Chapter 4

Control Statements

Introduction

Control statements in C programming are used to manage the flow of program execution by making decisions or repeating actions based on certain conditions. These statements enable programs to perform different tasks depending on logical conditions, control the repetition of code through loops, and alter the flow with jumps or exits. They include decision-making statements like if and switch, repetition structures such as loops, and flow-altering statements like break, continue, and goto.

Structure

The chapter covers the following topics:

* If statement
* Switch statement
* Repetition
* Break and continue
* Go to statements

Objectives

The objective of this chapter is to provide learners with a thorough understanding of control flow mechanisms in C programming. It aims to teach how to use the if and switch statements for decision-making, control the repetition of code using loops, and manage the flow within loops with break and continue statements. Additionally, the chapter will cover the goto statement, which allows for an unconditional jump in program execution. By mastering these control statements, learners will be able to write more flexible and efficient C programs that can adapt to different conditions and logic.

If statement

The if statement in C is a conditional control structure that allows the program to execute a block of code only if a specified condition evaluates to true. It enables decision-making in the code, making it possible to execute different actions based on different conditions.

Syntax

**Condition**: This is a boolean expression that evaluates to either true (non-zero) or false (zero). If the condition is true, the code block inside the curly braces is executed; if false, it is skipped.

Example:

Output:

The number is positive.

In this example, since the condition number > 0 evaluates to true, the message *The number is positive.* is printed. If the condition were false, the code inside the if block would not execute. This structure allows for dynamic and conditional execution of code based on variable states or user inputs.

Switch statement

The switch statement in C is a control structure that allows you to execute different blocks of code based on the value of a variable or expression. It provides a more concise and organized way to handle multiple conditions compared to using multiple if-else statements, especially when dealing with a single variable that can take on different constant values.

Syntax:

**Explanation**

* **Expression**: This is evaluated once and compared with the values of each case.
* **Case constant**: Each case defines a constant value to compare against the expression. If a match is found, the code block following that case is executed.
* **Break statement**: This is used to terminate the switch block. Without a break, execution will continue into the next case (this behavior is called "fall-through").
* **Default**: This is an optional case that runs if none of the case values match the expression. It is like the else part of an if-else structure.

Example:

Output:

In this example, the variable day is set to 3. The switch statement evaluates the value of the day:

* It checks each case starting from the top.
* When it finds that day equals 3, it executes the corresponding code block, printing *Wednesday*.
* The break statement then exits the switch block, preventing any further case evaluations.

The advantages of using a switch statement are as follows:

* A switch statement can be clearer and more readable than a series of if-else statements, especially when dealing with many conditions for a single variable.
* In some cases, compilers can optimize switch statements better than multiple if-else chains, potentially improving performance.
* Adding new cases to a switch statement is straightforward, making the code easier to maintain.
* The switch statement is a useful control structure for managing multiple conditional paths based on the value of a single expression, enhancing the readability and organization of your code.

Repetition

In programming, repetition (or iteration) refers to the execution of a block of code multiple times based on certain conditions. This is a fundamental concept that allows developers to automate repetitive tasks, process data in bulk, and manage various control flows in their programs. In C, there are several constructs to achieve repetition, primarily using loops.

For, while, and do-while loop

The explanation for each of these is as follows:

* **for loop**: The for loop is used when the number of iterations is known beforehand. It consists of three main components: initialization, condition, and update. The loop initializes a counter, checks a condition before each iteration, and updates the counter after each iteration.
* Syntax:

for (initialization; condition; update) {

// code to be executed

}

Example:

c

Copy code

for (int i = 0; i < 5; i++) {

printf("%d\n", i); // Prints numbers 0 to 4

}

* Best suited for situations where the iteration count is known. It offers a compact syntax combining initialization, condition checking, and counter-updating.
* **while loop:** The while loop is used when the number of iterations is not known in advance and depends on a condition. It continues to execute as long as the specified condition evaluates to true.
* Syntax:
* Example:

int i = 0; while (i < 5) {

printf("%d\n", i); // Prints numbers 0 to 4 i++; // Increment i

}

* Suitable for situations where the iteration count cannot be predetermined. The condition is checked before each iteration, which means the code may not execute at all if the condition is false initially.
* **do while loop:** The do while loop is similar to the while loop, but it guarantees that the block of code will execute at least once, as the condition is checked after the execution of the loop's body.
* Syntax:

do {

// code to be executed

} while (condition);

* Example:
* Ensures that the code block runs at least once, regardless of whether the condition is true or false initially. The condition is evaluated after the code execution, allowing for guaranteed execution before condition checking.

