KS5 Pseudo code Questions

AS level

A level

3 Flow constructs

1 Programming languages consist of three basic programming constructs. For each construct, state its name and give a working example.

Construct 1: .....

Example: .....

.....

.....

.....

Construct 2: .....

Example: .....

.....

.....

.....

Construct 3: .....

Example: .....

.....

.....

.....

[6]

Question			Answer/Indicative content	Mark	Guidance
1			• Selection/Branching (1) (AO1.1) • Working selection example (1) (AO1.2) e.g. ```if a>b then c=b+42 endif``` • Iteration (1) (AO1.1) • Working iteration example (1) (AO1.2) e.g. ```for count=1 to 10 print(count) next count``` • Sequence (1) (AO1.1) • Working Sequence example (1) (AO1.2) e.g. ```qty = input() total = qty * price```	**6** **AO1.1** **(3)** **AO1.2** **(3)**	Max 6 marks Do not penalise pseudocode if it is does not conform to the specification pseudocode guidelines.

Files

6 (a) A programmer is going to design a procedure that will prompt for and receive two values, A and B. The procedure will then compare them. The procedure will also write a suitable message to a file on disk depending on whether:

- the values are the same
- A is less than B, or
- B is less than A.

Use pseudocode to write the procedure.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... **[5]**

6	a)		• Read in A and B. • Correct comparisons • Correct output messages. • Open file • Write to and close file. E.g. `A = input("Enter value A")` `B = input("Enter value B")` `myFile = openWrite("output.txt")` `if A < B then` `  myFile.writeLine("A is less than B")` `elseif B < A then` `  myFile.writeLine("B is less than A")` `else` `  myFile.writeLine("A is equal to B")` `endif` `myFile.close()`	**5** **AO3.1** **(5)**	Max 5 marks Accept open file in append mode

Nested iteration, Logical Operators

7 Given the following pseudocode:

```
d = 5

if ((a > b) OR (b >= c)) then
   if ((c < a ) XOR (c < b)) then // Check to see if one or the other
                                  // comparisons are TRUE, but not both
      d = 15
   else
      d = 16
   endif
else
   d = 14
endif

print(d)
```

(a) State the value of `d` if `a=42`, `b=41` and `c=42` .....

(b) State the value of `d` if `a=42`, `b=36` and `c=4` .....

(c) State the value of `d` if `a=42`, `b=36` and `c=36` .....

(d) Give **one** potential value of `b` if the output value of `a=42`, `c=44` and `d=14`. .....

[4]

7		a	• 16	4 AO2.2 (4)	Max 4 marks
		b	• 16		
		c	• 15		
		d	• 42 • 43 (max. 1)		

Selection

5 A car has a feature in its engine management system that is intended to save fuel and emissions produced when the car is waiting at traffic lights or in a traffic jam. The default option is that if the gears are disengaged and the car is not moving, the engine is switched off. There is a display on the dashboard that indicates when the engine has been switched off in this way.

However, sometimes it is necessary to keep the engine running even when the car is stationary, in order to provide electric power to charge the battery, run the heater, run the air conditioning system or keep the lights on. This, in turn, is affected by the external and internal temperatures, the settings chosen by the driver and the intensity of light outside.

(a) Identify **four** inputs needed by this feature of the engine management system.

For each one suggest a suitable data type for its storage.

Input	Data type

[8]

(b) Identify **two** outputs from this engine management feature.

.....

.....

.....**[2]**

(c) Write an algorithm that would produce the results described for this engine management system feature.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[8]

