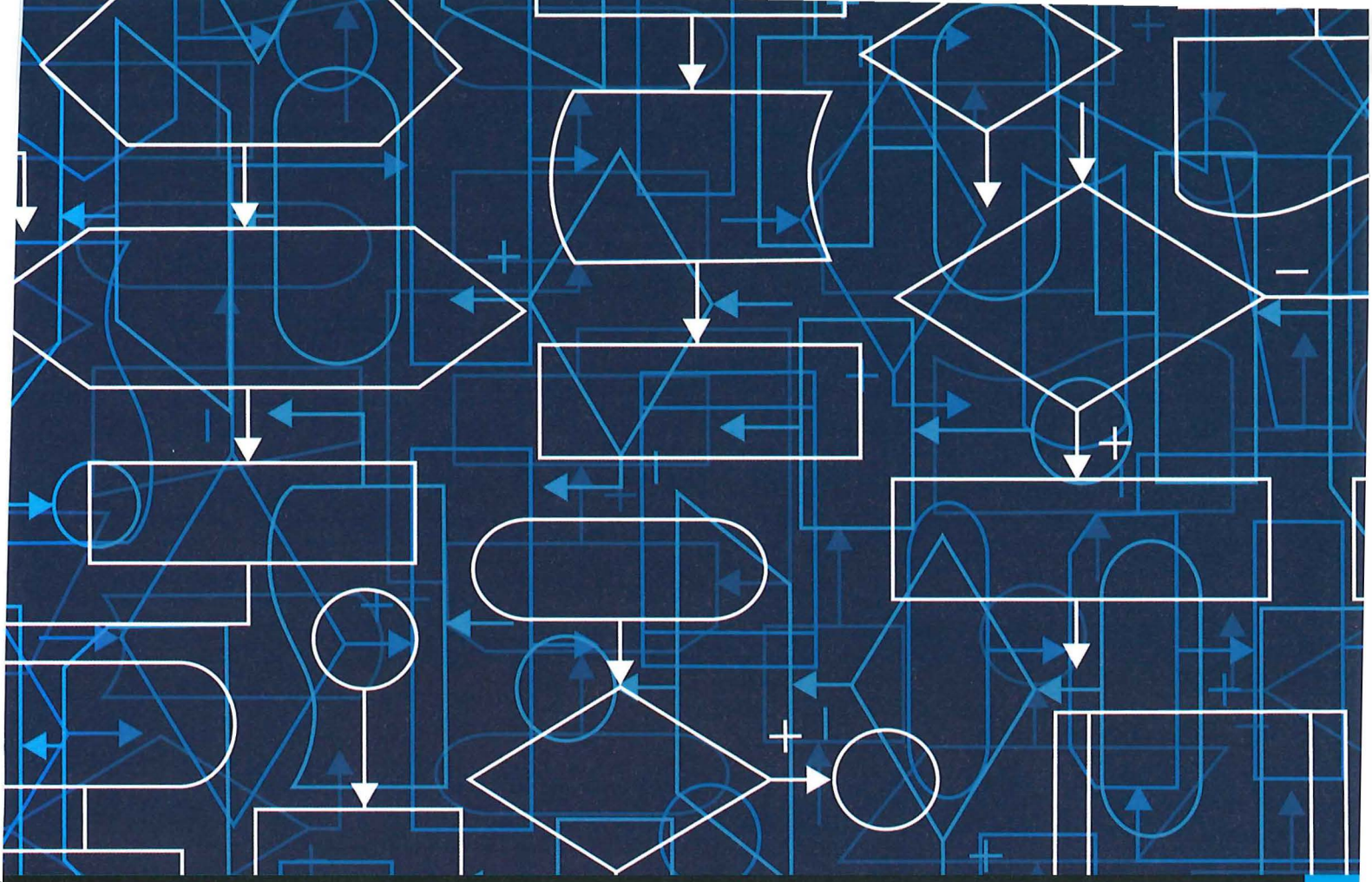
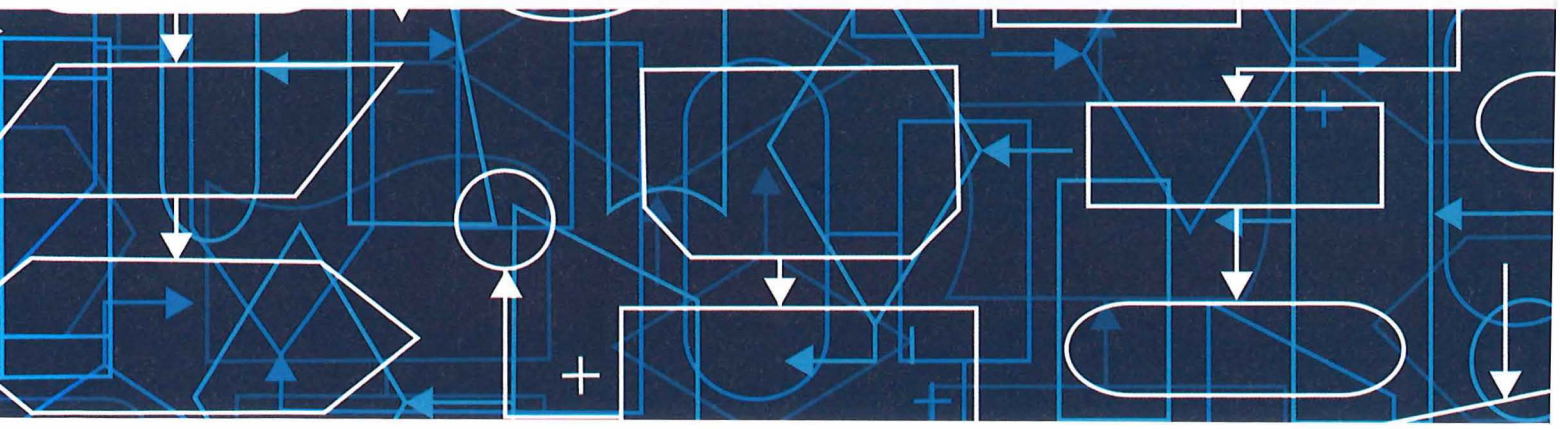


