EDEXCEL COMPUTER SCIENCE FOR GCSE

COMPUTER SCIENCE

SAMPLE CHAPTERS

Steve Cushing

DYNAMIC LEARNING

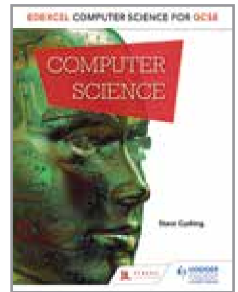
HODDER EDUCATION
LEARN MORE

Meet the challenges of the new GCSE specification with print and digital resources to support your planning, teaching and assessment needs alongside specialist-led CPD events to help inspire and create confidence in the classroom.

We will be seeking endorsement for the following textbook :

Edexcel GCSE Computer Science Student Book 9781471866227 June 2016 £19.99

Visit www.hoddereducation.co.uk/ComputerScience/GCSE/Edexcel to pre-order your class sets, or to sign up for your Inspection Copies or eInspection Copies.



ALSO AVAILABLE:

Edexcel GCSE Computer Science Dynamic Learning

Dynamic Learning is an innovative online subscription service with interactive resources, lesson planning tools, self-marking tests, a variety of assessment options and eTextbook elements that all work together to create the ultimate classroom and homework resource.

"I'd have no time left to teach if I collected all these resources. It's a great time saver."

Caroline Ellis, Newquay Tretherras

9781471886621 February 2017 £8.99

To sign up for a free 30-day trial, visit www.hoddereducation.co.uk/dynamiclearning

My Revision Notes: Edexcel GCSE Computer Science

Ensure your students have the knowledge and skills needed to unlock their full potential with this revision guide from our best-selling series.

Prices from £9.99
Pub date: January 2017
To sign up for Inspection Copies visit www.hoddereducation.co.uk/ComputerScience/GCSE/Edexcel

Philip Allan Events

Ensure that you are fully prepared for the upcoming changes by attending our specialist-led CPD courses.

For more information and to book your place visit www.hoddereducation.co.uk/Events

EDEXCEL COMPUTER SCIENCE FOR GCSE

COMPUTER SCIENCE

Steve Cushing

DYNAMIC LEARNING

HODDER EDUCATION

AN HACHETTE UK COMPANY

Although every effort has been made to ensure that website addresses are correct at time of going to press, Hodder Education cannot be held responsible for the content of any website mentioned. It is sometimes possible to find a relocated web page by typing in the address of the home page for a website in the URL window of your browser.

Hachette UK's policy is to use papers that are natural, renewable and recyclable products and made from wood grown in sustainable forests. The logging and manufacturing processes are expected to conform to the environmental regulations of the country of origin.

Orders: please contact Bookpoint Ltd, 130 Milton Park, Abingdon, Oxon OX14 4SB. Telephone: (44) 01235 827720. Fax: (44) 01235 400454. Lines are open 9.00–17.00, Monday to Saturday, with a 24-hour message answering service. Visit our website at www.hoddereducation.co.uk

First published in 2016 by

Hodder Education
An Hachette UK Company,
Carmelite House
50 Victoria Embankment
London EC4Y 0DZ

Contents

2 Using flowcharts

Specification references

You should:

1.1.1 understand what an algorithm is, what algorithms are used for and be able to interpret algorithms (flowcharts, pseudo-code, written descriptions, program code)

1.1.2 understand how to create an algorithm to solve a particular problem, making use of programming constructs (sequence, selection, iteration) and using appropriate conventions (flowchart, pseudo-code, written description, draft program code)

1.1.3 understand the purpose of a given algorithm and how an algorithm works

1.1.4 understand how to determine the correct output of an algorithm for a given set of data

2.4.1 understand how to write code that accepts and responds appropriately to user input

! Key points

- A **flowchart** is a diagram representation of an algorithm.
- For the examination, you will need to be able to interpret flowcharts.
- Flowcharts are a graphical method of designing programs.
- A well-drawn flowchart is easy to read.

There are a lot of different design procedures and techniques for building large software projects. The technique discussed in this chapter, however, is for smaller coding projects and is referred to by the term 'top down, structured **flowchart** methodology'. We will explore how to take a task and represent it using a flowchart. A flowchart puts the sentences from a sequence into shaped boxes. The shapes indicate the action.

You will know from the last chapter that a sequence is where a **set of instructions** or actions are **ordered**, meaning that each action follows the previous action.

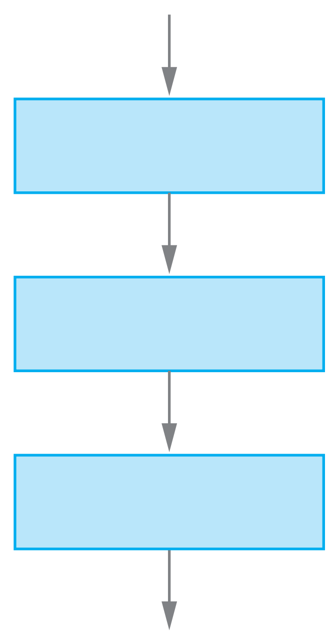


Figure 2.1 A sequence

! Key point

In a flowchart the lines express the order of execution.

Flowchart advantages

- Flowcharts are a graphical way of writing pseudo-code.
- They are all standardised: we all pretty much agree on the symbols and their meaning.
- They are very visual.

