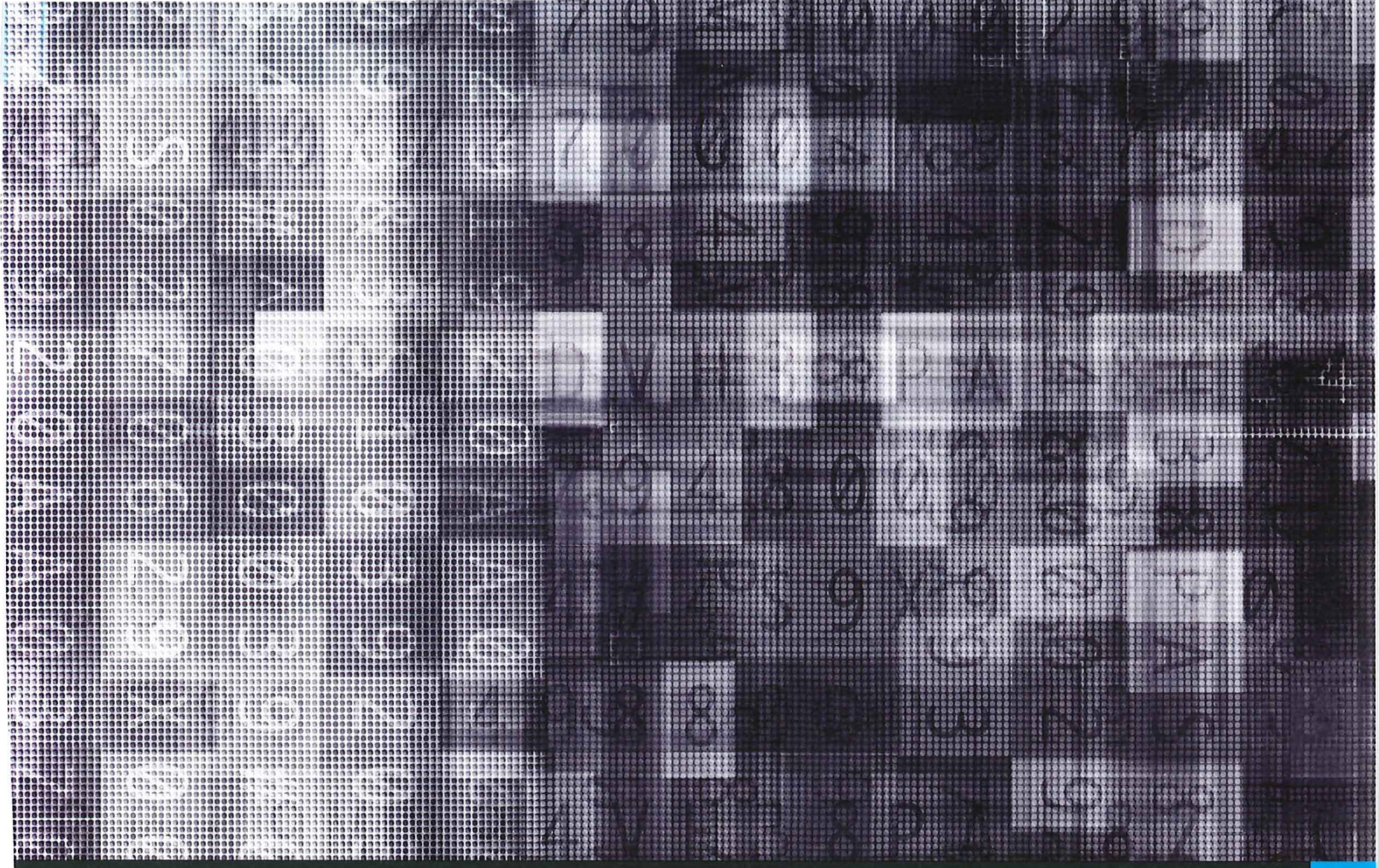
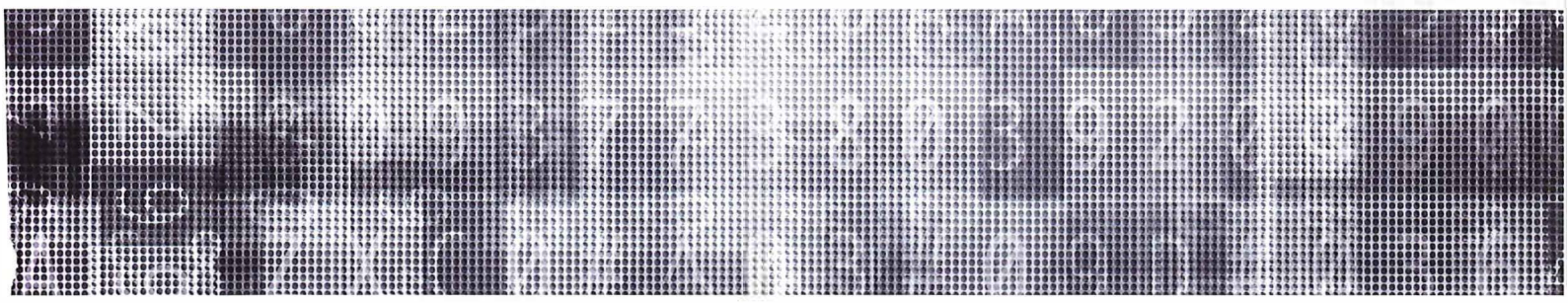