Chapter 12:
Pseudocode and Flowcharts

Learning objectives

By the end of this chapter you will be able to:

- use pseudocode and flowcharts to assign a value to a variable
- use pseudocode and flowcharts to create conditional statements
- use pseudocode and flowcharts to create loop structures
- use pseudocode and flowcharts to input and output values
- use pseudocode and flowcharts to total and count values.



12.01 Introduction

Pseudocode and flowcharts are tools that a programmer may use to help design a program, or understand one that already exists. This chapter examines how we can use them.

Many of the programming concepts discussed in this chapter are described in more detail in Chapter 11.

12.02 Pseudocode

Pseudocode is a way of describing what happens in a computer program. It is a list of instructions that show how the program will work. Pseudocode is set out with the same structure as a programming language, but it is not a programming language in itself. Pseudocode can be used to show people who do not know a particular programming language how the program works. Additionally, programmers can use it to plan and design programs.

KEY TERM

Syntax – the rules of a language.

There is not a standard **syntax** for pseudocode, but there are some commonly followed conventions. This means that pseudocode written by one programmer can be easily understood by another.

Pseudocode covers a range of programming elements:

- comments
- variables
- arrays
- selection
- iteration
- input and output
- procedures and functions.

Comments

In pseudocode comments are written to state what a particular line or section of code is for. A commented line starts with a hash symbol:

```
# this is a comment!
# this variable holds the player's score
```

It is just as important to comment our code when writing pseudocode as it is when writing real code. Without comments, any code can be difficult to understand. We should be aware of writing too few or too many comments in our code. Too few comments can make it harder to understand a program. Too many comments can make it difficult for us to find what we are looking for in a program.

Variables

SYLLABUS CHECK

Understand and use pseudocode for assignment, using ←.