Flowchart disadvantages

- Flowcharts can be time consuming and difficult to modify.
- They need special software for symbols although some software has these built in.
- The structured design elements are not all implemented.

General rules for flowcharts

- All symbols of the flowchart are **connected by flow lines** (these must be arrows not lines to show direction).
- Flow lines **enter the top** of the symbol and **exit out the bottom**, except for the Decision symbol, which can have flow lines exiting from the bottom or the sides.
- Flowcharts are drawn so flow generally goes from the **top to the bottom** of the page.
- The beginning and the end of the flowchart is indicated using the Terminal symbol.

Let's look at a simple sequence. Say we want to calculate A plus B, where A = 200 and B = 400.

Start
A = 200 B = 400
Add = 200 + 400
Output = 600
End

We could create the simple flowchart shown in Figure 2.2.

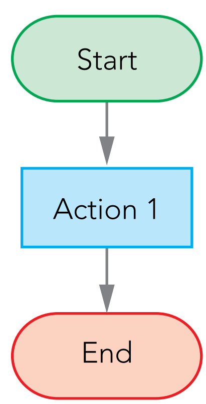


Figure 2.2 A simple flowchart

Let's look at a another sequence, for example the sequence you carry out each morning in the bathroom. This sequence could be:

- Brush teeth
- Wash face
- Comb hair.

As you can see, **sequences** are a useful tool for showing what happens and in what logical order, but each step, for example 'brush teeth', needs to be defined in more detail to be carried out.

Key points

- Flowcharts must have flow lines with arrows to show the order of execution.
- An algorithm is a sequence of steps that can be followed to complete a task.
- A sequence is where a set of instructions or actions are ordered, meaning that each action follows the previous action.

Task

1 Produce a sequence to show how to brush your teeth.

Questions

1 What is a Terminator?
2 What is a Sequence?
3 What is an Input/Output?

Once we have picked up our toothbrush, turned on the tap and added the toothpaste we can put the toothbrush in our mouth and brush. The act of actually brushing your teeth could be recorded in a linear way (press, brush up, brush down, brush up, brush down, etc.), but it would be much simpler to explain the brushing once and then tell the user to **repeat** the same action ***x* times**. We will explore this later when we consider looping (iteration), but for now let us explore how we can use a flowchart to represent simple sequences. First we need a few more elements.

! Key point

Cleaning your teeth is called a procedure in coding. You perform the same action every day, for example: pick up brush, put toothpaste on brush, brush teeth for two minutes, spit out, clean brush. These actions could be given a procedure name: 'Brushing Teeth'.

! Key points

- You use a **PROCESS** symbol for an **OPERATION** or **ACTION STEP**.
- You use a **TERMINATOR** symbol for a **START** or **END** in a **PROCESS**.
- You use a **DECISION** symbol for a **QUESTION** or **BRANCH** of a process.
- Flowchart symbols contain text called labels.
- For the examination, you will need to understand how to create an algorithm using a flowchart.

Basic elements of flowcharts

The flowchart symbols denoting the basic building blocks of programming are shown in Figure 2.3 below. Text inside a symbol is called a label.

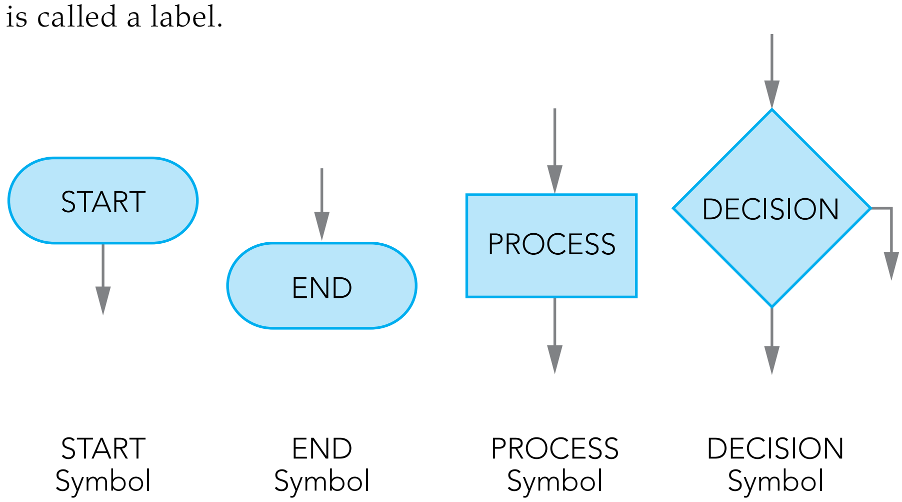


Figure 2.3 Basic elements of a flowchart

The **START** symbol represents the **start of a process**.

The process symbol is **labelled** with a brief description of the process carried out by the flowchart. The **END** symbol represents the **end of a process**. It contains either 'End' or 'Return' depending on its function in the overall process of the flowchart.

Representing a process

A **PROCESS** symbol is representative of some operation that is carried out on an element of data. It usually contains a brief description of the process being carried out on the data. It is possible that the process could be even further broken down into simpler steps by another complete flowchart representing that process. If this is the case, the flowchart that represents the process will have the same label in the 'Start' symbol as the description in the 'Process' symbol at the higher level. A process always has exactly **one input arrow** and **one output arrow**.