Chapter 13:
Algorithm Design and Problem-solving

Learning objectives

By the end of this chapter you will:

- understand that every computer system is made up of sub-systems
- be able to use top-down design and structure diagrams
- be able to explain standard methods of solution including pseudocode and flowcharts
- be able to suggest and apply suitable test data
- understand the need for validation and verification checks to be made on input data
- be able to use trace tables to find the values of variables at each step in an algorithm
- be able to identify errors in given algorithms and suggest ways of removing these errors
- be able to produce an algorithm for a given problem.



13.01 Computer systems and sub-systems

SYLLABUS CHECK

Understand that every computer system is made up of sub-systems, which in turn are made up of further sub-systems.

Use library routines and subroutines.

We often think of a computer system as one single system that performs tasks. However, it a system that is made up of sub-systems, that can often be made up of sub-systems too.

For example, a smartphone does not just have one system that allows us to make telephone calls. It has many sub-systems that are part of this. These sub-systems manage:

- inputs and outputs via the touch screen
- file storage
- radio transmissions.

These sub-systems also have sub-systems of their own:

- The input/output sub-system has sub-systems to update the display and detect user input.
- The file handling sub-system has sub-systems to write data to storage and read data from storage.
- The radio transmissions sub-system has sub-systems to manage outgoing and incoming transmissions.

Each system and sub-system is created from a series of instructions that tell the computer what to do. These steps are known as an algorithm.

KEY TERM

Subroutine – a short section of code within a program.

Library routine – a commonly used function that is available to a programmer.

Algorithm designers can also make use of **subroutines** and **library routines** to simplify programs.

A subroutine is a small section of a program that is part of a larger program. For example, a game might include a subroutine to display a high-score table.

A library routine is a collection of small, commonly used programs and subroutines that can be used in another program. For example, a library routine could create random numbers. The main program can use this library routine whenever a random number needs to be created. This kind of library routine could be used to simulate throwing dice in a game.

TEST YOURSELF

Research two library routines for a particular programming language. What can they be used for?

13.02 Top-down design and structure diagrams

SYLLABUS CHECK

Use top-down design, structure diagrams, flowcharts and pseudocode.

Be able to produce an algorithm for a given problem (either in the form of pseudocode or flowchart).

Algorithms can be designed in several ways and different programmers prefer some methods above others. Algorithm design methods include:

- top-down design
- structure diagrams
- flowcharts
- pseudocode.