A **variable** is a named location in memory that holds a data value. The data held in a variable may change repeatedly throughout the running of a program. In pseudocode variables are shown using the format in Figure 12.01.

KEY TERM

Variable – a named location in memory that holds a data value.

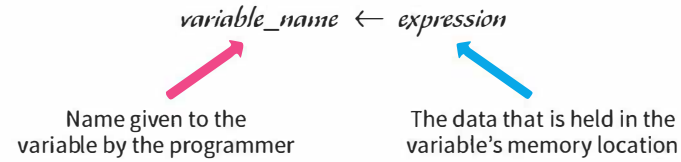


Figure 12.01 Variables shown in pseudocode

We can assign a value to the variable:

```
score ← 0        # create a variable named score, assigned the value 0
```

A variable's data type is not usually specified in pseudocode. If required, a comment can be included stating the type to be assigned.

To learn more about variables, see Chapter 11.

Arrays

An **array** is a data structure that holds a collection of values of the same data type. In pseudocode an array is declared by stating the array name. Data can be assigned by placing the values in between parentheses []. An array can also be left blank so data is assigned during the program.

KEY TERM

Array – a data structure that holds a collection of values of the same data type.

```
highscores ← [ ]                    # create an empty array
playernames ← [Sarah, Vikram, Amy]  # create an array with 3 values assigned
```

The arrays above are one-dimensional arrays. This type of array holds one set of related values.

Programmers can also use two-dimensional arrays. This type of array holds two separate sets of related values. In pseudocode two-dimensional arrays are declared by using two sets of parentheses, one for each set of data:

```
# create an array with two data sets
namesandscores ← [Sarah, Vikram, Amy] [100, 88, 72]
```

Multi-dimensional arrays are not part of the syllabus that you need to know. This information has been included to help you to understand the concept of an array and that they can be expanded to hold more than one set of data.

To learn more about arrays, see Chapter 11.

Counting

SYLLABUS CHECK

Use the following commands and statements: counting (e.g. Count ← Count + 1).

Sometimes in a program we need to keep a record of how many times something has happened. The method used to do this is called a count. A variable is used to hold the value of the current count. A count may be added to or subtracted from. In pseudocode a count is done using the format:

```
count ← count + 1          # add 1 to the value of count
count ← count – 1          # subtracts 1 from the value of count
count ← count + number     # add the value of a variable named number to the value of count
```

If we begin our count at 0 we would start by assigning a variable named count with the value 0:

```
count ← 0
```

We might want to keep a record of how many times we run a particular set of instruction in a program. Within this set of instructions we would have the line of code:

```
count ← count + 1
```

This would take the value stored in count (initially 0) and add 1 to it. The value stored in count would then become 1. The next time the set of instructions is run it would take the value stored in count (now 1) and add 1 to it. The value then stored in count would be 2.

Totalling

SYLLABUS CHECK

Use the following commands and statements: totalling (SUM ← SUM + Number).

Totalling is the process of keeping a running total of certain values in a program. A variable is used to hold the value of the total. In pseudocode totalling is done using the format:

```
total ← total + number     # the total is incremented by the value of the number
```

We would start by assigning a variable named total with the value 0:

```
total ← 0
```

We may want to keep a running total for an item in a program. We would add a related value to the total whenever it is input:

```
number ← 5
total ← total + number
```

This would take the value held in the variable with the name 'total' (initially 0) and add to it the value that is held in the variable with the name 'number'. The value held in total will then become 5.

At a further point in the program another value needs to be added to the total:

```
secondnumber ← 4
total ← total + secondnumber
```

This would take the value held in total (now 5) and add to it the value that is held in secondnumber (this is 4). The value held in total will then become 9.

To learn more about counting and totalling, see Chapter 11.

Selection

SYLLABUS CHECK

Use the following conditional statements:

- IF ... THEN ... ELSE ... ENDIF
- CASE ... OF ... OTHERWISE ... ENDCASE.

Selection in a program allows a program to follow different paths. The choice of path is determined using **conditions**. A path is followed depending on the whether a particular condition is met or not. For example, a message of congratulations might be output if an exam score was high enough to pass. Another message to 'try again' might be output if the score was too low. Selection can be described in pseudocode using IF statements and CASE statements.

