# Chapter 8. Cyclic algorithms

## 8.1 Types of cyclic processes

In programming, a loop is a construct that allows you to execute the same set of instructions repeatedly. This is especially useful when you need to process large amounts of data or repeat certain operations.

The first type of cyclic process is an arithmetic cycle. It is used when it is known exactly how many times a cyclic process needs to be performed. For example, you can perform the operation 10 times or in the range from 1 to 100.

The second type of cyclic process is an iterative cycle. It is executed when the number of cyclic repetitions is unknown, but the condition for ending iterations is known, and it is the only one. For example, a loop can continue until the value of a variable reaches a certain limit.

The third type of cyclic process is the search cycle. It is used when it is not known how many times a cyclic process needs to be executed and there is more than one reason for it to end.

## 8.2 Operators of cyclic processes

A precondition loop checks conditions before executing each iteration of the loop. If the condition is true, the body of the loop is executed. If the condition is false, the loop ends and control passes to the next parts of the program. Thus, the execution of the loop depends on the truth of the condition.

In the C++ language, the while operator is used to implement cyclic processes with a precondition. This loop checks the conditions before each iteration and continues as long as the condition remains true. The syntax and block diagram of the algorithm with the while precondition are presented in figure 8.1.

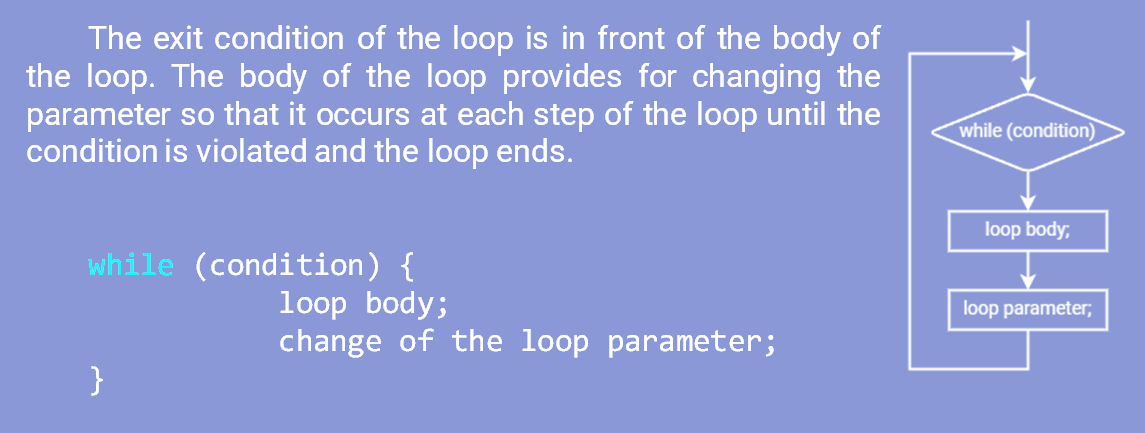


Figure 8.1 – Flowchart of a loop algorithm with a while precondition

A post condition loop checks the condition after each iteration of the loop. That is, the body of the loop is executed at least once, and then a condition is checked to continue executing the loop. If the condition is true, the loop continues. If the condition is false, the loop ends and control passes to the next parts of the program.

In C++, the do while operator is used to implement cyclic processes with a post condition. Unlike the while loop, this loop performs the operations first and then tests the condition. This way the operation is performed at least once. Syntax and block diagram of the algorithm with the do while post condition (figure 8.2).