KEY TERM

Top-down design – a design process where an overall task is broken down into smaller tasks.

Decomposition – the process of breaking down a large task into smaller tasks.

Structure diagram – a diagram that shows tasks that have been broken down and how they relate to each other.

Module – an individual section of code that can be used by other programs.

When designing a computer system we refer to the system as a 'complex problem'. The sub-systems within a system are referred to as 'sub-problems'. **Top-down design** is a design method that involves taking a complex problem and breaking it down into smaller sub-problems. These sub-problems can often be broken down into even smaller sub-problems. This process is known as **decomposition**.

The complex problem of the smartphone system was broken down into sub-problems:

- a sub-system to manage inputs and outputs via the touch screen
- a sub-system to manage file storage
- a sub-system to manage radio transmissions.

This can be represented diagrammatically as a **structure diagram** (also known as a top-down diagram). A structure diagram shows how each system (problem) and sub-system (sub-problem) is broken down from top to bottom. Each box is referred to as a **module**.

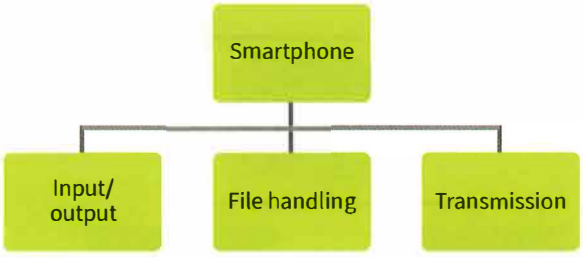


Figure 13.01 A simple structure diagram of sub-systems for a smartphone

Once each sub-problem has been identified, it can then be broken down into smaller sub-problems. These sub-problems are represented in the structure diagram with their own modules:

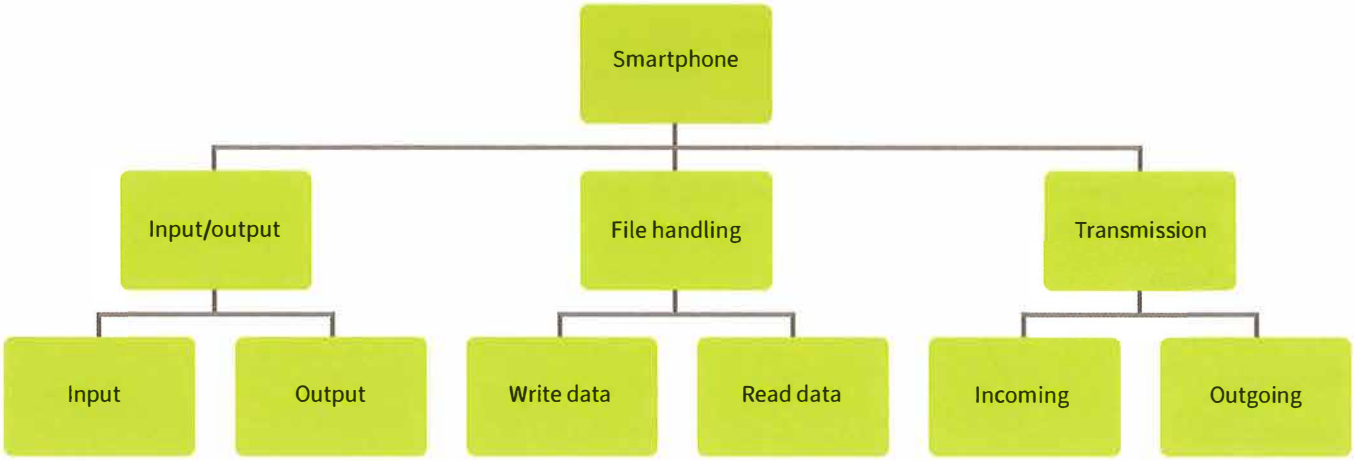


Figure 13.02 A structure diagram of sub-systems within sub-systems in a smartphone

The decomposition process can be continued until each problem is broken down into its simplest state. A designer starts at the top and breaks down each problem and sub-problem, hence the term 'top-down design'.

