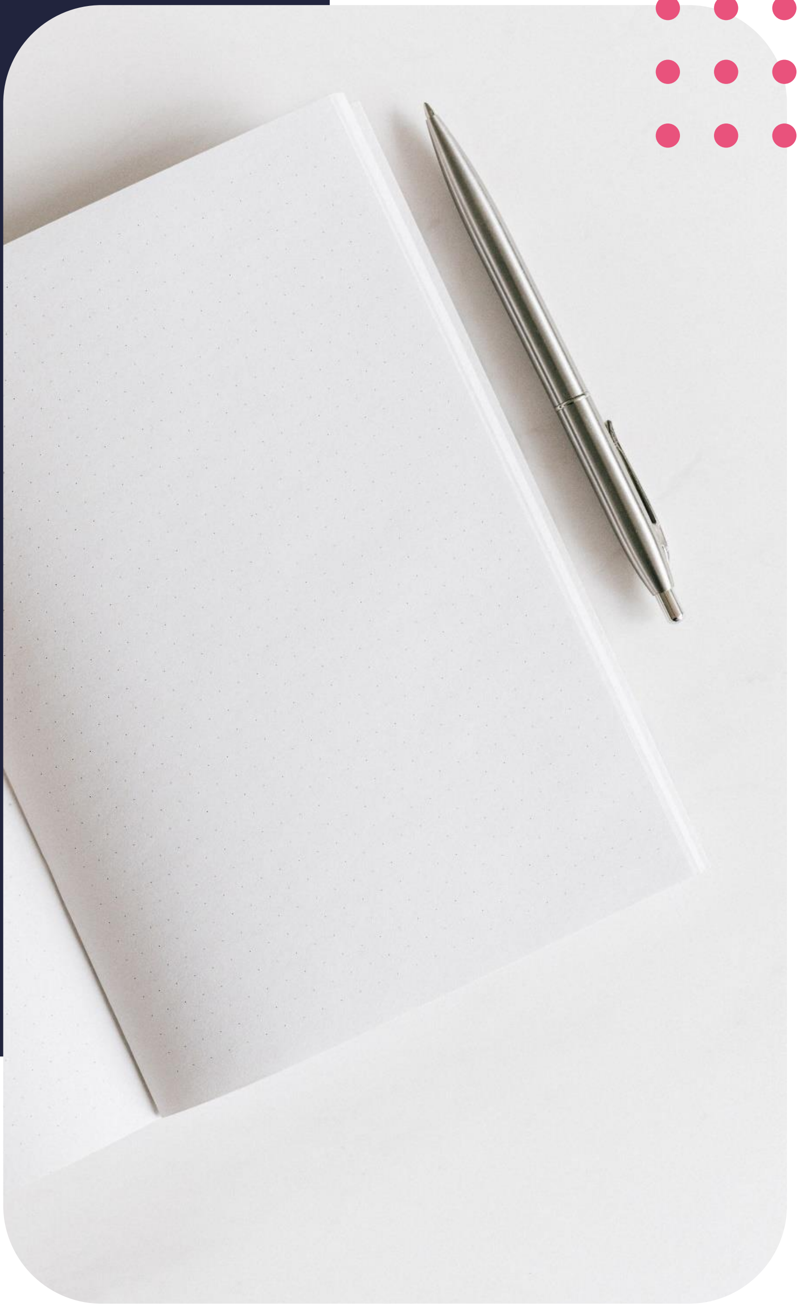
Diploma in Computer Science

Formulating Solutions

Contents



Lesson outcomes

By the end of the lesson, you should be able to:

- Explore Pseudocode as a way of representing Algorithms
- Appreciate flowcharts as an additional way of representing algorithms
- Identify the symbols used in flowcharts
- Explore problems and create pseudocode and flowcharts from them

Relevance of pseudocode

As we discussed in our previous lesson, a computer program starts its life in the form of a stated problem. This problem and all its complexity is summarised into a small relatively simple statement. For us to be able to understand the statement, we need to break it down as we demonstrated in our previous lesson. This is still not enough though, breaking the problem down like that is only enough for us to figure out what is going on. The computer still won't know what to do with our strategies in that state. This is where the next stage comes in: - Pseudocode.