In practice, sequences are not a simple line. Often the next action depends on the last decision. This is called **selection**. In selection, one statement within a set of program statements is executed depending on the state of the program at that instance. We ask a question and choose one of two possible actions based upon that decision.

Key points

- All flowcharts must have a START and an END symbol.
- The DECISION symbol will have exactly one input and two outputs.

Representing a decision

A DECISION/SELECTION symbol always makes a **Boolean** choice. We will explore Booleans in more detail later in the book. The label in a decision symbol should be a question that clearly has **only two possible answers** to select from.

Key points

- One of the most confusing things in a flowchart is telling the loops apart from the selections. This is because both use the diamond shape as their control symbol. Mark them clearly.
- A 'Decision' symbol always makes a Boolean choice.

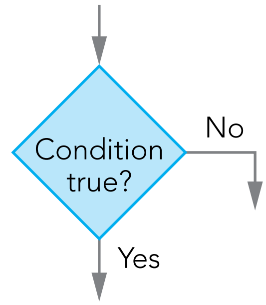


Figure 2.4 How to represent a decision

The DECISION symbol will have exactly one input arrow and two output arrows. The two outputs will be labelled with the two answers to the question, in order to show the direction of the logic flow depending upon the selection made.

Selections are usually expressed as decision key words, such as 'if ... then ... else ... endif, switch or case'. They are at the heart of all programming.

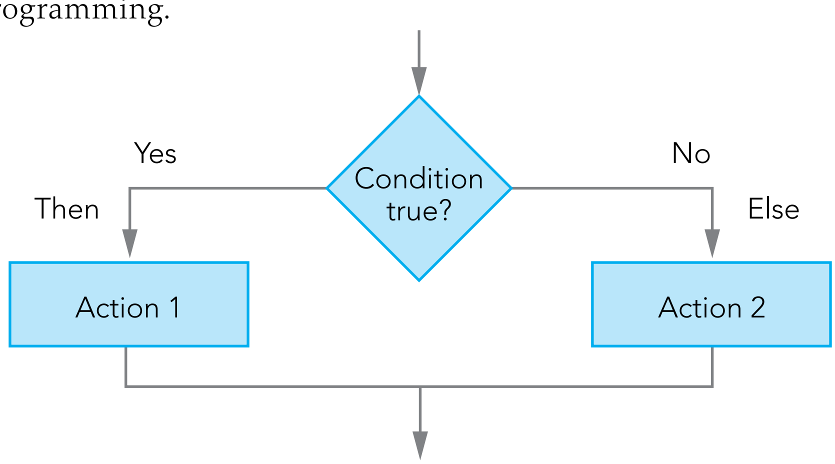


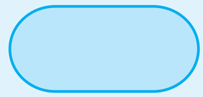
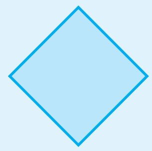
Figure 2.5 A flowchart representing a selection

Questions

4 What is a flowchart?

5 What is a Selection?

Flowcharts can use the following symbols:

Symbol	Purpose	Use
	Flow line	The lines show the sequence of operations.
	Terminal (Start/Stop)	Denotes the start and end of an algorithm.
	Processing	Denotes a process to be carried out.
	Decision	Used to represent the operation in which there are two alternatives, true and false.

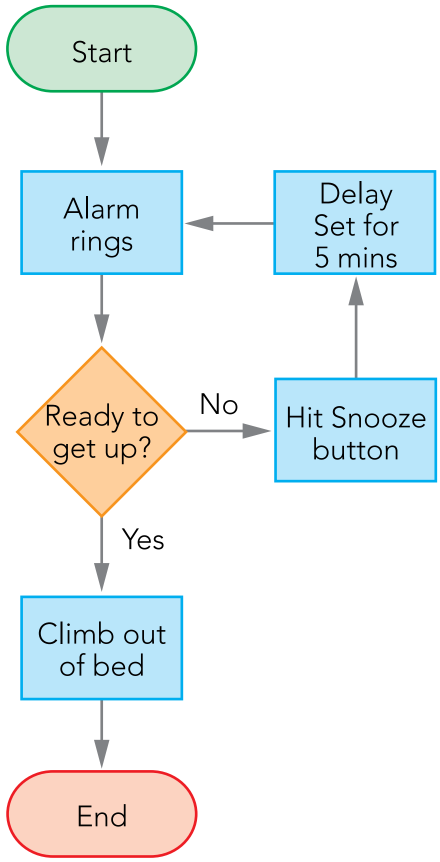


Figure 2.6 A flowchart showing what happens on a school morning

We can use a decision to create a flowchart of what happens in the morning on school days, as shown in Figure 2.6.

We also explored selections when we looked at the sequence of making tea. We explored using **IF** someone wants sugar and **IF** someone wants milk. The process of making the tea differed according to their answer to these questions.

The flowchart below shows a different process for making tea and adds two decision boxes.

