Topic: 2.1.2 Pseudocode and flowcharts

Introduction to Pseudocode

This section covers the use of pseudocode in the production of algorithms. Students should use standard computing text books to find out information on the features of programming languages (high level and low level), interpreters, compilers, assemblers, user documentation and technical documentation.

No specific programming language is referred to; development of algorithms using pseudocode uses generic descriptions of looping, branching, data manipulation, input/output, totaling and counting techniques.

The section is broken down into four areas:

1. Description of common pseudocode terms
2. Writing algorithms using pseudocode
3. Finding errors in sections of pseudocode
4. Exercises

Understand and use assignment statements

Assignment
An assignment is an instruction in a program that places a value into a specified variable.
Some typical assignments are:

```
TheLength ← 20.5
TheUsersName$ ←  “Charlie”
TheArea ← TheWidth * TheLength
TotalCost ← LabelledCost + 15
Counter ← Counter + 1
```

Note that the last example is a common method used to increment the value of a variable. It could be read as:

“The new value of Counter is its existing value plus one"

What is an algorithm?

It is a procedure for solving a problem in terms of the actions to be executed and the order in which those actions are to be executed. An algorithm is merely the sequence of steps taken to solve a problem. The steps are normally "sequence," "selection," "iteration," and a case-type statement.

Topic: 2.1.2 Pseudocode and flowcharts

The "selection" is the " **if ….. then ….. else ….. endif**" statement, and the iteration is satisfied by a number of statements, such as the " **for … to … next, while … endwhile** and **repeat … until** " while the case-type statement is satisfied by the "**case of ….. otherwise ….. endcase**" statement.

Relational operators, e.g. =, <, <=, >, >= and <>

Relational operators are used in the format: [Expression] [Operator] [Expression] and will always return a Boolean (True or False) value.

Relational operators are typically used with the IF selection and also within conditional loops (REPEAT-UNTIL or WHILE-WEND).

In the following table, the variables a and name$ have the following assignments:

```
a←3+5
name$← "JAMES"
```

Operator	Meaning/purpose	Example	Result
=	Equal – tests if two expressions are identical	IF a=8 IF name$="JANET" IF a=14	True False False
<	Less Than – tests if the first expression is less than the second	IF a<8 IF name$<"JANET" IF a<14	False True True
>	Greater Than – tests if the first expression is greater than the first	IF a>8 IF name$>"JANET" IF a>14	False False False
<>	Not Equal – tests if two expressions are different	IF a<>8 IF name$<>"JANET" IF name$<>"JAMES" IF a<>14	False True False True
<=	Less Than or Equal	IF a<=8 IF name$<="JANET" IF a<=14	True True True
>=	Greater Than or Equal	IF a>=8 IF name$>="JANET" IF a>=14	True False False

Topic: 2.1.2 Pseudocode and flowcharts

Boolean operators AND, OR and NOT

AND and OR

The AND and OR operators always return a Boolean result and are used in the format:

```
[Boolean] [Operator] [Boolean]
```

The following 'truth' table summarises the result of the Boolean operations:

Values Results

Values		Results	
X	Y	X AND Y	X OR Y
True	True	True	True
True	False	False	True
False	True	False	True
False	False	False	False

NOT

The NOT operator reverses the result of the Boolean expression and is used in the format:

```
NOT [Boolean]
```

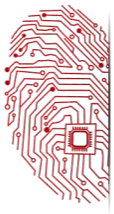
The following truth table summarises the NOT operation:

X	NOT X
True	False
False	True

Examples of Boolean 'logic'

Consider the following algorithm, which is used to monitor a printer and display its output via an LCD display in the front panel:

```
IF NOT(PaperTrayEmpty) AND (FilesWaiting > 0) THEN
      OUTPUT "PRINTING…"
ELSE
      OUTPUT "PLEASE ADD PAPER"
END IF
```



Topic: 2.1.2 Pseudocode and flowcharts

Branching

There are two common ways of branching:

1. **case of ….. otherwise ……. endcase**
2. **if ….. then ….. else ….. endif**

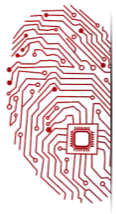
case of	if … then
case **number** of	**if** number = 1 **then** x ← x + 1
1: **x ← x + 1**	**else if**
2: **y ← y + 1**	number = 2 **then** y ← y + 1
otherwise print **"error"**	**else print** "error"
endcase	**endif**
	endif

SELECT-CASE

This selection method is used if there are **MORE THAN TWO** possible outcomes to a test:
Creating a Select-Case statement is simple to do. The next program will prompt the user to select the key A-D and the program will respond by telling the user what key was entered.