What is pseudocode

The word pseudocode is actually two words joined together; pseudo, Greek for false, and code, for, well, code as we know it. So, we're pretty much writing wannabe code here!

Pseudocode's main purpose is to show how an algorithm, or, our proposed solution should work, think of it as an army's strategy before the actual war; it describes how the soldiers will move into the enemy's land, who will take what position and what weapons they will use.

Similarly, using pseudocode, we can demonstrate where a particular mechanism or technique will be used in the program. Pseudocode is written in English statements. It's not really proper English though, it's just a bunch of terse statements that describe a concept. It can be described as an algorithm in the form of informative text and annotations written in plain English, minus the strict syntax of programming languages. This means that pseudocode cannot be compiled and run by any computer. remember when we said we are going to be writing wannabe code?!

Why do you need pseudocode?

You may be wondering why you even need to write fake code when you can just write actual code and save yourself time and effort. Turns out, there are quite a number of reasons why you might want to do so. It is actually industry practice to write this fake code first!

If you're working on a project for a client, for example, you can't show the client those cryptic lines of code that you will be writing. You need something that is easy to understand and follow to bring the client up to speed with the work that you will be doing. Pseudocode is much more readable to someone who does not know how to write code than actual code. Even complex programs like assembly language can be written in pseudocode!

Because pseudocode doesn't crucify you with the strict commandments of programming languages, you are free to express your ideas in plain English, this gives you the leeway to lay out your thoughts. Because of this, writing pseudocode is one of the best ways to start implementation of your algorithm. Pseudocode will bridge the gap between the algorithm and the flowchart. A flowchart is also one of the crucial parts of program development process, and we are going to look at it shortly.

Pseudocode spells out what a line of code, or a portion of code should do. This makes the process of writing code much, much easier as you will be having a reference.it also serves as documentation of sorts. If you are working on a project as a team, it's easier for a team member to look at your pseudocode and understand how your program works, rather than looking at the code itself. As we progress with the course, you will see the importance of documentation. A quick definition here, documentation in the context of software is written text or illustrations that accompany software, explaining how the software operates, how to use the software and all the information pertaining to the software development cycle.

Long story short, the goal of pseudocode can be condensed into one word. Clarity. Remember the Travelling Salesman problem that we looked at in the previous lesson? Imagine just jumping into an IDE (integrated development environment, defined in lesson 1) and writing the code from scratch! Where do you even start? What would you even write? Algorithms almost always need you to start with pseudocode, unless you're a wizard, or you're writing dead simple programs like adding 2 numbers.

Pseudocode is universal. If you planned to implement your algorithm in java and then decide you now want to do it in C, your pseudocode is still pretty much valid, as long as the programming language that you choose is cut out for the task.

Laying out your thoughts

When writing pseudocode, you're basically writing down your thoughts. For your thoughts to be usable by you, or anyone else at a later stage, clarity is important. This CANNOT be stressed enough! In pseudocode, we try to make every sentence say just one thing. This is especially important when you transcribe the pseudocode into actual code; you will be less likely to run into problems of overly complex code that cannot be translated into one block of code. Also, your code should not be too general, for example, when you are writing pseudocode for a system that has multiple sensors, you can't simply just say "turn on the sensor". You need to specify which sensor to switch on, as well as any parameters that may be part of the procedure.

Writing pseudocode

While pseudocode is not aligned to Programming standards it is often a good idea to align your pseudo code to the syntax of the language that you intend to code in.