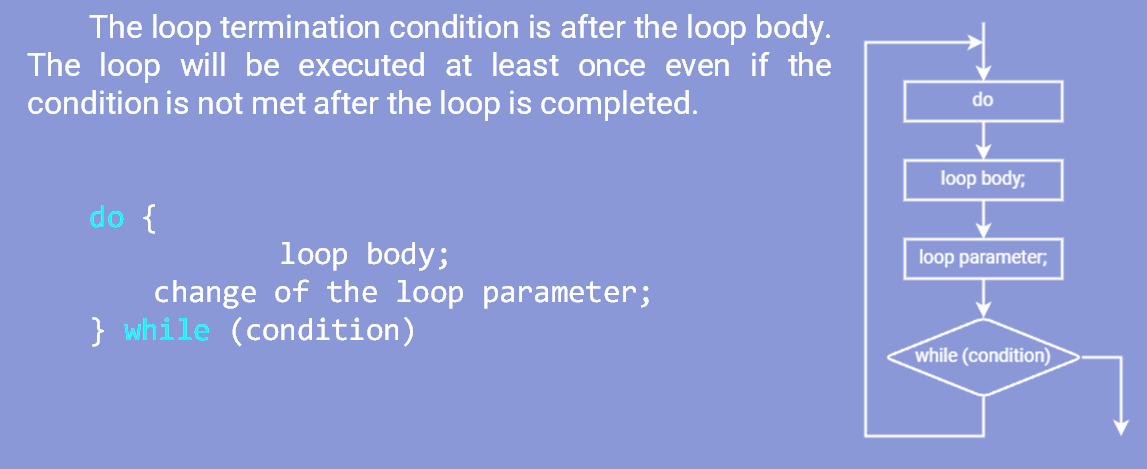


Figure 8.2 – Block diagram of the loop algorithm with the do while post condition

An auto-parameter loop uses an arithmetic change in the value of a counter variable or loop parameter at each iteration.

In the C++ language, the for operator is used to implement cyclic processes with automatic parameter changes. It has a special structure consisting of an initialization, a condition, and an update expression. This loop is usually used in cases where the number of iterations is known. The syntax and block diagram of the algorithm with automatic change of the for parameter are presented in figure 8.3.

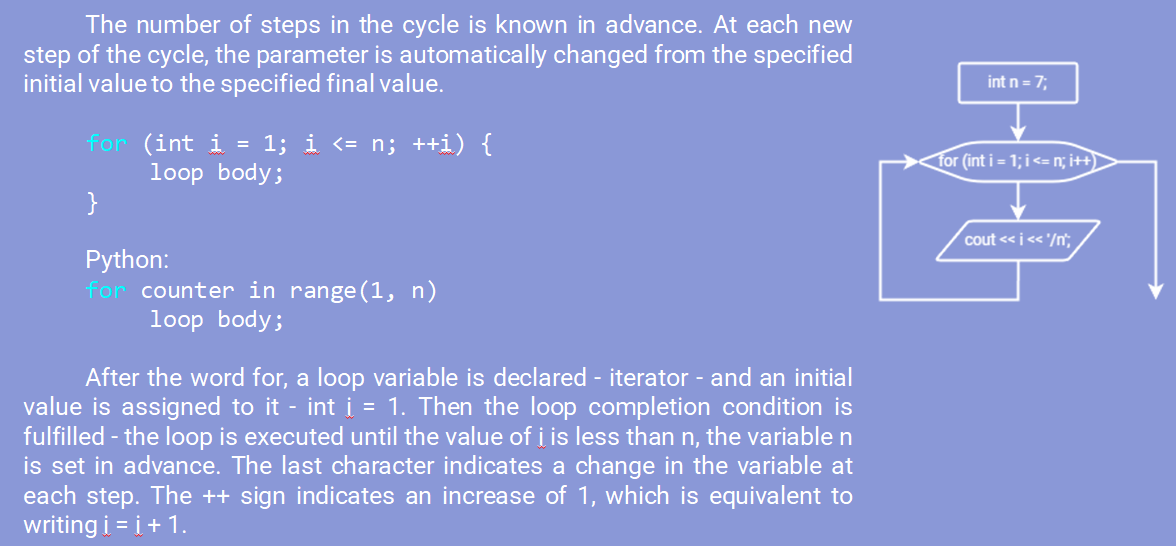


Figure 8.3 – Block diagram of the loop algorithm with automatic change of the for parameter

## 8.3 Break statement

An important operator for cyclic processes is the break operator. It allows you to exit the loop even if the loop condition is still true. This can be useful when you want to terminate a loop early based on certain conditions. An example of program code with a break statement is shown in figure 8.4.