KEY TERM

Condition – a state in a program that will be met or not met.

Selection – a statement that represents choice in an algorithm.

IF statements

IF statements allow a programmer to define several paths through a program. A Condition will determine which path is followed. IF statements take the form IF ... THEN ... ELSE ... ENDIF:

The exam score selection example described above could be written in pseudocode as:

```
IF examscore > 50 THEN
    PRINT "Congratulations! You passed"
ELSE
    PRINT "Sorry, you did not pass. Try again"
ENDIF
```

This means that if the value of examscore is greater than 50, the congratulations message is output. For any exam score less than or equal to 50, the 'try again' message would be output.

This example could be extended to include an extra message for especially high marks:

```
IF examscore > 90 THEN
    PRINT "Congratulations! You passed with distinction!"
ELSE
    IF examscore > 50 THEN
        PRINT "Congratulations! You passed!"
    ELSE
        PRINT "Sorry, you did not pass. Try again"
    ENDIF
ENDIF
```


CASE statements

When many possible paths exist, using IF statements can make code complicated. Another selection method that can be used is the CASE statement. CASE is used when specific discrete data values are used, not for those that might be in a range. For example:

- If examscore can be one of 1, 2, 3, 4 or 5 use CASE because specific values are considered.
- If examscore is greater than 50 use IF because a large possible range of values are considered.

When dealing with many paths with specific condition values, using CASE allows for simpler, easier to read code. In pseudocode, CASE statements take the form CASE ... OF ... OTHERWISE ... ENDCASE:

```
CASE examscore OF
1: PRINT "Sorry, you did not pass"
2: PRINT "Sorry you did not pass, but you were close"
3: PRINT "You passed, but only just"
4: PRINT "You passed with a decent score!"
5: PRINT "You passed with distinction!"
OTHERWISE
    PRINT "Your score was not recognised"
ENDCASE
```

To learn more about selection, see Chapter 11.

Iteration

SYLLABUS CHECK

Use the following loop structures:

- FOR ... TO ... NEXT
- REPEAT ... UNTIL
- WHILE ... DO ... ENDWHILE.

Sometimes a programmer may need to make a section of code run repeatedly. For example, if we need to enter a series of numbers, the input code will need to be repeated until all numbers have been entered. **Iteration** can be implemented using three types of loop:

- **count-controlled loops** using FOR
- **condition-controlled loops** using WHILE
- **condition-controlled loops** using REPEAT.

KEY TERM

Iteration – where a section of code is run repeatedly.
Count-controlled loop – iteration that is performed a stated number of times.
Condition-controlled loop – iteration that continues until, or while, a condition is met.

FOR loops

A FOR loop is used when the loop is to be repeated a fixed number of times. As the number of times is fixed, we say the loop is 'count-controlled'. We need to increase a counter each time the loop is performed. The loop ends when the counter matches the number of times the loop is to be repeated.

In pseudocode a count-controlled loop takes the form FOR ... TO ... NEXT:

```
FOR count ← 0 TO 4:
    PRINT "Type in a number"
    number ← USERINPUT
    total ← total + number
NEXT count
```

This loop will iterate five times. This is because the first value of count is 0 and the last value of count is 4, giving five iterations. This will allow the user to enter five numbers.

TEST YOURSELF

What else will the program do?

WHILE loops

A WHILE loop is used when the programmer does not know exactly how many times the loop will need to be performed. The loop continues while a condition is being met. In pseudocode a WHILE loop takes the form WHILE ... DO ... ENDWHILE:


```
WHILE hungry = TRUE DO
    PRINT "I'm hungry"
    hungry ← USERINPUT
ENDWHILE
```

It is possible that the code in a WHILE loop is never run. This is because the condition is tested at the beginning of the loop. If the condition is not met then the loop will not run and the program will continue.

TEST YOURSELF

What will this program do?