This helps you to shorten the time that you need to transcribe the pseudo code into actual code. it also helps you to capture the logic and flow of the solution

Analysing the problem

In lesson 6, we looked at the formulation of problems. The purpose of formulating a problem is to give you a starting point when writing your code. You have a clearer idea of what needs to be done. Writing pseudocode breaks down our problem into the step-by-step instructions that we required to solve the problem.

The first step in writing pseudo code is to arrange the tasks in sequence. this helps follow how the program goes from one point to another. This is especially useful for complex tasks where we have to break down the problem into small portions. It is also useful to start with a statement that states the goal that we want to achieve in the pseudocode. For example, if we are writing a program that checks if a number is a palindrome or not, we might add the statement at the top “this program will check if a number is a palindrome or not.” Straight away we already know what the program does without even reading the rest of the code.

A few house-rules

While pseudocode does not follow the conventions syntax and semantics of programming languages, it is generally good practice to follow a few rules that we are going to look at in order to write clear pseudo code

We have already mentioned that you should state the goal of the pseudocode at the very top of the document so that anyone reading your pseudocode will straight away know wat the program is about.

While naming any variables in your pseudocode use naming conventions for example if you are writing a programme that adds 2 numbers you might want to name your variables as number1 and number2 and so on. In module 2 we're going to take a closer look at what variables are but just in case you are wondering if variable is ais a space in memory that is reserved for storage of certain predefined data, like in our example here the variable number1 reserves an area of memory for storage of that number. Of course, this variable will be used in pseudocode as it would in an actual program, but this also helps in the transcription of your pseudocode into actual code.

If there are any loops, decisions and such in your programme which would be indented by an integrated development environment, it's a good idea to also indent these in your pseudocode to make it readable and easier to follow.

Try to use conventional programming structures such as if-then, while-do, case and so on. It is however worth noting that you should keep your coat simple to understand and not make it to programmatic remember it is supposed to be understandable even for a layman such as your client so you should try to keep away from writing too many technical terms.

Make sure your pseudo code does not leave anything to assumption that is it should cover every little bit of what your programme is going to do. Remember the pseudocode should not be ambiguous but should clearly state every step of the algorithm.

There aren't that many rules inside the code and they can be summarised in this simple sentence; keep it simple, keep it concise, use proper naming conventions, use indentations and white spaces, use control structures, while trying to avoid making your pseudo code abstract and too generalised.

Here are some of the widely recognised terms used in pseudocode.

Input- is used when the user is inputting something.

Output- is used when output appears on the screen.

While- is used to indicate a loop. A loop is when a section of code is repeated over and offer until a certain condition is met.

For- for is used for a counting loop.

if- then- else- is used to denote at decision. It is used when a choice has to be made.

Repeat- until- is also a type of loop where a block of code is repeated until a certain condition is met. it's not very different from the while loop.

Remember when we say it some blocks of code need to be indented? When you are writing code using while, for, repeat, and if, these blocks of code need to be indented to show that you are using a selection or an iteration structure.

Another good practice is to capitalise these key commands. This ensures that they are never confused for anything else, although that probably wouldn't happen as these words are reserved specifically for that. For example, you might want your pseudocode to read "IF number1 is less than 2 THEN output "Hello"" rather than "if number1 is less than 2 then output "Hello"".

Another good practice is to organise your pseudocode in sections. This ensures that you can clearly see where one part of the algorithm executes a certain action. If ever there's a problem with the code, you can quickly locate a block of code that is incorrect.

It's also worth remembering that since we are writing fake code, we can get away with doing whatever we want with it! We can break a few rules just to get it to make sense, as long as it can be transcribed into real code!

Flow charts

Another method used to represent an algorithm before writing code is by using flowcharts. Flowcharts help us understand the flow of an algorithm. Flowcharts are often used in conjunction with pseudocode, and this helps us make sure that no step is missed during the planning phase. It would be very disappointing to write code, then when it's time to run it, you realise that you left out an entire step and you have to go back to the drawing board! It wastes a lot of time and if it is a commercial project it will cost a lot too!

flow charts differ from pseudo code in that they follow a bit stricter rules and follow quite a number of conventions let's look at some off the symbols used in a flow chart

select the start and stop points of a sequence are denoted by a rounded rectangle.