Question			Answer	Marks	Guidance
5	(a)		• Target temperature (1 – AO 2.1) integer/floating point (1 - AO 3.1). • Wheel movement (1 – AO 2.1) Boolean (1 – AO 3.1). • Engine running (1 – AO 2.1) Boolean (1 – AO 3.1). • Internal temperature (1 – AO 2.1) integer/floating point (1 – AO 3.1). • External temperature (1 – AO 2.1) integer/floating point (1 – AO 3.1). • External light level (1 – AO 2.1) integer/floating point (1 – AO 3.1). • Heating on (1 – AO 2.1) Boolean (1 – AO 3.1). • Air conditioning on (1 – AO 2.1) Boolean (1 – AO 3.1). • Gears engaged (1 – AO 2.1) Boolean (1 – AO 3.1).	**8** **AO2.1** **(4)** **AO3.1** **(4)**	Up to 4 marks (AO 2.1) one mark for each correct identification of input. Up to 4 marks (AO 3.1) one mark for identifying the correct data type. Any example of driver choices/settings related to something switched on (1 – AO 2.1) Boolean (1 – AO 3.1). Any example of driver choices/settings related to a level being set (1 – AO 2.1) integer/floating point (1 – AO 3.1).
	(b)		• Start engine (1), stop engine (1), signal to dashboard display (1).	**2** **AO3.1** **(2)**	1 mark for each correct identification up to a maximum of two identifications.
	(c)		• Use of endless loop while ignition is turned on (1). • If external temp<target then override (1). • If external light<target then override (1). • If internal temperature<set temperature and heater on then override (1). • If internal temperature>set temperature and air conditioning on then override (1). • If battery output<target then override (1). • If wheels not moving and gears not engaged and not override then stop engine (1). • If override and engine not running then start engine (1).	**8** **AO3.2** **(8)**	Up to 8 marks - 1 mark for each correct step in process. Various approaches could be correctly used. Allow credit for evidence of each step as given, however expressed.

Built in functions

(c) A programmer at Nobugs has written some program code that includes two user defined functions.

```
function my_function1(number)
    return number*number*number
endfunction

function my_function2(number)
    return number*number*number*number
endfunction

number=int(input("Enter a number "))
print(my_function1(number))
print(my_function2(number))
```

(i) Apart from the two functions written by the programmer, identify **two** other functions used in this piece of program code.

1.....

.....

.....

2.....

.....

.....**[2]**

(ii) The programmer tests this code with the input value of 3. State the output that would be obtained.

.....

.....

.....**[2]**

Question			Answer	Marks	Guidance
			use functions (1).		
	(c)	(i)	• int (1). • input (1). • print (1).	2 AO2.1 (2)	Up to 2 marks for valid identifications that demonstrates application of knowledge to given context.
		(ii)	• 27 (1). • 81 (1).	2 AO2.1 (2)	Up to 2 marks for stating valid output that demonstrates application of knowledge to given context.

User defined Function

When a video is selected, the program gives an estimate of the file size of the converted video.
The estimate in kilobytes is calculated by multiplying:

- the number of pixels in the video's resolution by…
- the number of frames per second by…
- the length of the video in minutes by…
- the value 0.0013.

(c) Write a function in pseudocode that estimates the size of a converted video.
It should:

- take in 3 parameters: `pixels, framesPerSec, lengthMins`
- calculate the estimated file size
- return a string with the file size, including units
- use megabytes for sizes under 1000 megabytes, otherwise the estimate should be given in gigabytes.

Examples:

- 480000 pixels at 24 frames per second for 60 minutes will return a size of 898.56 MB
- 480000 pixels at 24 frames per second for 120 minutes will return a size of 1.797 GB.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

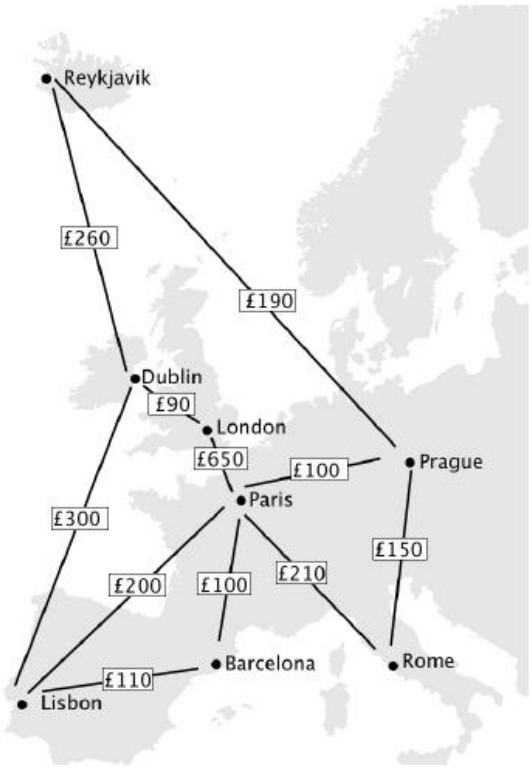
.....