REPEAT loops

A REPEAT loop can also be used when the programmer does not know exactly how many times the loop will need to be performed. The loop ends when a condition is met. In pseudocode a REPEAT loop takes the form REPEAT ... UNTIL:

```
REPEAT
    PRINT "You are hungry!"
    hungry ←USERINPUT
UNTIL hungry = FALSE
```

This loop repeats until the user says they are not hungry (the input is FALSE).

Code in a REPEAT loop is always run at least once. This is because the condition is not tested until the code in the loop has run the first time.

TEST YOURSELF

If a teacher has a set of 10 exam marks to enter, which type of loop should they use? Explain why.

To learn more about iteration, see Chapter 11.

Input and output

SYLLABUS CHECK

Use the following commands and statements: INPUT and OUTPUT (e.g. READ and PRINT).

Most programs will need to allow a user to enter data and will also have to present data to the user. This is known as input and output. Data may also be input from or output to a file.

USERINPUT

An input from a user in pseudocode takes the form USERINPUT:

```
studentname ← USERINPUT
```

This instruction waits for the user to type something in and then assigns that data to the named variable.

OUTPUT

An output to a user in pseudocode takes the form PRINT:

```
PRINT "Hello!"        # outputs a string (the text Hello)
PRINT highscore       # outputs a variable's value
```

READLINE

Data can also be input from a file. When using pseudocode, the programmer must remember to specify the filename. Reading from a file is described in pseudocode as:

```
name ← READLINE(filename, record number)
```

This would read the data from whatever filename and record number was stated and hold it in the variable 'name'.

We have a file called computer_science_students.txt that holds the names of a class of students who study Computer Science. The data in the file is shown in Table 12.01.

Table 12.01

Record number	Student name
0	Amy
1	Jan
2	Bosire
3	Amaan
4	Seema

The pseudocode to read the third record into a variable would be:

```
name ← READLINE(computer_science_students.txt, 2)
```

The variable 'name' would now hold the value 'Bosire'. The number 2 is used for the READLINE command because the record count starts at 0. This means the first name in the file is 0, the second is 1, the third is 2, etc.

WRITELINE

Data can also be output to a file. WRITELINE overwrites a line in a file. If no line exists, a new one is created. As with READLINE, the programmer must remember to specify the filename. Writing to a file is described in pseudocode as:

```
WRITELINE (filename, record number, value)
```

We can add a new student to the data in Table 12.01:

```
new _student ← "Francesca"
WRITELINE(computing_students.txt, 5, new_student)
```

This would result in the data file now having the contents shown in Table 12.02.

Table 12.02

Record number	Student name
0	Amy
1	Jan
2	Bosire
3	Amaan
4	Seema
5	Francesca

WRITELINE can also be used to remove a value:

```
WRITELINE(computing_students.txt, 3, "")
```

The contents of the file would now be as shown in Table 12.03.

Table 12.03

Record number	Student name
0	Amy
1	Jan
2	Bosire
3	
4	Seema
5	Francesca

Procedures

Sometimes sections of code are used more than once in a program. When this occurs, procedures and functions should be used. This is to avoid the same set of code being typed repeatedly.

KEY TERM

Procedure – a small section of code that can be run repeatedly from different parts of the program.

Function – a procedure that returns a value.

Procedures are sections of code that can be reused. They do not return a value. In pseudocode, a procedure is named and takes the form PROCEDURE ... ENDPROCEDURE:

```
PROCEDURE Menu
    PRINT "1. Continue"
    PRINT "2. Quit"
ENDPROCEDURE
```

In pseudocode, procedures are run using the CALL statement:

```
CALL Menu
```

TEST YOURSELF

What would this procedure do? Why might we need to repeat those instructions in a program?

Functions

Functions are similar to procedures. The difference is that functions have one or more values put into them and one or more values are given back to main program at the end. We call this 'passing values' in and out of the function.