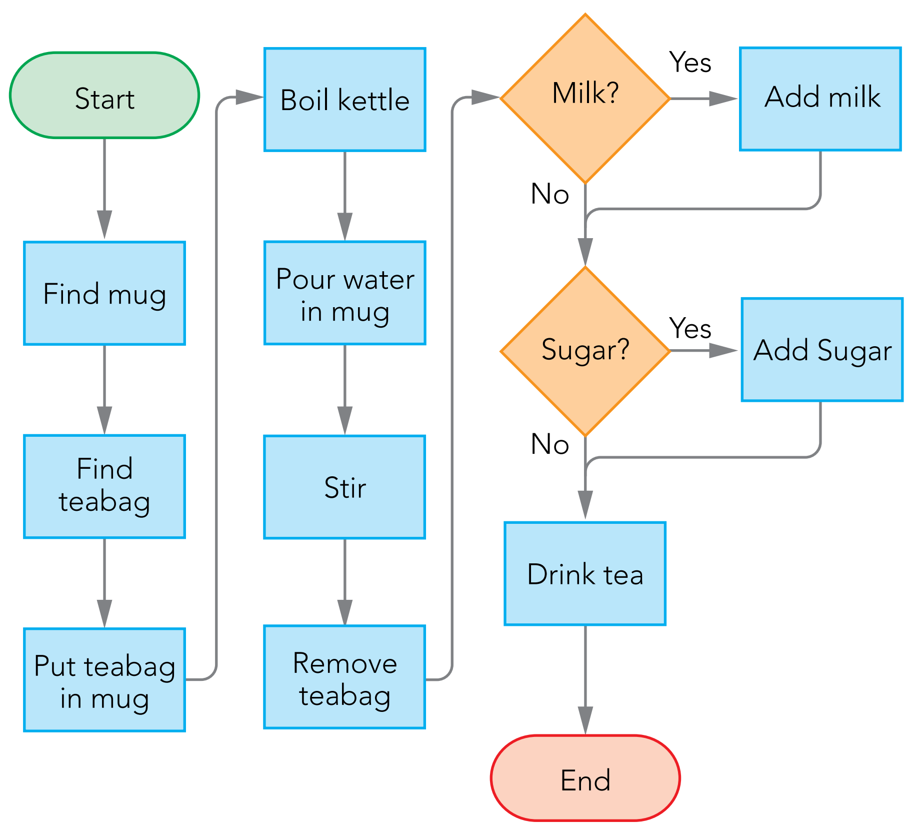


Figure 2.7 A flowchart showing a difference process for making tea

If we wanted to show how to play the game of snakes and ladders we could explain how to play the game in English as follows:

```
Start game
Throw the dice: the number indicated by dice is x.
Move your counter x squares on the board and check:
  Have you landed on snakes head?: no/yes
    If yes slide down snake to its tail.
    If no check next statement
  Have you landed on the bottom of the ladder?: no/yes
    If yes move up the ladder.
    If no check next statement
  Have you reached the last block of the game?: no/yes
    If yes
      Output "you are the winner"
    If no
      Give the dice to the next player
Repeat until someone reaches the last block of the game.
End
```

Key point

Repetition is used when the same bit of code is needed several times. Instead of writing it over and over again, you can use the REPEAT command. Repetition can also be called iteration (looping).

We have more decisions in this example and could show the game with the following flowchart.