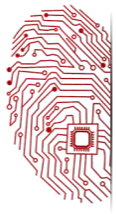
We will create a Select-Case statement for the A-D keys entered.

```
DIM KeyPressed AS STRING

CLS

PRINT
PRINT

INPUT "Please Enter A Key (A,B,C,D): ", KeyPressed

KeyPressed = UCASE$(KeyPressed)

PRINT

SELECT CASE KeyPressed
  CASE "A"
     PRINT "A Was Entered"
  CASE "B"
     PRINT "B Was Entered"
  CASE "C"
     PRINT "C Was Entered"
  CASE "D"
     PRINT "D Was Entered"
```



Topic: 2.1.2 Pseudocode and flowcharts

```
  CASE ELSE
     PRINT "Some Other Key Was Entered"
END SELECT

DIM Score AS INTEGER

CLS

PRINT
PRINT
INPUT "Enter The Test Score: ", Score

PRINT

SELECT CASE Score
   CASE IS >= 97
     PRINT "Grade A+"
   CASE 93 TO 96
     PRINT "Grade A"
   CASE 90 TO 92
     PRINT "Grade A-"
   CASE 87 TO 89
     PRINT "Grade B+"
   CASE 83 TO 86
     PRINT "Grade B"
   CASE 80 TO 82
     PRINT "Grade B-"
   CASE 77 TO 79
     PRINT "Grade C+"
   CASE 73 TO 76
     PRINT "Grade C"
   CASE 70 TO 72
     PRINT "Grade C-"
   CASE 67 TO 69
     PRINT "Grade D+"
   CASE 63 TO 66
     PRINT "Grade D"
   CASE 60 TO 62
     PRINT "Grade D-"
   CASE ELSE
     PRINT "Fail"
END SELECT
```



Topic: 2.1.2 Pseudocode and flowcharts

IF-THEN-ELSE-ENDIF

This selection method is used if there are **TWO** possible outcomes to a test:

```
| IF { conditional statement } THEN |
|    { statement 1 }                |
| ELSE                              |
|    { statement 2 }                |
| ENDIF                             |
```

Example:

```
| if X > 10 then            |
|    PRINT X; " is > 10"    |
| else                      |
|    PRINT X; " is <= 10"   |
| endif                     |
```

```
Dim count As Integer = 0
Dim message As String

If count = 0 Then
   message = "There are no items."
ElseIf count = 1 Then
   message = "There is 1 item."
Else
   message = "There are " & count & " items."
End If
```

Topic: 2.1.2 Pseudocode and flowcharts

Loops

Iteration

Iteration is a control structure in which a group of statements is executed repeatedly – either a set number of times or until a specific condition is True.
There are three common ways of performing a looping function:

1. **for … to … next,**
2. **while … endwhile**
3. **repeat … until**

The following example input 100 numbers and finds the total of the 100 numbers and outputs this total. All three looping techniques are shown:

for … to	while … endwhile	repeat … until

`for count ← 1 to 100`
`  input number`
`  total ←total + number`
`  next`
`  print total`
	`while count < 101`
	`  input number`
	`  total ← total+ number`
	`  count ← count + 1`
	`endwhile`
	`  print total`
		`repeat`
		`  input number`
		`  total ← total+number`
		`  count ← count+1`
		`until count=100`

FOR-NEXT

This is an unconditional loop in which the number of repetitions is set at the beginning.

```
FOR X = 1 TO 5
      Answer = X*3
      OUTPUT X, Answer
NEXT
```