In pseudocode, a function takes the following form:

```
FUNCTION(values to be passed in)
    ...
    RETURN (values to be passed out)
ENDFUNCTION
```

The following function performs a calculation on a number and returns the answer:

```
FUNCTION Fahrenheit_to_celsius (fahrenheit):
    celsius <-- ((fahrenheit -32) * (5/9))
    RETURN Celsius
ENDFUNCTION
```


In pseudocode, functions are run using the CALL statement:

```
fahrenheit ← 32
CALL Fahrenheit_to_celsius (fahrenheit)
```

TEST YOURSELF

What will this function do?

To learn more about procedures and functions, see Chapter 11.

12.03 Flowcharts

Pseudocode is a text-based method of designing programs. Flowcharts are a graphical method of designing programs. They allow the designer to see a visual representation of a problem. They are also useful in understanding the logic of complicated problems.

Flowcharts are usually drawn using some standard symbols, shown in Table 12.04.

Table 12.04

Purpose	Symbol
Start and end of the program	Start Stop
Computational steps or processing function of a program	
Input or output (of data)	
Decision making and branching	
Direction of flow	

Flowcharting guidelines

The following are some guidelines for flowcharting:

- All necessary requirements should be listed in logical order.
- The flowchart should be clear, neat and easy to follow.
- There should not be any room for uncertainty in understanding the flowchart.
- The usual direction of the flow of a procedure or system is from left to right or top to bottom.
- Only one flow line should come from a symbol, unless it is a decision symbol.

Figure 12.02 Computational steps

- Only one flow line should enter a decision symbol, but two or three flow lines, one for each possible answer, may leave the decision symbol.

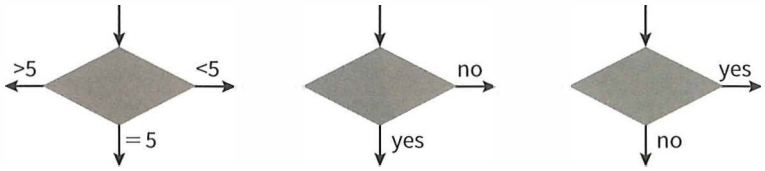


Figure 12.03 Decision symbol with a single flow line entering and two or three leaving the symbol

- Only one flow line is used for a terminal symbol. Some flowcharts may spread over many pages, making it essential to know where a process begins and ends.

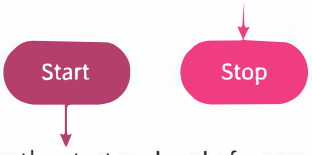


Figure 12.04 Terminal symbols for the start and end of a program

- Ensure that the flowchart has a logical start and finish.
- It is useful to test if a flowchart is correct by working through it with simple test data.

Writing a flowchart for an algorithm

SYLLABUS CHECK

Produce an algorithm for a given problem (either in the form of pseudocode or flowchart).

Draw a flowchart to find the average of two numbers.

Algorithm:

Input: two numbers, aNumber and secNumber

Output: the average of aNumber and secNumber


```
INPUT aNumber
INPUT secNumber
total ← aNumber + secNumber
ave ← total /2
OUTPUT ave
```

The flowchart for this algorithm is shown in Figure 12.05.

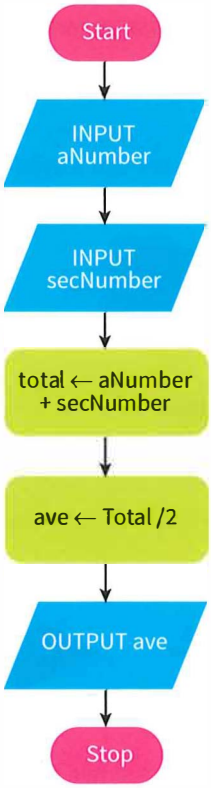


Figure 12.05 Complete flowchart for an algorithm

Selection in flowcharts

Selection using IF statements is represented in flowcharts using a decision box.

Algorithm:

Input: number

Decision: Is the number positive or negative?

Output: Number is positive, Number is negative

```
INPUT number
DECISION Is number greater than 0?
IF YES OUTPUT Number is positive
IF NO OUTPUT Number is negative
```

The flowchart for this algorithm is shown in Figure 12.06.