A good way to approach top-down design is to follow these steps:

- To solve a large problem, break the problem into several smaller tasks and work on each task separately.
- To solve each task, treat it as a new problem that can be broken down into smaller problems.
- Repeat this process with each new task until each can be solved without the need for any further decomposition.

A finance office needs an algorithm to calculate weekly pay. The algorithm needs to determine how many normal hours and how many overtime hours each employee works. Overtime is paid at 1.5 times the normal hourly rate. An employee's total weekly pay should be output at the end.

A top-down solution might look like the one in Figure 13.03.

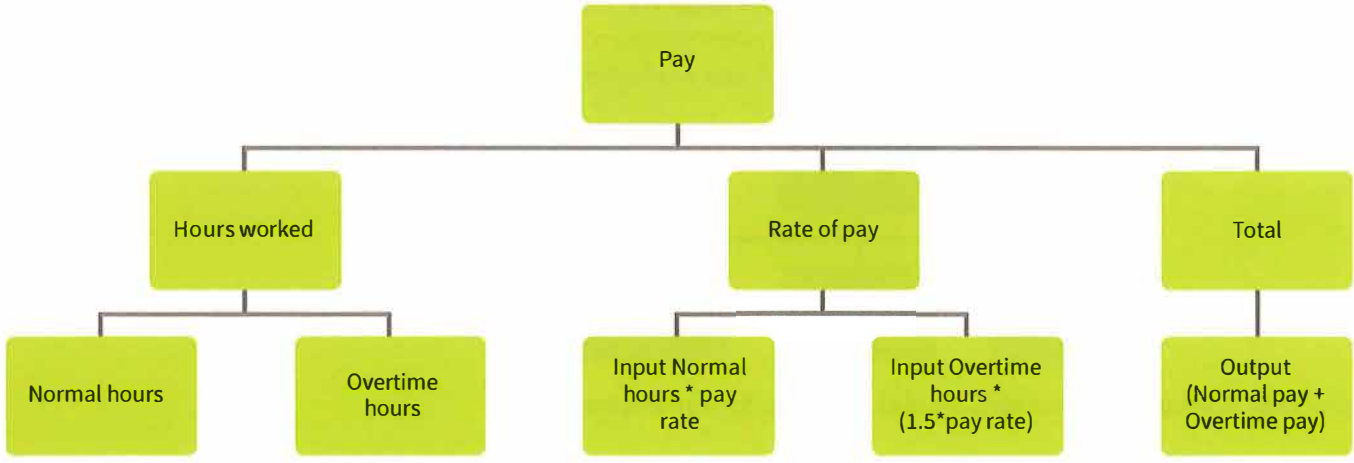


Figure 13.03 A structure diagram of an algorithm to calculate an employee's weekly pay

From this top-down design a suitable algorithm could be created to calculate the correct weekly pay for each employee.

TEST YOURSELF

The finance office is based in the UK where pay is currently taxed at 20%. Create a new top-down design to include the deduction of tax from pay.

Using top-down design to design an algorithm has several benefits:

- Algorithms can be developed quickly as more than one person can work to solve the problem.
  - Different designers can work on modules individually then bring the sub-problems together at the end.
  - Once each module has been designed, they can be given to different programmers to code. This speeds up the development of the program.
  - Each programmer can be given a module that is suitable for their area of expertise. For example, an expert in interface design could be given the interface modules and an expert in file-handling could be given the file-handling modules.
  - As each module is a shorter program they are easier to write, test and de-bug than the full system.
- The structure diagram shows how different modules relate to each other. This can be important in helping to visualise how a program works.
- Modules can be re-used in different programs that require the same structure.
- Structure diagrams make it easier for another designer to understand the logic of an algorithm.