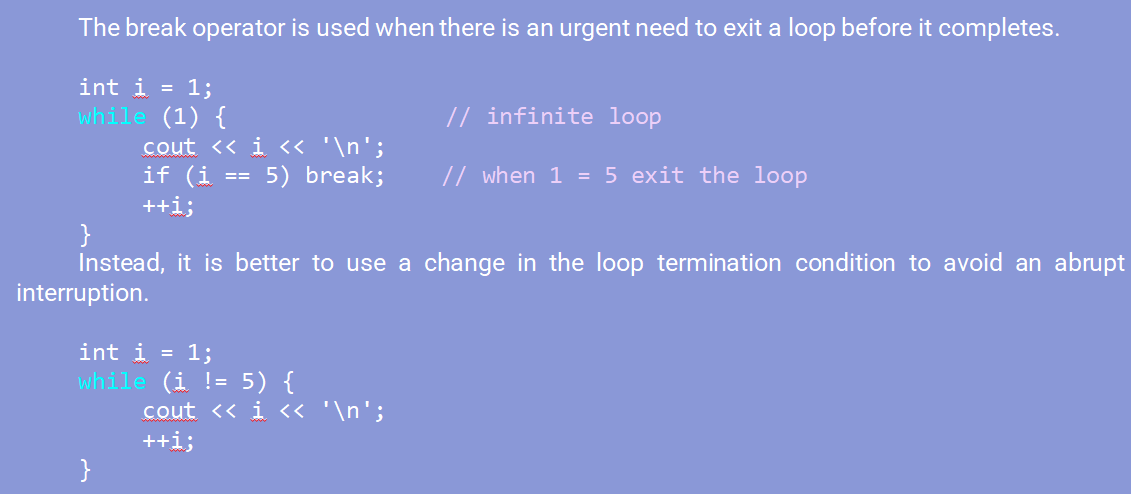


Figure 8.4 – Example program code with the break statement

## 8.4 Continue statement

The continue statement allows you to skip the current iteration of the loop and move on to the next one. This can be useful when you want to skip certain operations in a loop based on certain conditions. An example of program code with the continue statement is shown in figure 8.5.

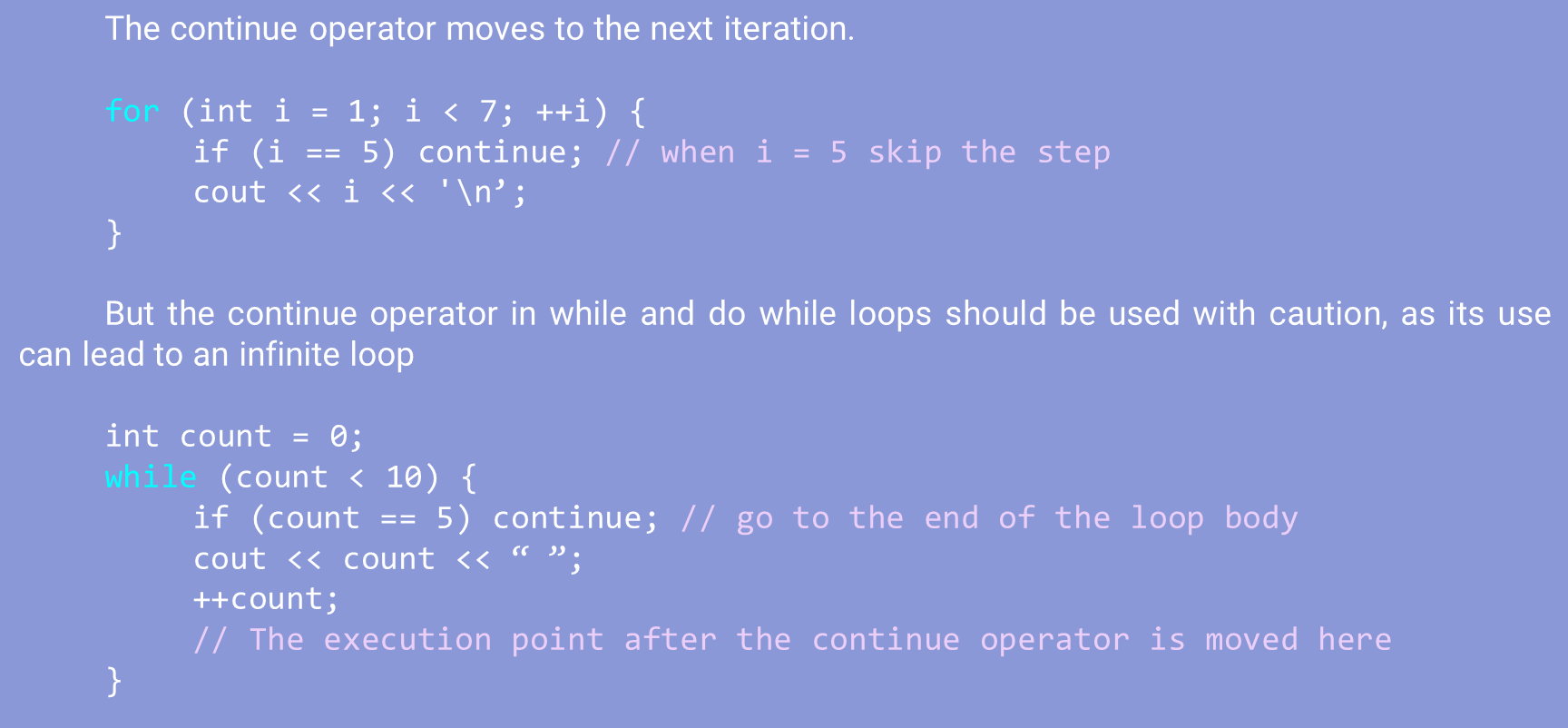


Figure 8.5 – Example program code with the continue statement

## 8.5 Nested loops

There are two types of nested loops:

1. Outer Loop: There is only one outer loop.
2. Inner loop: There can be several inner loops. It is important to note that an internal cycle can become an external cycle if its body is also a cyclical process.

Both loops have a loop parameter that changes its value at each step of the loop. Cyclic processes complete their execution when the corresponding parameter reaches a given limit [28].

Nested loops interact as follows:

1. The loop variable takes a value and is checked to see if it exceeds the set limit. If the limit is not exceeded, execution of the inner loop begins.
2. If there is a nested loop in the loop body, its parameter must pass through all available values.
3. After the inner loop completes, the outer loop parameter is changed and the process repeats from step 1.

An example of such an algorithm is presented in figure 8.6.

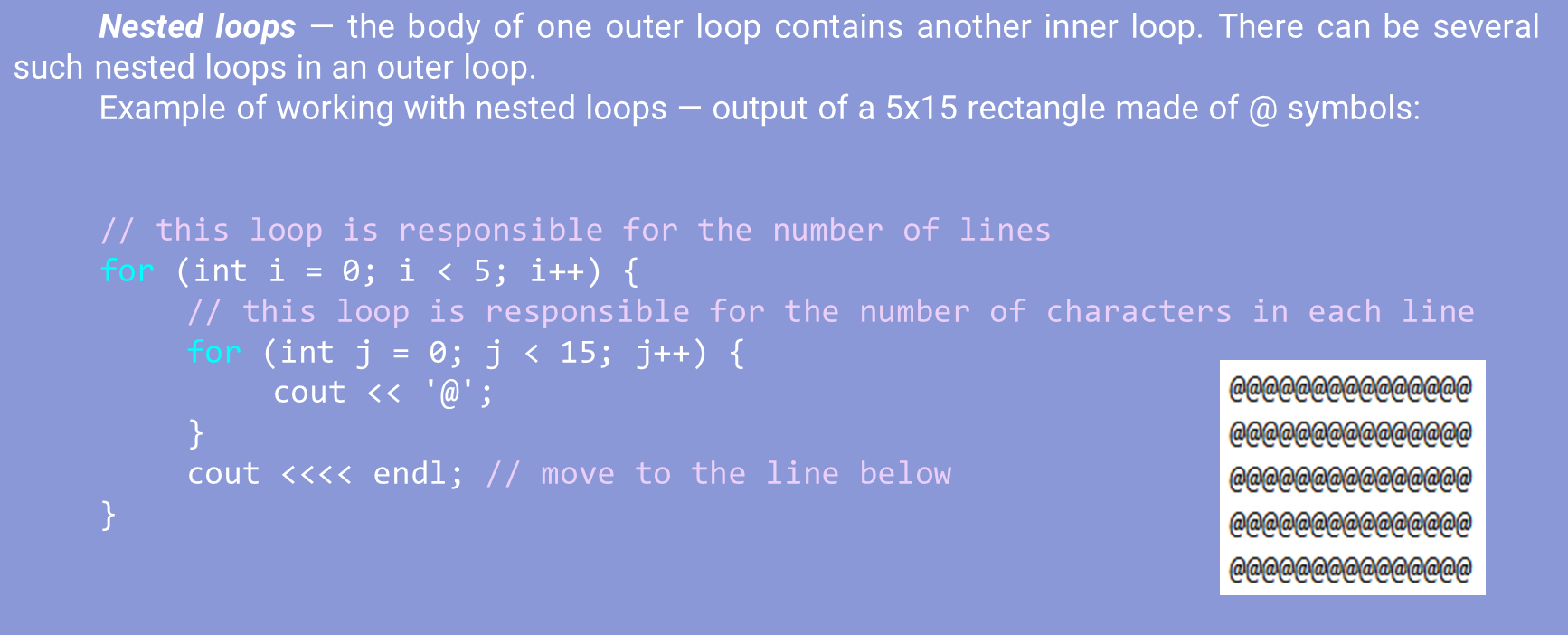


Figure 8.6 – Example of nested loops