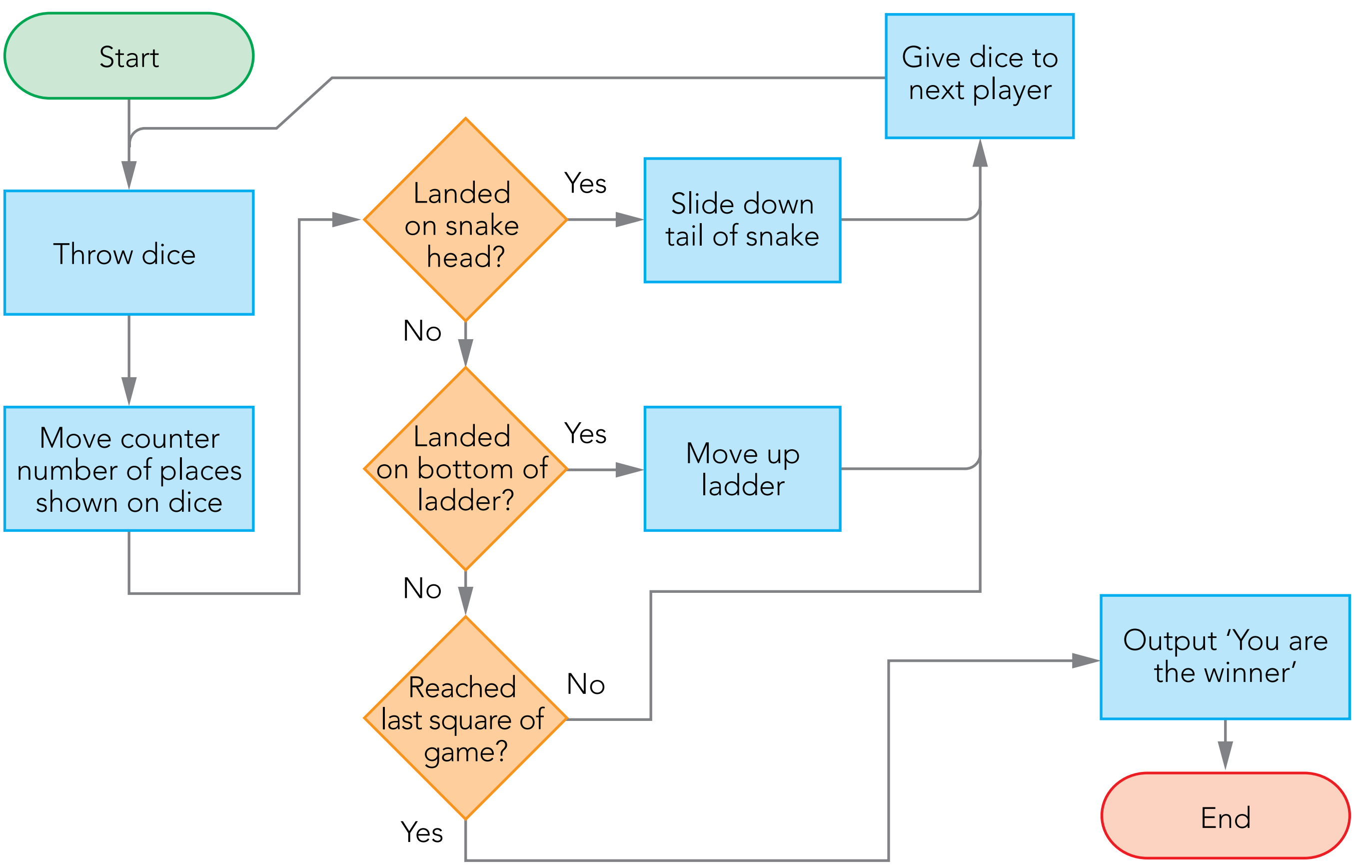


Figure 2.8 A flowchart showing the game of snakes and ladders

Other structures we will use in this book include:

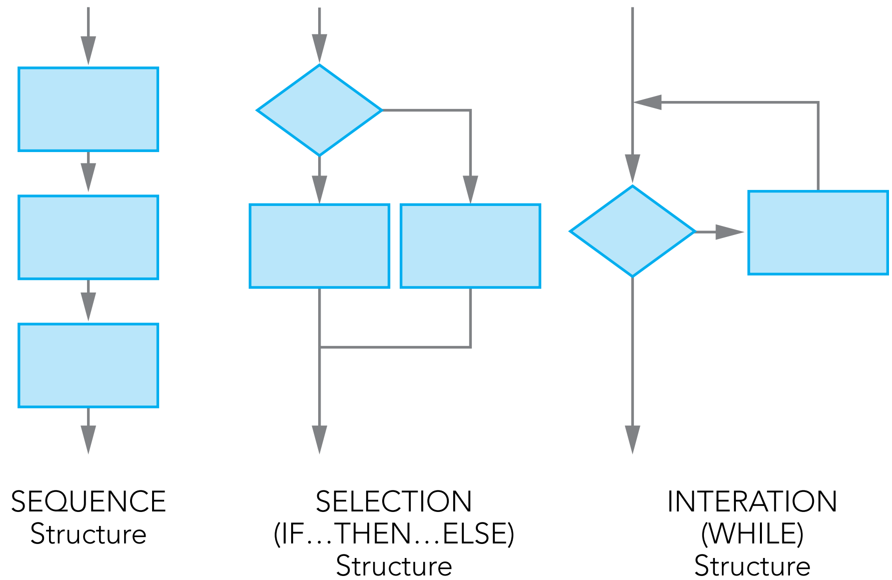


Figure 2.9 Other structures we will use in this book

On-page and off-page **connectors** may also appear in some flowcharts. This occurs when a flowchart goes over more than one page. For the purposes of this chapter we will only explore flowcharts that can be represented on a single page. If a flowchart is so big it needs to go onto another page, you should split it into sub-processes

Subprocesses

We can also use **subprocesses** in flowcharts using the symbol below.

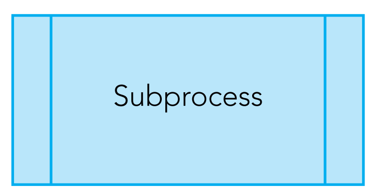


Figure 2.10 The subprocess symbol

Subprocesses are useful because:

- they help with the modularisation of complex programs;
- they provide a way of simplifying programs by making common processes available to a wide number of programs;
- they lead to more reliable programs since once a process is tested and works it can be made a subprocess and need not be tested again.

In flowcharts subprocesses are also useful in sticking to the rule that a flowchart should fit on a single page.

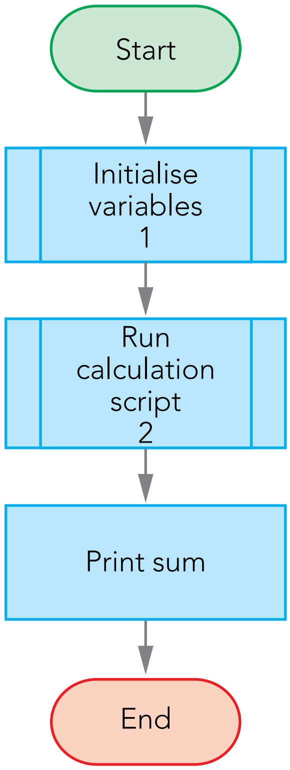


Figure 2.11 The subprocess

Figure 2.11 shows an example of the main page of a flowchart. It contains two subprocess symbols. Each subprocess symbol contains text which describes briefly what the subprocess does. Each subprocess symbol also contains a page reference where the flowchart for the subprocess will exist.