13.03 Standard methods of solution

SYLLABUS CHECK

Explain standard methods of solution.

There are a number of ways of designing a solution to a problem, in the form of an algorithm. The two main methods of solution are flowcharts and pseudocode.

Flowcharts

A flowchart is a way of representing the structure and flow of an algorithm. Flowcharts show the sequence, selection and iteration within an algorithm. Individual steps (sequence) are represented, as are the different paths (selection) and loops (iteration) that might be followed whilst working through the algorithm.

Table 13.01

Advantages of using flowcharts	Disadvantages of using flowcharts
The sequence of steps can easily be seen	They can take a lot of time to produce
Paths through the algorithm can be easily followed	Changes to the algorithm mean sections of the flowchart have to be re-drawn. This can take a lot of time
The logic of the algorithm (sequence, selection and iteration) can often be understood better when represented visually	Large flowcharts can become extremely complicated and difficult to follow
Flowcharts can form part of the documentation for a system, helping others to easily understand how a program works	

To learn more about flowcharts, see Chapter 12.

Pseudocode

Pseudocode is a text-based method of describing an algorithm. It looks like a programming language, but in fact it is not a programming language at all. Instead, it borrows terminology from programming languages, but it does not follow the **syntax** of any particular language.

KEY TERM

Syntax – the structure of a language.

Pseudocode is useful because it represents a half-way point between a programming language and a description of the algorithm in plain English.

A pseudocode algorithm for calculating the cost of a purchase could be:

```
discount ← 0.9
price per unit ← USERINPUT
quantity ← USERINPUT
result ← (price per unit * quantity)
IF result > 100 THEN
    result ← (result * discount)
ENDIF
PRINT result
```

TEST YOURSELF

Using the cost-calculating algorithm, determine the cost of purchasing a single item when:

1 the unit price is €75

2 the unit price is €120.

To learn more about pseudocode, see Chapter 12.

There are often a number of different ways to create a solution to a problem. Some of the ways may be effective, some may not be too effective and could be improved. In order to consider how effective a solution is we need to look at the algorithm and ask some questions:

- Could the algorithm be simplified in any way?
- Are any parts of the algorithm repeated that could be made more efficient if placed in a loop/procedure?
- Does the algorithm have any parts that are not used and therefore unnecessary?

If the answer is yes to any of the questions above then the algorithm is not the most effective way of solving the problem and we can look to improve the algorithm as a result.

13.04 Test data

SYLLABUS CHECK

Be able to suggest and apply suitable test data.

During the development of a program, and when it is complete, it is good practice to test the code to make sure it does not contain any errors (bugs). It is possible for even the simplest part of a program to contain bugs. To test a program, each section of the code is run using sample data. This sample data is known as test data.

Three types of test data should be used to make sure a program is bug-free:

- Normal (typical) data is valid data that the code would be expected to accept. For example, if an input should be an integer in the range 1–100, normal data would be any integer in that range.
- Extreme data is also valid data that the code would be expected to accept. However, the data used is at the boundary of what should be accepted. In the case of an input requiring an integer in the range 1–100, the values 1 and 100 would be considered to be extreme test data.
- Invalid (erroneous) data is data that the code should not accept. Data that should not be accepted is entered to test that the code responds correctly to such an input. For example, entering a letter when code expects an integer should result in an error message and the user being asked to re-input the data.

A program has been written that calculates sales of tickets for an outdoor festival. A customer is allowed to buy between 1 and 5 tickets. The program could be tested with the test data shown in Table 13.02.

Table 13.02

Type of test data	Examples
Typical	2 3 4
Extreme	1 5
Invalid	0 6 1.5 'y' entering nothing at all

TEST YOURSELF

Explain why each of the above data meets the criteria of normal, extreme and invalid data?

Before a program is tested a testing table is usually drawn up. A testing table should contain the headings shown in Table 13.03.