A process comprising a bunch of instructions, or commands is denoted by a rectangle

when you need to make a decision, which is either a yes or a no do you use a rhombus.

Input or output, that is data that is received by the computer or a signal that is sent out of the computer, is denoted by a parallelogram.

If you are jumping from one point in the sequence to another you use a connector in the form of a circle. For any other connections between symbols, you use arrows.

In case you have all this mixed up, here's a table that summarises everything.

Name	Symbol	Usage
Start/stop or end	⬬	The beginning and end of the sequence/program
Process	■	An instruction
Decision	◆	A decision typically an *if* statement
Input/output	▱	Data being received by a computer or data produced by the computer
Connector	●	A jump from one point in the sequence to another
Direction of flow	⇄	Direction of the flow of the program

Looking at a flowchart gives you a general idea of where the program starts, the processes that it executes and the output that it produces. It's a pretty good place to start before writing pseudocode.

Remember in previous lessons when we said a computer is a state machine? A flowchart shows each of the states that the computer goes through to its final state.

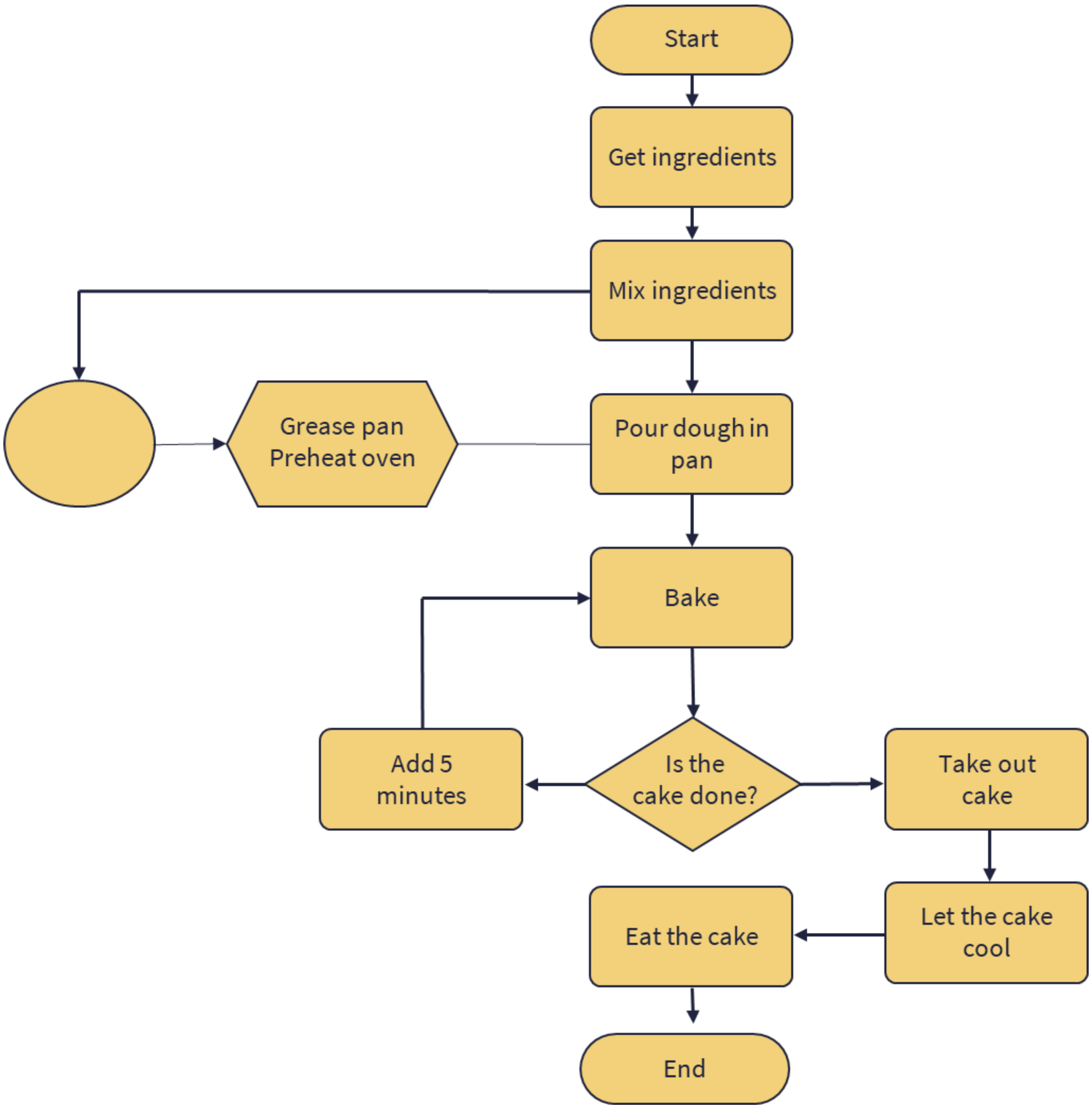
Examples

Now that we have covered all the basics, it's time to get our hands dirty with some algorithms. Let's go back to our example from lesson 3, baking a cake.