Chapter review

In this chapter we built upon the last chapter to explore sequences in more detail and established how we can show these using flowcharts.

We looked at the basic elements of flowcharts and introduced the concept of decisions and how these can be represented.

Remember, before tackling any computer science task or examination question on this topic you must:

- understand what an algorithm is and what algorithms are used for, and be able to interpret algorithms in the form of flowcharts
- Understand and be able to explain and create algorithms to solve a particular problem, making use of programming constructs such as sequence, selection and iteration
- understand and be able to use appropriate conventions in flowcharts
- understand and be able to explain the purpose of a given algorithm and explain how it works
- understand and be able to explain and determine the correct output of an algorithm for a given set of data.

3 Pseudo-code

Specification references

You should:

1.1.1 understand what an algorithm is, what algorithms are used for and be able to interpret algorithms (flowcharts, pseudo-code, written descriptions, program code)

1.1.2 understand how to create an algorithm to solve a particular problem, making use of programming constructs (sequence, selection, iteration) and using appropriate conventions (flowchart, pseudo-code, written description, draft program code)

1.1.3 understand the purpose of a given algorithm and how an algorithm works

1.1.4 understand how to determine the correct output of an algorithm for a given set of data

2.1.2 understand the benefit of producing programs that are easy to read and be able to use techniques (comments, descriptive names (variables, constants, subprograms), indentation) to improve readability and to explain how the code works

2.2.1 understand the structural components of a program (variable and type declarations, command sequences, selection, iteration, data structures, subprograms)

2.2.2 be able to use sequencing, selection and iteration constructs in their programs

2.4.1 understand how to write code that accepts and responds appropriately to user input

Basic elements of pseudo-code

Pseudo-code is another way to develop an algorithm. It consists of **natural language-like statements** that precisely describe the steps required.

Pseudo-code must:

- contain statements which **describe actions**;
- focus on the **logic** of the algorithm or program;
- avoid language-specific elements;
- be written at a level so that the desired programming code can be generated with little effort from each statement;
- contain steps, subordinate numbers and/or indentation used to show dependent statements in selection and repetition structures.

Pseudo-code advantages

- Pseudo-code is similar to everyday English.
- It helps programmers to plan an algorithm.
- It can be done easily on a word processor.
- It is easily modified.
- It implements structured concepts well.

! Key points

- Pseudo-code is a language designed to express algorithms in an easy to follow form.
- Pseudo-code is an easy to read language to help with the development of coded solutions.
- When writing in pseudo-code resist the urge to write in whatever language you are most comfortable with.

Pseudo-code disadvantages

- Pseudo-code is not visual like flowcharts.
- There is no accepted standard, so it varies widely.
- It is not an actual programming language.
- It is an artificial and informal language.
- Some people have a tendency to put actual code in. This makes it harder to understand.

? Question

1 What is pseudo-code?

The importance of syntax

Syntax is the set of rules, principles, and processes that enable us to understand a language. The syntax rules of a language define the spelling and grammar and as with natural human languages each language has its own rules. Computers are very inflexible and understand what you write only if you state what you want in the exact syntax that the computer expects and understands.

Each programming language has its own rules and specialist syntax including the words that the computer understands, which combinations of words are meaningful, and what **punctuation** is necessary for the code to be correctly structured. Whilst pseudo-code does not have a fixed syntax, you will need to understand the syntax used in the examination papers. Understanding the importance of syntax is also vital when you start using a programming language.

! Key point

For the examination you will need to understand how to create an algorithm using appropriate conventions (flowchart, pseudo-code, written description, draft program code).

Symbols

When we write code in English we also use **symbols** in the form of punctuation. Symbols are used because they are human-readable. The symbols you use are important as they have an effect in your code.

these-words-are-seperated-by-a-symbol **#the – is the symbol used here**

there is also a symbol in this sentence **#here the space is the symbol used**

Symbols can also be used as what are called **identifiers**. In some programming languages, they are also called **atoms** rather than symbols.

The symbols ←, <<, <, –, and = are often used as what are called **operators**.

! Key point

The symbols ←, <<, <, –, = are often used to represent the assignment operator in programming languages.

In pseudo-code, you use the following syntax to receive data from a device. The red brackets < > are only to show where you add something; you don't need to put them in your code.

```
Syntax
RECEIVE <add identifier here> FROM (type) <add device here>
```

Examples

RECEIVE Name FROM (STRING) KEYBOARD

or

RECEIVE LengthOfJourney FROM (INTEGER) CARD_READER

or

RECEIVE YesNo FROM (CHARACTER) CARD_READER

Common action keywords

Several keywords are often used to indicate common input, output, and processing operations.

- Input: **READ, OBTAIN, GET**
- Output: **PRINT, DISPLAY, SHOW**
- Process/compute: **COMPUTE, CALCULATE, DETERMINE**
- Initialise: **SET, INIT**
- Add one: **INCREMENT**

In pseudo-code, you use the following syntax to send output to the screen. The red brackets < > are only to show where you add something; you don't need to put them in your code.

! Key point

For the examination, you will need to be able to interpret program code.

Syntax

SEND <add expression here> TO DISPLAY

Example

SEND 'Have a good day.' TO DISPLAY

Whilst there is no common way of writing pseudo-code, in this book we have written the commands in capital letters to differentiate them from the examples in Python and to help you understand what the command words are.