Table 13.03

Test description	Test data	Test type	Expected outcome	Actual outcome
Enter number of tickets to be bought	1	Extreme	Entry should be accepted and customer transferred to the 'confirm order' screen	Successful
Enter number of tickets to be bought	6	Invalid	Error message 'Please enter a number between 1 and 5' should display	Successful

Testing should be thorough and complete. Several examples of normal, extreme and invalid data should be entered to make sure that the code works as expected.

13.05 Validation and verification

SYLLABUS CHECK

Understand the need for validation and verification checks to be made on input data (validation could include range checks, length checks, type checks and check digits).

When we enter data it is impossible to guarantee that the data we enter is error free. When data is entered or copied from one place to another, errors can easily be introduced. It is very easy to accidently omit data, mistype data, or misread a value or word. To help prevent data entry errors, two techniques are used: validation checks and verification checks.

KEY TERM

Validation – an automatic check to make sure data is sensible and possible.

Verification – a check to see if the data entered is correct.

Validation checks

Validation checks ensure that data being entered is both sensible and possible. Table 13.04 show types of validation checks.

Table 13.04

Validation check	Description	Example data
Range check	Specifies that data entered must be numerical data in a range, for example in the range 1 to 18	5 (valid) 19 (invalid)
Presence check	Makes sure that a field has to have data entered in it, for example, when entering a new password	Any data entered (valid) Leaving it blank (invalid)
Type check	Specifies that only a certain type of data can be entered, for example, not allowing letters into a field that should only contain numbers	5 (valid) five (invalid)

Validation check	Description	Example data
Lookup	Providing the user with a drop-down list to select data. For example, when selecting an employment status, a drop-down list of the options employed, self-employed, unemployed	Anything from the list (valid) Anything else (invalid)
Length check	Specifies that data entered has to be a certain number of characters long. For example, a password needs to have 8 or more characters	p@s5w0rd (valid) l3tme1n (invalid)
Format check	Specifies that data entered has to conform to a specific format. Some pieces of data are formed of characters in a very specific order. For example, a UK vehicle registration has two letters, followed by two numbers, followed by a space, followed by three letters	MA63 SLK (valid) D15G JPT (invalid)

More than one type of validation check might be needed for a set of data. For example, a shop would apply a format check to make sure each product it sells is given a selling price in the same format. Additionally, a presence check would also be applied to make sure no product is left without a selling price.

TEST YOURSELF

A real estate agent stores data about the properties they sell. Suggest suitable validation for each data item in Table 13.05 and justify your choice.

Table 13.05

Data	Validation checks
Property type	
Number of bedrooms	
Garden	
Seller's family name	
Selling price	

Check digits

When entering a series of numbers it is easy for a user to incorrectly enter a number. A special form of validation checks that a series of numbers has been entered correctly by carrying out a series of calculations on the numbers entered.

Various methods exist and a good example is the check digit applied to International Standard Book Numbers (ISBN). Every book sold has an ISBN. This consists of a series of either 10 or 13 numbers. The check digit method for making sure that 10/13 correct numbers have been entered is known as the ISBN-13 check digit. The ISBN-13 check digit makes use of modulo-10 division. This is a method that divides a sum by 10 and uses the remainder as part of the final calculation.

Consider the ISBN number 9782053703008. With ISBN-13, the check digit is the last digit in the ISBN number. In this case the check digit is 8. This digit is removed from the number as it is used in the final check. Each remaining digit is given a value by which is it multiplied, called a 'weighting'. The values are standard and are applied to all ISBN-13 numbers. The values are 1 and 3 and they alternate from number to number as shown in Table 13.06.