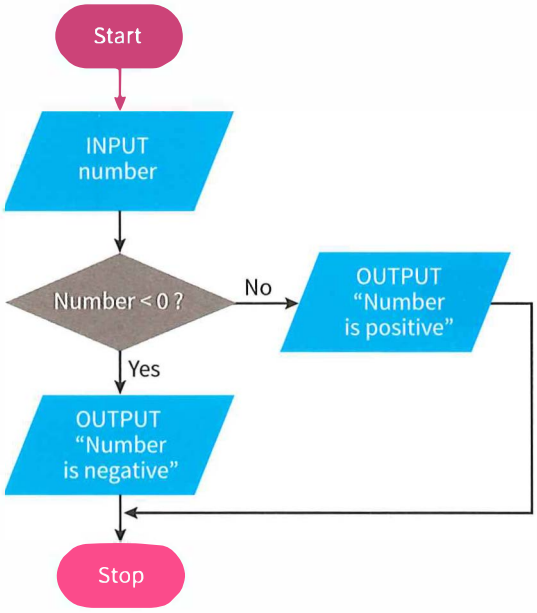


Figure 12.06 IF statement represented in a flowchart

CASE statements can also be represented. In practice CASE statements are represented in flowcharts in the same manner as IF statements.

Algorithm:

Input: number for animal

Output: dog, cat, horse

```
INPUT numAnimal
CASE numAnimal OF
1: OUTPUT dog
2: OUTPUT cat
3: OUTPUT horse
OTHERWISE
    OUTPUT Out of range
```

The flowchart for this algorithm is shown in Figure 12.07.

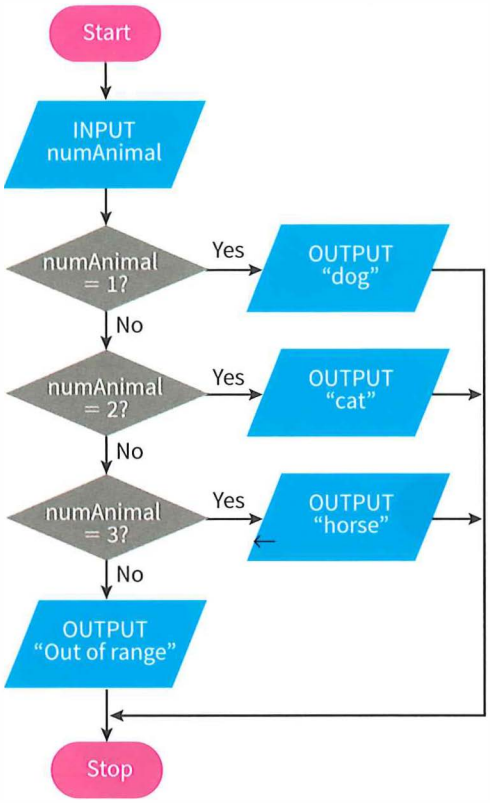


Figure 12.07 CASE statement represented in a flowchart

Iteration in flowcharts

Each of the types of iteration can be drawn in a flowchart.

Figure 12.08 shows a FOR loop.

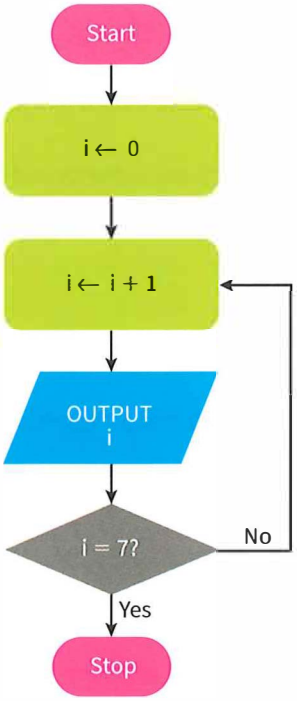


Figure 12.08 FOR loop represented in a flowchart

TEST YOURSELF

What does the program shown in Figure 12.08 do?

Figure 12.09 shows an WHILE loop. The symbol ! can be used to say that a value is not equal to something.