we start by getting our ingredients, cocoa, 3 eggs, 1 tablespoon of baking soda, 1 cup sugar, raisins, and flour.

Step 1. Mix the ingredients in a bowl until the mixture is consistent. While the mixer is running, grease the pan and preheat the oven. When the mixture is consistent, pour it into the pan and put the pan into the oven. Bake for 20 minutes. Check with a fork if the cake is done. If it is not done, bake for 5 more minutes, otherwise bake for 5 more minutes. If the cake is done, take out of the oven and let it cool. When the cake is cool, cut it and eat it!

Let's now take these step by step instructions and draw a flowchart.



You can try to make your own flowchart in Microsoft word or using dedicated flowchart software such as lucid chart, which I used to make this flowchart. Now that we have our flowchart, we have a reference for writing pseudocode.

```
This program bakes a cake
START
Get ingredients
PROCEDURE mix ingredients
        Get bowl
Add 2 cups flour
        Add 1 cup sugar
Add raisins
        Add cocoa
        Add eggs
        Add baking powder
        Turn on mixer
PROCEDURE preparation
        Grease pan
        Preheat oven
PROCEDURE bake
        Pour dough into pan
        Put in oven
        Bake 20 mins
                IF cake is ready
                THEN take out of oven
                Cool and eat
                ELSE add 5 minutes
                Bake
END
```