Table 13.06

ISBN digit	9	7	8	2	0	5	3	7	0	3	0	0
Weighting	1	3	1	3	1	3	1	3	1	3	1	3

To calculate the check digit, carry out the following steps:

1 Multiply each digit by its weighting, then add together the result of each of the calculations:

$(9 \times 1) + (7 \times 3) + (8 \times 1) + (2 \times 3) + (0 \times 1) + (5 \times 3) + (3 \times 1) + (7 \times 3)$
$+ (0 \times 1) + (3 \times 3) + (0 \times 1) + (0 \times 3)$
$= 9 + 21 + 8 + 6 + 0 + 15 + 3 + 21 + 0 + 9 + 0 + 0$
$= 92$

2 Apply modulo-10 division to the sum to determine the remainder. Modulo-10 division means the number is divided by 10 and the remainder of this calculation is noted:

$92 \div 10 = 9$ remainder 2

3 Finally, subtract the remainder from 10. The result is the check digit value:

$10 - 2 = 8$.

The result of the calculation matches the check digit at the end of the ISBN: they are both 8. As they match, the computer knows that the ISBN number has been entered correctly.

Check digits are also used in data transmission. To find out more about this, see Chapter 2.

Verification checks

Validation helps to make sure that data is both sensible and possible. However, validation does not ensure that the data is correct. Even if the data is reasonable and possible, it might still be incorrect.

In a range check of 1 to 10 a user may mean to enter a value of 8, but instead enter a value of 7. The value would still be accepted as it is between 1 and 10, but it is incorrect as the user meant to enter 8. **Verification** is intended to catch such errors.

There are three ways in which data can be verified; two are automated methods:

- Double entry – The user enters the data twice. The computer checks to see that both entries match. If they do not match, the user is prompted to re-enter the data again. This is often used when entering a password. It makes sure the user has set the password they wanted and has not mistyped it.
- Twin entry – Two users enter the same data separately. The computer checks to see that both entries match. If they do not match, both users are prompted to re-enter the data again.
- Proofreading – One user enters the data and a second user reads it. If it appears to be okay, the data is accepted into the system. Otherwise the second user amends the data, then submits it.

Despite the use of validation and verification, data can still be entered incorrectly. For example, it is possible for a user to enter a number or name incorrectly the same way twice. Validation and verification prevent many mistakes but it cannot stop all data entry errors from occurring.

13.06 Trace tables

SYLLABUS CHECK

Be able to use trace tables to find the value of variables at each step in an algorithm.

Sometimes when we test a program it produces unexpected results. For example, consider the following simple pseudocode algorithm for calculating the average of a set of four numbers:

```
total ← 0
count ← 0
WHILE count < 3:
    INPUT ("Enter a number") as float
    GET number
    total = total * number
    count = count + 1
END WHILE
average = total / count
PRINT ("Average = ", average)
```

A programmer converts the algorithm to code. The program is run and tested with the following data:

1, 2, 3

The total of this data is 6, which gives an average of 2 (6 ÷ 3 = 2). However, when the program is run, the average is output as 0. There is clearly an error, but where?

KEY TERM

Dry run – paper-based run through an algorithm or a program.

Trace table – a table used to trace the values of variables through the execution of a program.

To determine where errors are in code, programmers often perform what is called a **dry run** using a **trace table.** With a dry run, the programmer makes a list of all the variables in the order they appear in the code and places them in a table. A trace table for the algorithm above would look like Table 13.07.

Table 13.07

total	count	number	average

To perform the dry run, the programmer works manually through each line of code, making a note of any variable that changes value by placing that new value in a new line in the trace table. From this it can often be seen where an error is occurring as shown in Table 13.08.

Table 13.08

total	count	number	average
0			
	0		
		1	
	1		
		2	
	2		
		3	
	3		
			0

From this trace table it can be immediately seen that the error lies in the section of the algorithm that adds the input number to the total. The average is output as 0, because the total stays 0 and $0 \div 3 = 0$.

Inspecting the algorithm reveals an incorrect line:

```
total = total * number
```

This should read:

```
total = total + number
```

When a dry run is performed again using the same data, the trace table gives the values shown in Table 13.09.