Sample code:

```
|10 sum = 0                                              |
|20 FOR x = 1 to 10                                      |
|30 print x                                              |
|40 input "enter a number";n                             |
|50 sum = sum + n                                        |
|60 NEXT x                                               |
|70 print "The sum of the numbers you gave is";sum       |
```

Topic: 2.1.2 Pseudocode and flowcharts

WHILE-ENDWHILE

This is a conditional loop, which has a test at the start and repeats until the condition is false:

```
X = 0
WHILE X < 6 DO
      X = X + 1
      Answer = X*3
      OUTPUT X, Answer
ENDWHILE
```

Sample code:

```
| 10 sum = 0                                              |
| 20 x = 1                                                |
| 30 WHILE x < 11                                         |
| 40    print x                                           |
| 50    input "enter a number";n                          |

| 60    sum = sum + n                                     |
| 70    x = x + 1                                         |
| 80 WEND                                                 |
| 90 print "The sum of the numbers you gave is";sum        |
```

REPEAT-UNTIL

This is a conditional loop, which has a test at the end and repeats until the condition is true:

```
X = 0
REPEAT
      X = X + 1
      Answer = X*3
      OUTPUT X, Answer
UNTIL X > 4
```

Common pseudocode terms:

1.1) Counting

Counting in 1s is quite simple; use of the statement **count ← count + 1** will enable counting to be done (e.g. in controlling *a repeat loop*). The statement literally means*: the (new) count =the (old) count + 1.*

It is possible to count in any increments just by altering the numerical value in the statement (e.g. count ← count – 1) will count backwards.

Topic: 2.1.2 Pseudocode and flowcharts

1.2) Totaling

To add up a series numbers the following type of statement should be used:

```
total ← total + number
```

This literally means *(new) total = (old) total + value of number.*

1.3) Input/output

Input and output indicated by the use of the terms **READ number, PRINT total, PRINT "result is" x** and so on.

Writing algorithms using pseudocode

The following five examples use the above pseudocode terms. These are the same problems discussed in section 3.1 using flow charts – both methods are acceptable ways of representing an algorithm.

2.1 Example 1
A town contains 5000 houses. Each house owner must pay tax based on the value of the house. Houses over $200 000 pay 2% of their value in tax, houses over $100 000 pay 1.5% of their value in tax and houses over $50 000 pay 1% of their value in tax. All others pay no tax.
Write an algorithm to solve the problem using pseudocode.

```
for count ← 1 to 5000
      input house
      if house > 50 000 then tax ← house * 0.010
      else if house > 100 000 then tax ← house * 0.015
      else if house > 200 000 then tax ← house * 0.020
            else tax ← 0
            print tax
next
```

Notes:
(1) a **while** loop or a **repeat** loop would have worked just as well
(2) the use of **endif** isn't essential in the pseudocode

Topic: 2.1.2 Pseudocode and flowcharts

For example,

```
count ← 0
while count < 5001
      input house
      if house > 50000 then tax ← house * 0.010
            else if house > 100 000 then tax ← house * 0.015
            else if house > 200 000 then tax ← house * 0.020
                  else tax ← 0
            endif
            endif
      endif
      print tax
      count ← count + 1
endwhile
```

EXERCISE: Re-write the above algorithm using a **repeat** loop and modify the **if … then … else** statements to include both parts of the house price range.
(e.g. **if** house > 50000 and house <= 100000 **then** tax = house * 0.01)

2.2 Example 2

The following formula is used to calculate n: n = x * x/(1 – x)
The value x = 0 is used to stop the algorithm. The calculation is repeated using values of x until the value x = 0 is input. There is also a need to check for error conditions. The values of n and x should be output.

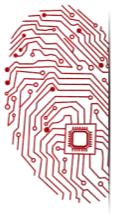
Write an algorithm to show this repeated calculation using pseudocode.

NOTE: It is much easier in this example to input x first and then loop round doing the calculation until eventually x = 0. Because of this, it would be necessary to input x twice (i.e. inside the loop and outside the loop). If input x occurred only once it would lead to a more complicated algorithm.