Questions in the Edexcel written examination that involve code will use the following pseudo-code command words alongside other words:

ELSE

END FOR

END IF

END WHILE

FOR

IF

INPUT

OUTPUT

REPEAT

RETURN

THEN

WHILE

Commenting on your code

Key point

A comment is explanatory text for the human reader.

Good code is not only well written, but should also be **well annotated**. There are programmers who argue that comments are not necessary if the code is written well, but you are undertaking an examination and it will be useful to explain what your code does and why.

You will find many examples of commented code in this book. Comments are shown using either // or #. Different programming languages have different ways to tell the computer that **this is a comment NOT the code**. You can make all the code you write in pseudo-code a comment when you write the actual code using your chosen language. This is considered good practice when learning to code. You can also **comment out** bits of code to **find errors**, but we will explore this later.

Key points

- Good code is well written and well annotated.
- For the examination, you will need to understand the benefit of producing programs that are easy to read, and be able to use techniques (such as comments, descriptive names and indentation) to improve readability and to explain how the code works.
- For the examination, you will need to understand how to create an algorithm making use of programming constructs such as sequence, selection and iteration.

Questions in the Edexcel written examination that involve code will use the following pseudo-code syntax for comments:

```
# some text
```

Multiline comments will show the hash # for each separate comment line.

```
# some text
# some more text on a new line
```

Comments remind you and the examiner why you included certain **functions**. They also make **maintenance** easier for you later.

Have you ever tried to work with someone else's complex spreadsheet or database? It's not easy. Now imagine how difficult it is if you're looking at someone else's programming code.

Question

2 What is a comment?

When you fully document your code with comment tags, you're answering (at least) three questions:

- Where is it?
- Why did I do that?
- What does this code do?

Task

1 Describe the main reasons why a programmer would wish to annotate or add comments to their code.

No matter how simple, concise, and clear your code may end up being, it's impossible for code to be completely **self-documenting**. Even with very good code it can only tell the viewer *how* the program works; comments can also say *why* it works.

Adding selection

As we discovered in the last chapter on flowcharts, another important aspect of programming is **selection**. If we want to write pseudo-code that tells a user to enter a number to a variable, and then we want the code to see if the number they entered is a 3 or a 4 we could write a **selection algorithm** in pseudo-code that could look like this:

```
RECIEVE inputNumber FROM (int) KEYBOARD        #Input
IF inputNumber = 3                             #Selection (Process)
   SEND "your number is 3" TO DISPLAY          #Output
ELSE IF inputNumber = 4
   SEND "your number is 4" TO DISPLAY
ELSE
   SEND "your number is not 3 or 4" TO DISPLAY
END IF
```

We could also write the code a different way:

```
RECIEVE inputNumber FROM (int) KEYBOARD
IF inputNumber = 3
THEN SEND "Your number is a 3" TO DISPLAY
ELSE IF inputNumber = 4
   THEN SEND "Your number is a 4" TO DISPLAY
   ELSE SEND "Your number is not a 3 or a 4" TO DISPLAY
   END IF
```

! Key points

- The **IF** statement is used to create a decision structure, which allows a program to have more than one path of execution.
- The **IF** statement causes one or more statements to execute only when a Boolean expression is true.
- An **IF-ELSE** statement will execute one block of statements if its condition is true, or another block if its condition is false.

Task

2 Use this book and other sources such as the Internet to research how to identify and correct errors in algorithms.

3 Use this book and other sources such as the Internet to research how standard algorithms (bubble sort, merge sort, linear search, binary search) work.

Chapter review

In this chapter we have explored pseudo-code and how to use it to show program flow and decision making.

We also explored the importance of syntax and how to comment on your code.

Remember, before tackling any computer science task or examination question on this topic you must:

- understand and be able to explain what algorithms are used for
- be able to interpret and write algorithms in pseudo-code
- understand and be able to explain and create algorithms to solve a particular problem, making use of programming constructs (such as sequence, selection, iteration)
- understand and be able to explain and apply the purpose of a given algorithm and how an algorithm works
- understand how to determine the correct output of an algorithm for a given set of data
- understand and be able to use appropriate conventions in pseudo-code
- understand and be able to explain the benefit of producing programs that are easy to read and be able to use techniques such as comments and descriptive names for variables, constants, and subprograms alongside indentation to improve readability and to explain how the code works
- be able to use sequencing, selection and iteration constructs in your programs
- understand and be able to apply structural components of a program including variable and type declarations and sequences, selection, iteration, data structures, and subprograms.

2 Using flowcharts

Each chapter clearly references the understanding and skills students will need to practise and exhibit in their exams

Specification references

You should:

1.1.1 understand what an algorithm is, what algorithms are used for and be able to interpret algorithms (flowcharts, pseudo-code, written descriptions, program code)

how to create an algorithm to icular problem, making use of g constructs (sequence, selection, iteration) and using appropriate conventions (flowchart, pseudo-code, written description, draft program code)

1.1.3 understand the purpose of a given algorithm and how an algorithm works

1.1.4 understand how to determine the correct output of an algorithm for a given set of data

2.4.1 understand how to write code that accepts and responds appropriately to user input

Key points clarify significant information for students to be able to process and recall easily

Key points

- A **flowchart** is a diagram representation of an algorithm.
- For the examination, you will need to be able to interpret flowcharts.
- Flowcharts are a graphical method of designing programs.
- A well-drawn flowchart is easy to read.