Table 13.09

total	count	number	average
0			
	0		
		1	
1			
	1		
		2	
3			
	2		
		3	
6			
	3		
			2

Trace tables are extremely useful as they can help to quickly pinpoint where an error is occurring, simply by noting where a variable holds a different value than what we expected. However, if the program has many iterations a trace table can be extremely long.

SYLLABUS CHECK

Identify errors in given algorithms and suggest ways of removing these errors.

13.07 Errors in a program

KEY TERM

Syntax error – an error that occurs because the programmer does not use the correct language structure in a program.

Logic error – an error that causes a program to do something unexpected.

Programmers are human. Humans make mistakes. When programming, humans make two main types of mistake: **syntax errors** and **logic errors**.

Syntax errors

A syntax error occurs when a programmer does not follow the rules or structure of the language they are writing in. Syntax errors occur in various forms:

For example, in Visual Basic, a variable is declared by using the following syntax:

```
Dim noOfStudents As Integer
```

Possible syntax errors are shown in Table 13.10.

Table 13.10

Code with a syntax error	Description of error
`Dim noOfStudents`	The line is incomplete
`Dim noOfStudents As Interger`	The line contains a spelling error
`Dim Integer As noOfStudents`	The elements are in the wrong order

If a program contains a syntax error it will stop when the line containing the error is reached.

Logic errors

A logic error is an error in the code that causes the program to do something it should not, for example, produce an incorrect result to a calculation. Consider the following code, which should calculate and output an exam percentage:

```
print (score/possibleMarks) * 1000
```

In calculating the percentage, the code should multiply the result by 100. The code above multiplies by 1000. Although the code will run, it will display the wrong answer.

Other examples of logic errors include:

- Using a conditional loop where the condition is unintentionally never met would result in what we call 'an infinite loop' (it will never stop looping).
- Assigning the wrong data type to a variable, for example specifying an integer instead of a float, may result in rounded up numbers.
- Using the same variable name for different purposes within a program may mean that the variable's value changes unexpectedly.

Summary

- Computer systems consist of sub-systems. These sub-systems can have sub-systems of their own.
- Algorithms are step-by-step designs that describe how systems and sub-systems work.
- Subroutines are short sections of code within a program.
- Top-down design is a design method where a problem is broken down into smaller problems. Structure diagrams are used to show how modules fit together and relate to each other.
- Flowcharts represent the structure and flow of an algorithm.
- Pseudocode is a text-based method of describing how an algorithm works.
- Test data is used to test a program. A test table documents the results of testing.

- It is impossible to guarantee that data entered is error free. Validation checks and verification checks help to reduce errors.
- Validation checks make sure that data entered is both sensible and possible.
- Verification tries to ensure that data entered is correct.
- Dry runs are paper-based runs of an algorithm or program.
- Trace tables allow a programmer to easily pinpoint where errors are, but can be extremely long and time-consuming to produce.
- Syntax errors occur when the programmer enters code that does not follow the rules of the programming language.
- Logic errors occur where the programmer writes code that does not perform as intended.

Exam-style questions

1 Describe what is meant by a library routine and suggest reasons why using such a routine is of benefit to a programmer. (2 marks)

2 Seema wishes to design an algorithm using a flowchart. Discuss the possible benefits and drawbacks of using a flowchart to design her algorithm. (2 marks)

3 Consider this algorithm:


```
value ←-- 0
nextValue ← 0
INPUT value
INPUT nextValue
WHILE nextValue != 0# != means not
   IF nextValue > value# note indentation
      value = nextValue
   ENDIF
   INPUT nextValue
ENDWHILE
OUTPUT value
```

Complete the following trace table using the input data 4, 3, 10, 1, 0, 15: (5 marks)

value	nextValue	WHILE nextValue != 0	IF nextValue > value	OUTPUT value

4 Explain the difference between a logic error and a syntax error. (2 marks)

5 Explain the difference between validation and verification. (2 marks)

6 Why is it important to use test data? Use examples in your answer. (4 marks)