(Also note in the algorithm that <> is used to represent ≠ "not equals to").

A **while** loop is used here, but a **repeat** loop would work just as well.

```
input x
while x <> 0 do
      if x = 1 then print "error"
      else n ← (x * x)/(1 - x)
            print n, x
      endif
      input x
endwhile
```



Topic: 2.1.2 Pseudocode and flowcharts

2.3 Example 3

Write an algorithm using pseudocode which takes temperatures input over a 100 day period (once per day) and output the number of days when the temperature was below 20C and the number of days when the temperature was 20C or above.

(NOTE: since the number of inputs is known, a **for … to** loop can be used. However, a **while** loop or a **repeat** loop would work just as well).

```
total1 ← 0 : total2 ← 0
for days ← 1 to 100
      input temperature
      if temperature < 20 then total1 ← total1 + 1
            else total2 ← total2 + 1
      endif
next
print total1, total2
```

This is a good example of an algorithm that could be written using the **case** construct rather than **if … then … else**. The following section of code replaces the statements ***if*** *temperature < 20* ***then …… endif****:*

```
case temperature of
1: total1 = total1 + 1
2: total2 = total2 + 1
endcase
```


03-111-222-ZAK | OlevelComputer AlevelComputer | @zakonweb | zak@zakonweb.com | www.zakonweb.com

Topic: 2.1.2 Pseudocode and flowcharts

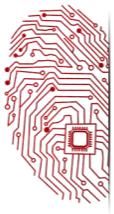
2.4 Example 4

Write an algorithm using pseudocode which:

- inputs the top speeds of 5000 cars
- outputs the fastest speed and the slowest speed
- outputs the average speed of all the 5000 cars

(NOTE: Again since the actual number of data items to be input is known any one of the three loop structures could be used. It is necessary to set values for the fastest (usually set at zero) and the slowest (usually set at an unusually high value) so that each input can be compared. Every time a value is input which > the value stored in fastest then this input value replaces the existing value in fastest; and similarly for slowest).

```
fastest ← 0: count ← 0
slowest ← 1000
repeat
      input top_speed
      total ← total + top_speed
      if top_speed > fastest then fastest ← top_speed
            if top_speed < slowest then slowest ← top_speed
            endif
      endif
      count ← count + 1
until count = 5000
average ← total * 100/5000
print fastest, slowest, average
```



Topic: 2.1.2 Pseudocode and flowcharts

2.5 Example 5

A shop sells books, maps and magazines. Each item is identified by a unique 4 – digit code. All books have a code starting with a 1, all maps have a code starting with a 2 and all magazines have a code beginning with a 3. The code 9999 is used to end the program.

Write an algorithm using pseudocode which input the codes for all items in stock and outputs the number of books, maps and magazine in stock. Include any validation checks necessary.

(NOTE: A 4-digit code implies all books have a code lying between 1000 and 1999, all maps have a code lying between 2000 and 2999 and all magazines a code lying between 3000 and 3999. Anything outside this range is an error)

```
books ← 0: maps ← 0: mags ← 0
repeat
  input code
  if code > 999 and code < 2000 then books ← books + 1
   else if code > 1999 and code < 3000 then maps ← maps + 1
   else if code > 2999 and code < 4000 then mags ← mags + 1
   else print "error in input"
 endif:endif:endif
until code = 9999
print books, maps, mags
```

(NOTE: A function called INT(X) is useful in questions like this. This returns the integer (whole number) part of X e.g. if X = 1.657 then INT(X) = 1; if X = 6.014 then INT(X) = 6 etc. Using this function allows us to use the **case** statement to answer this question:

```
books ← 0: maps ← 0: mags ← 0
repeat
  input code
  x ← INT(code/1000) * divides code by 1000 to give a
  case x of * number between 0 and 9
    1: books ← books + 1
    2: maps ← maps + 1
    3: mags ← mags + 1
    otherwise print "error"
  endcase
until code = 9999
print books, maps, mags
```

(This is probably a more elegant but more complex solution to the problem)