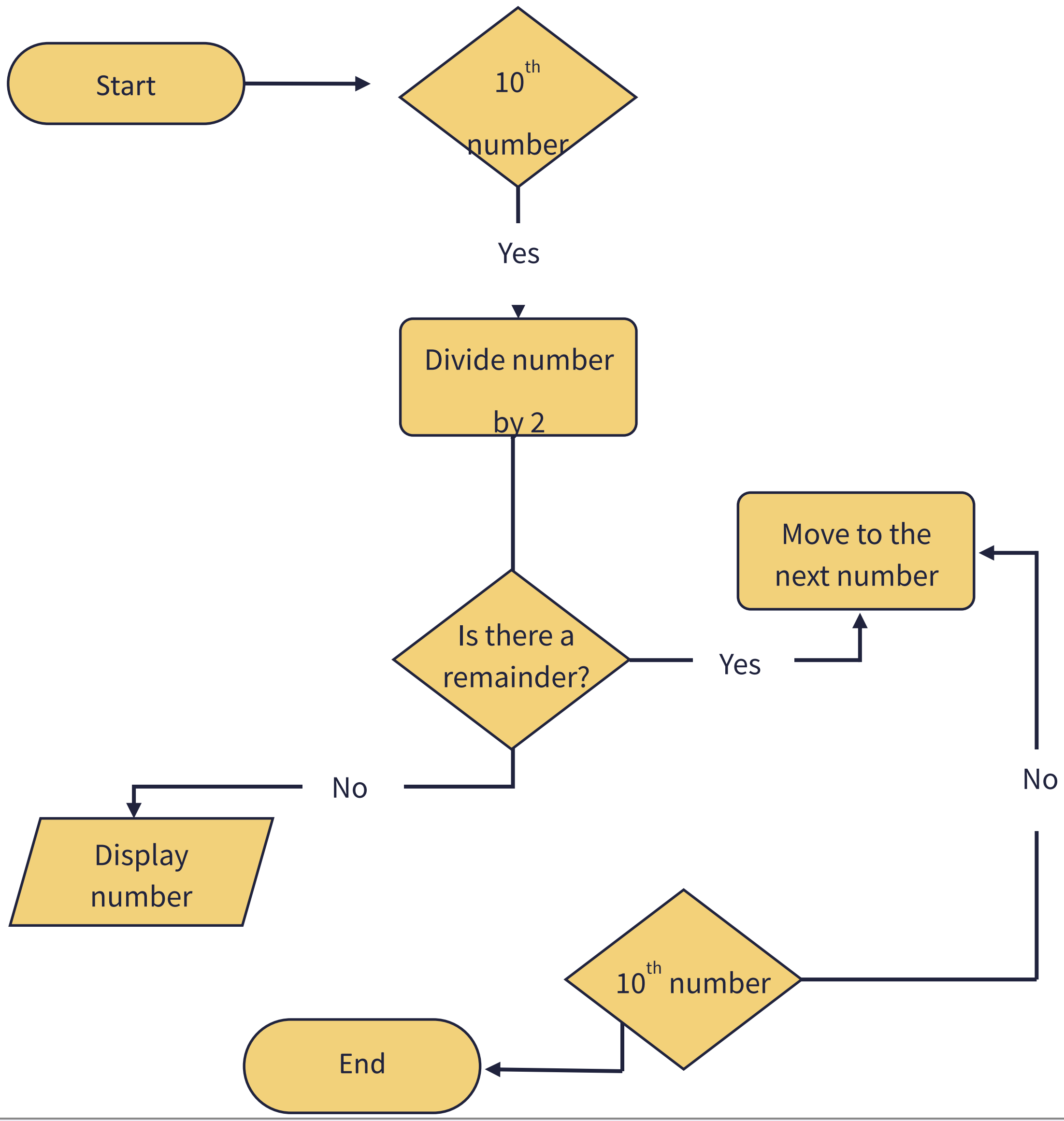
Notice that our pseudocode almost looks like code, but no compiler on earth can ever make sense of that code. This highlights perfectly, the point that pseudocode is more for humans to understand the program than for the machine to make sense of it.

Notice also that we were indenting whenever we were within a procedure. Just by glancing at the pseudocode, you can easily see where each procedure starts and ends. The decision in our pseudocode is part of the baking procedure, and to make it clear, it is further indented. This helps us avoid confusion as we can clearly see that we are still within the baking procedure.

Now that we have the hang of things, let's look at an example that is a little harder. This time, we want to write pseudocode for a program that asks for 10 numbers from the user, then looks at the numbers and displays only even numbers. We first have to break down the problem and see exactly how we are going to go about it.

Let's start off with the flowchart.