This table provides a quick reference to the main features and use cases of each type of loop in C, helping you choose the appropriate loop structure based on your needs (*Table 4.1*)

|  |  |  |  |
| --- | --- | --- | --- |
| **Feature** | **for loop** | **while loop** | **do while loop** |
| **Syntax** | for (initialization; condition; update) { /\*  code \*/ } | while (condition) { /\* code \*/ } | do { /\* code \*/ } while (condition); |
| **Condition**  **Check** | Before each iteration | Before each iteration | After each iteration |
| **Execution Guarantee** | May not execute if condition is false  initially | May not execute if condition is false  initially | Executes at least once |
| **Use Case** | When the number of iterations is known | When the number of  iterations is not known | When at least one execution is required |
| **Initialization** | Included in the loop  statement | Must be done before  the loop starts | Must be done before  the loop starts |
| **Update** | Included in the loop  statement | Must be done inside  the loop | Must be done inside  the loop |
| **Example** | for (int i = 0; i < 5;  i++) { printf("%d\n", i); } | int i = 0; while (i < 5)  { printf("%d\n", i); i++; } | int i = 0; do { printf("%d\n", i); i++;  } while (i < 5); |

Break and continue

In C, the break and continue statements are used to control the flow of loops. They provide a way to exit loops or skip to the next iteration, enhancing the flexibility and functionality of your loop constructs. Here is a detailed explanation of each:

* **break statement:** The break statement is used to terminate the current loop or switch statement immediately. When break is encountered, the program control jumps to the statement following the loop or switch.
* Usage:
* To exit a loop prematurely based on a condition.
* To stop execution of a switch statement after a case has been executed (if not using fall- through).
* Example:
* Output:

1

2

3

4

* In this example, the loop prints numbers from 1 to 4. When i equals 5, the break statement is executed, terminating the loop.
* **continue statement:** The continue statement is used to skip the current iteration of a loop and move to the next iteration. When continue is encountered, the remaining code inside the loop for that iteration is skipped, and the loop proceeds with the next iteration.
* Usage:
* To skip specific iterations based on a condition.
* Useful when certain conditions should prevent the execution of the remaining code in the current iteration.
* Example:
* Output:
* In this example, the loop prints only the odd numbers from 1 to 10. When i is even, the continue statement is executed, causing the loop to skip the current iteration and move to the next value of i. The break and continue statements provide powerful ways to control the flow of loops in C. By using these statements judiciously, you can create more efficient and readable code, making it easier to handle complex looping scenarios.

The summary of differences is shown in the following table:

|  |  |  |
| --- | --- | --- |
| **Feature** | **break** | **continue** |
| **Function** | Exits the loop entirely | Skips to the next iteration of the loop |
| **Usage** | Used to terminate loops or switch cases | Used to skip specific iterations |
| **Effect** | Control jumps to the statement after the  loop | Control jumps to the loop's condition  check |
| **Example** | Terminating a loop when a condition is  met | Skipping an iteration based on a  condition |

Go to statements

The goto statement in C is a control flow statement that allows you to jump to a labeled statement within the same function. While it provides a way to transfer control unconditionally to another part of the code, its use is generally discouraged because it can make the code harder to read and maintain.

**Syntax**: The syntax for the goto statement is as follows:

**Label**: A user-defined identifier followed by a colon (:) that marks a location in the code. You can use this label as a destination for the goto statement. When the goto label is encountered, the program jumps to the line marked by the label and continues execution from there.

Example: Here is a simple example demonstrating the use of the goto statement:

Output:

In this example:

* The program prints numbers from 0 to 4.
* When i equals 5, the goto end statement is executed, causing the program to jump to the end label.
* The code execution then continues from the end label; printing *Exited the loop*.

While the goto statement can be a powerful tool in specific situations, it should be used sparingly and with caution. Emphasizing structured programming principles typically leads to clearer, more maintainable code.

Conclusion

In C programming, control statements manage decision-making and repetition, enabling dynamic program flow. The if statement executes a block of code only if a specified condition is true, while the switch statement selects a code block to execute from multiple options based on a variable’s value. Repetition or looping structures, for, while, and do-while loops, allow code to execute repeatedly based on a condition. for loops are typically used when the number of iterations is known, while loops execute as long as a condition remains true, and do-while loops guarantee at least one iteration since the condition is checked after the loop body. Control keywords like break and continue to modify loop execution by exiting or skipping iterations, and goto provides a way to jump directly to another code section, though it i generally avoided for readability and structure. The next chapter will delve into functions in C programming, which are essential for organizing and modularizing code. It will cover the definition and structure of functions, along with the concept of function prototypes and various parameter passing techniques, including pass-by-value and pass-by-reference. The chapter will also explore recursion, a powerful technique where a function calls itself to solve problems. Additionally, the chapter will introduce built-in functions, which are pre-defined in C for performing common tasks, allowing for more efficient programming by leveraging these ready-made functionalities.

Exercises

* Write a program that takes an integer input and uses an if statement to check if the number is positive, negative, or zero. Display an appropriate message for each case.
* Create a program that takes a day number (1-7) and uses a switch statement to print the corresponding day of the week.
* Write a program that uses a for loop to calculate the sum of the first 10 natural numbers and display the result.
* Develop a program that uses a while loop to print all even numbers between 1 and 20.
* Create a program that uses a do-while loop to prompt the user to enter a positive integer, repeating the prompt until the user enters a positive number.
* Write a program that uses a for loop to print numbers from 1 to 10, but exits the loop early if the current number is 5.
* Create a program that uses a for loop to print numbers from 1 to 10 but skips printing the number 5 using the continue statement.
* Write a program that accepts an integer and uses nested if statements to determine if the number is both positive and even, displaying appropriate messages.
* Create a simple program using goto that jumps to a label to re-run a calculation based on user input, such as a multiplication of two numbers.
* Write a program that calculates the factorial of a number using both a for loop and a while loop, then compare their implementations.