Key terms will help students develop computing language skills to facilitate greater subject understanding

Key term

For the examination, you will need to be able to interpret flowcharts.

Key point

In a flowchart the lines express the order of execution.

There are a lot of different design procedures and techniques for building large software projects. The technique discussed in this chapter, however, is for smaller coding projects and is referred to by the term 'top down, structured **flowchart** methodology'. We will explore how to take a task and represent it using a flowchart. A flowchart puts the sentences from a sequence into shaped boxes. The shapes indicate the action.

You will know from the last chapter that a sequence is where a **set of instructions** or actions are **ordered**, meaning that each action follows the previous action.

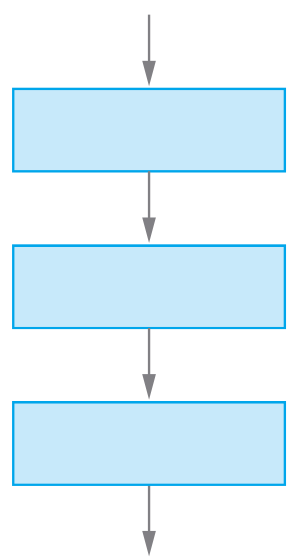


Figure 2.1 A sequence

2

General rules for flowcharts

- All symbols of the flowchart are **connected by flow lines** (these must be arrows not lines to show direction).
- Flow lines **enter the top** of the symbol and **exit out the bottom**, except for the Decision symbol, which can have flow lines exiting from the bottom or the sides.
- Flowcharts are drawn so flow generally goes from the **top to the bottom** of the page.
- The beginning and the end of the flowchart is indicated using the Terminal symbol.

Let's look at a simple sequence. Say we want to calculate A plus B, where A = 200 and B = 400.

Start
A = 200 B = 400
Add = 200 + 400
Output = 600
End

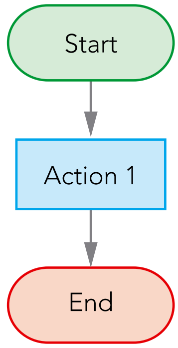


Figure 2.2 A simple flowchart

Let's look at a another sequence, for example the sequence you carry out each morning in the bathroom. This sequence could be:

- Brush teeth
- Wash face
- ir.

n see, **sequences** are a useful tool for s nd in what logical order, but each step h', needs to be defined in more detail to b

Key points

- Flowcharts must have flow lines with arrows to show the order of execution.
- An algorithm is a sequence of steps that can be followed to complete a task.
- A sequence is where a set of instructions or actions are ordered, meaning that each action follows the previous

Practice tasks will help build students' coding, programming and problem-solving skills

Task

1 Produce a sequence to show how to brush your teeth.

Practice questions will help build students' coding, programming and problem-solving skills

Questions

1 What is a Terminator?
2 What is a Sequence?
3 What is an Input/Output?

Each chapter concludes with a review of key information to assess and consolidate topic knowledge and understanding

Chapter review

In this chapter we built upon the last chapter to explore sequences in more detail and established how we can show these using flowcharts.

We looked at the basic elements of flowcharts and introduced the concept of decisions and how these can be represented.

Remember, before tackling any computer science task or examination question on this topic you must:

- understand what an algorithm is and what algorithms are used for, and be able to interpret algorithms in the form of flowcharts
- use a systematic approach to problem solving and algorithm creation representing those algorithms using flowcharts;
- understand and be able to use appropriate conventions in flowcharts

EDEXCEL GCSE COMPUTER SCIENCE STUDENT BOOK

These sample chapters are taken from the forthcoming **Edexcel GCSE Computer Science Student Book**.

Build student confidence and ensure successful progress through GCSE Computer Science. Our expert author provides insight and guidance for students to meet the demands of the new Edexcel specification, with challenging tasks and activities to test the computational skills and knowledge required for success in the assessment, and advice for successful completion of the non-examined assessment.

- Builds students' knowledge and confidence through detailed topic coverage and explanation of key terms.
- Develops computational thinking skills with practice exercises and problem-solving tasks.
- Instils a deeper understanding and awareness of computer science, and its applications and implications in the wider world.
- Helps monitor progression through GCSE with regular assessment questions, that can be further developed with supporting Dynamic Learning digital resources.

First teaching from September 2016

Author:
Steve Cushing is a well-respected and widely published author for secondary Computing, with examining experience.

Textbook subject to change based on endorsement review.

Visit **www.hoddereducation.co.uk/ComputerScience/GCSE/Edexcel** to pre order your class sets or to sign up for Inspection Copies or eInspection Copies.

ALSO AVAILABLE

Dynamic Learning

Edexcel GCSE Computer Science Dynamic Learning is an online subscription solution that supports teachers and students with high quality content and unique tools. Dynamic Learning incorporates Teaching and Learning resources, Whiteboard and Student eTextbook elements that all work together to give you the ultimate classroom and homework resource.

Sign up for a free trial – visit: **www.hoddereducation.co.uk/dynamiclearning**

HODDER EDUCATION
www.hoddereducation.co.uk