We have 7 distinct steps drawn out from the problem stated. This is not set in stone, however, you might come up with more, or less steps, depending on how you choose to solve the problem.



```
This programme reads numbers from the user and displays only even numbers
Start
PROCEDURE read numbers
        WHILE recorded numbers <10     DO:
Ask user for number
Store number
PROCEDURE find even numbers
        WHILE recorded numbers <10     DO:
Fetch number
Divide number by 2
        IF number has remainder
        THEN ignore number
        ELSE display number
END
```

Our pseudocode example looks clean and neat, and we can easily follow what is going on. You can even add “end procedure” at the end of every procedure to make it even easier to follow. Remember, in pseudocode, you set the rules, and you can do whatever you want, as long as your code stays withing the house rules and doesn’t become difficult to transcribe into actual code.

It is also a good idea to add the word END whenever you exit a procedure, if statement or any other control structure.

Now that we have covered the basics, we will now add a little more flesh to the keywords used when writing pseudocode.

INPUT: is used for data obtained from the user through typing or through an input device.

READ / GET: is used when reading data from a data file.

PRINT, DISPLAY, SHOW: This will show your output to a screen or whatever output device that you will be using in the system that you will be designing.

COMPUTE, CALCULATE, DETERMINE: is used to calculate the result of an expression.

SET, INIT: is used to initialise values. A quick definition here, initialising is setting a variable to an initial value, such as setting the value of a variable that will be used for counting to an initial value of 0.

INCREMENT, BUMP: is used when increasing the value of a variable

DECREMENT: is used when reducing the value of a variable

Here are a few problems that you can try on your own:

Problem 1: a school asks you to design a system that records each student’s surname and average mark. The system should be able to record average marks for 30 students and display these marks in a list. The system should then calculate the class average mark and display this below the class list. Write pseudocode and create a flowchart for this system.

You work for a software company and they ask you to create a module for their new program. The program should count the number of words in a document, then display the result to the user.

And for one that’s a bit more challenging:

Write pseudocode for a programme that asks the user for the description of a shape. Based on certain keywords, it will identify the shape, then ask the user for dimensions. The program then proceeds to calculate the surface area and/or volume of the identified shape, then display the name of the shape as well as the results of the calculation to the user.

Don’t worry, we are going to carry over our examples to the next lesson and develop them into algorithms, then into code!

References

Pseudocode: Definition & Examples Video (2020). Pseudocode: Definition & Examples - Video & Lesson Transcript | Study.com. [online] Study.com. Available at: https://study.com/academy/lesson/pseudocode-definition-examples-quiz.html

Stem.org.uk. (2017). Introduction to Pseudocode | STEM. [online] Available at: https://www.stem.org.uk/resources/elibrary/resource/404766/introduction-pseudocode

BBC Bitesize. (2020). Flowcharts - Designing an algorithm - KS3 Computer Science Revision - BBC Bitesize. [online] Available at: https://www.bbc.co.uk/bitesize/guides/z3bq7ty/revision/3

GeeksforGeeks. (2018). How to write a Pseudo Code? - GeeksforGeeks. [online] Available at: https://www.geeksforgeeks.org/how-to-write-a-pseudo-code/

Ngunyi Macharia (2018). How to write Pseudocode: A beginner’s guide - Noteworthy - The Journal Blog. [online] Medium. Available at: https://blog.usejournal.com/how-to-write-pseudocode-a-beginners-guide-29956242698

Ngunyi Macharia (2018). How to write Pseudocode: A beginner’s guide - Noteworthy - The Journal Blog. [online] Medium. Available at: https://blog.usejournal.com/how-to-write-pseudocode-a-beginners-guide-29956242698

Pasko, R. and Bauer, M. (n.d.). Problem Solving Basics and Computer Programming Solving Problems with Solutions Requiring Sequential Processing. [online] Available at: http://www.cs.iit.edu/~cs104/ProblemSolving.pdf

Problem Solving. (n.d.). [online] Available at: http://metalab.uniten.edu.my/~hazleen/CSEB113/ch2.pdf

Rice.edu. (2020). FOR3_1.html. [online] Available at: http://www.owlnet.rice.edu/~ceng303/manuals/fortran/FOR3_3.html