= 70) {
        printf("Grade C\n");
    } else if (marks >= 60) {
        printf("Grade D\n");
    } else {
        printf("Fail\n");
    }

    return 0;
}
```

2.4. `switch` Statement

The `switch` statement is used to execute one code block out of many based on the value of a variable or expression. It's an alternative to multiple `if-else` conditions.

Syntax:

```
switch (expression) {
    case value1:
        // code to execute if expression == value1
        break;
    case value2:
        // code to execute if expression == value2
        break;
    // more cases...
    default:
        // code to execute if none of the cases match
}
```

Example:

A simple menu-driven program.

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

    printf("Select an option:\n");
    printf("1. Add\n");
    printf("2. Subtract\n");
    printf("3. Multiply\n");
    printf("4. Divide\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("You chose Addition.\n");
            break;
        case 2:
            printf("You chose Subtraction.\n");
            break;
        case 3:
            printf("You chose Multiplication.\n");
            break;
        case 4:
            printf("You chose Division.\n");
            break;
        default:
            printf("Invalid choice.\n");
    }

    return 0;
}
```

3. Iterative Statements (Loops)

Loops allow executing a block of code repeatedly as long as a specified condition is true. C provides three types of loops: `for`, `while`, and `do-while`.

3.1. `for` Loop

The `for` loop is used when the number of iterations is known in advance. It consists of three parts: initialization, condition, and increment/decrement.

Syntax:

```
for (initialization; condition; increment/decrement) {
    // code to be executed repeatedly
}
```

Example:

Print numbers from 1 to 5.

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

    return 0;
}
```

3.2. `while` Loop

The `while` loop executes a block of code as long as the specified condition is true. It's used when the number of iterations is not known beforehand.

Syntax:

```
while (condition) {
    // code to be executed repeatedly
}
```

Example:

Print numbers from 1 to 5 using a `while` loop.

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

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

    return 0;
}
```

3.3. `do-while` Loop

The `do-while` loop is similar to the `while` loop but guarantees that the code block is executed at least once, as the condition is checked after the execution of the code block.

Syntax:

```
do {
    // code to be executed
} while (condition);
```

Example:

Print numbers from 1 to 5 using a `do-while` loop.

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

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

    return 0;
}
```

4. Nested Loops

Nested loops are loops within loops. The inner loop executes completely every time the outer loop executes once.

Example:

Print a multiplication table.

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

    return 0;
}
```

5. The `break` and `continue` Statements

5.1. `break` Statement

The `break` statement is used to exit a loop or `switch` statement prematurely, before the condition fails.

Example:

Stop the loop when the value reaches 3.

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

    return 0;
}
```

5.2. `continue` Statement

The `continue` statement skips the rest of the loop's current iteration and moves to the next iteration.

Example:

Skip printing the number 3.

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

    return 0;
}
```

6. The `goto` Statement

The `goto` statement allows the program to jump to another part of the code by using a label. However, it's generally discouraged as it can make the code hard to follow and debug.

Syntax:

```
goto label;
// code to be skipped
label:
    // code to be executed after the jump
```

Example:

Using `goto` to jump to a specific part of the code.

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

    start: // label
    printf("%d\n", i);
    i++;

    if (i <= 5) {
        goto start; // jump back to the label
    }

    return 0;
}
```