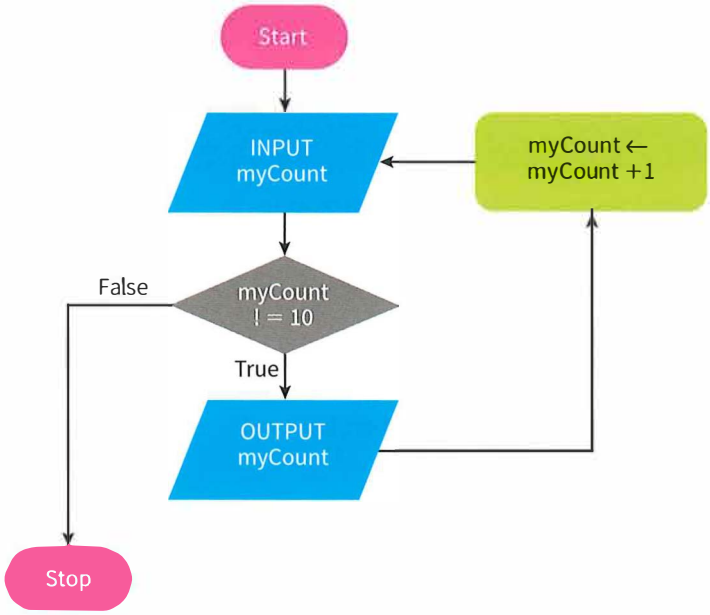


Figure 12.09 WHILE loop represented in a flowchart

TEST YOURSELF

What will the program shown in Figure 12.09 do?

Figure 12.10 shows a REPEAT loop.

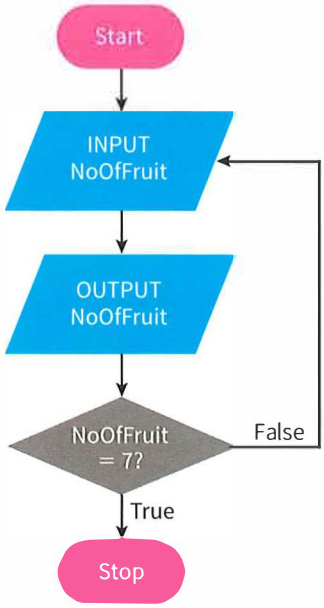


Figure 12.10 REPEAT loop represented in a flowchart

TEST YOURSELF

Design a flowchart algorithm for finding the area of a rectangle.

Hints:

- Define the inputs and the outputs.
- Define the steps.
- Draw the flowchart.

Summary

- Pseudocode and flowcharts are tools that a programmer may use to help design a program, or understand one that already exists.
- Pseudocode is a text based list of instructions that specify how a program is to work.
- Pseudocode is set out with the same structure as a programming language.
- Programmers can use Pseudocode to plan and design programs.
- Pseudocode equivalents exist for comments, variables, arrays, selection, iteration, input and output, functions and procedures.
- Flowcharts are a graphical method of designing algorithms.
- Flowcharts allow the designer to see a visual representation of a problem.
- Flowcharts are useful in understanding the logic of complicated problems.
- Flowcharts use symbols to represent processes, selection, iteration, input and output.

Exam-style questions

1 Explain why comments should be used in pseudocode. (2 marks)

2 Explain why flowcharts are a useful design tool. (2 marks)

3 What symbol is used in a flowchart to represent a decision? (1 mark)

4 Design a pseudocode algorithm that will calculate the area and perimeter of a triangle. The user must choose which answer they want to be output. (5 marks)

5 Design a flowchart algorithm that makes sure a user only enters an integer in the range 1 to 10. (4 marks)

6 Design a flowchart algorithm for a simple drinks machine. The user can choose from tea, coffee, hot chocolate, orange juice and apple juice. If the user inputs a drink that is not listed an error message should be displayed. (4 marks)

7 Amy has been asked to design an algorithm for a very large program. Which design tool, pseudocode or flowchart, do you think would be most suitable? Explain your answer. (3 marks)