..... **[4]**

	c		• function takes in all three parameters and returns a string. • Calculates file size correctly. • Files under 1000MB quoted in MB • Files 1000MB or over quoted in GB	4 **AO3.2**	Example function estimateFileSize(pixels, framesPerSec, lengthMins) output = "" fileSize = pixels * framesPerSec * lengthMins * 0.0013 if fileSize > 1000000 then fileSize = fileSize / 1000000 output = str(fileSize) + "GB" else fileSize = fileSize / 1000 output = str(fileSize) + "MB" endif return output endfunction allow 1024 or 1000 KB to MB and MB to GB.

Array, Iteration, Built in function

2 Atlas Airlines runs flights across cities in Europe. It stores the prices of different flights in its computer system.



(b) A function `tripCost` has been written that takes in two cities and returns the price of a direct flight between them.

e.g. `tripCost("Dublin", "London")` returns 90.

A journey is represented by an array called cities. An example of a trip from Dublin to Rome is shown below:

Dublin
London
Paris
Rome

(i) Write a program in the language or pseudocode of your choice that uses the cities array to calculate and output the cost of a given journey as a monetary value. In the case above this would be £950.

.....

.....

.....

.....

.....

.....

.....

.....

..... [5]

	(b)	(i)	• Creates a variable to represent total cost and initialises it to 0 (1). • Iterates up to the penultimate item of array (1). • Adds to the total cost… (1). • …Uses the correct arguments in the tripCost function (1). • Outputs the total cost formatted with a £ prefix (1).	5 AO3.2 (5)	For 5 marks – 1 mark for each correct step in process. Any program that has the functionality specified in the question should receive full marks. Example: `totalCost=0` `for i=0 to cities.Length-2` `totalCost=totalCost+` `tripCost(cities[i],cities[i+1])` `next i` `print("£"+totalCost)`

function

```
totalCost=0
for i=0 to cities.Length-2

        totalCost = totalCost + tripCost(cities[i], cities[i+1])
next i
print("£", totalCost)
```

2D array, Iteration

(c) Each airport has a three letter code. The airline's system stores the airports and corresponding airport codes:

Code	Name
BCN	Barcelona International
DUB	Dublin
LIS	Lisbon
LHR	London Heathrow
CDG	Paris, Charles De Gaulle
PRG	Prague
RKV	Reykjavik
FCO	Rome, Fiumicino

In a programming language or pseudocode of your choice write a program that takes in an airport code and finds and displays the airport name. You can assume a 2D array called airports has already been declared and populated with the data above. There is no need to validate the input and you can assume that the user will only enter a code that exists in the array.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[6]

Question	Answer	Marks	Guidance
(c)	• Takes in code of airport (1). • Iterates through the array (1). • Checks the value of the code column at each iteration (1). • To see if it is equal to code given (1). • When it is, it takes the airport name from the name column (1). • And prints it to the screen (1).	6 AO3.2 (6)	For 6 marks – 1 mark for each correct step in process. Any program that has the functionality specified in the question should receive full marks. Array could be 0 or 1 based. Examples include: `code=input("Please enter code")` `i=0` `while airports[1,i]!=code` `  i=i+1` `endwhile` `print("The airport is: "+airports[2,i])` OR `code = input("Please enter code")` `name=""` `for i=0 to 7` `  if airports[1,i]==code then` `  name=airports[2,i]` `  endif` `next i` `print("The airport is: "+name)`

2D Array, For, String manipulation, Built in functions, Casting

7 A DIY store has an offer: 'Spend £20 or more on decorating products and get 10% off all gardening products.'

When items are scanned in at the checkout they are stored in a 2-dimensional array called `purchases`, which stores the item name, category and price.

A receipt with the appropriate discounts deducted is then produced.

Examples of the array and corresponding receipt are shown in Fig. 2 and Fig. 3.

Matt Pink Paint	Decorating	6.99
Floral Wallpaper	Decorating	7.99
Magnolia Gloss Paint	Decorating	5.49
Weed Killer	Gardening	2.99
Picture Frame	Decorating	8.99
Plug Socket	Electrics	6.99
Doorbell	Electrics	15.99
Matt White Paint	Decorating	4.99
Tiles	Decorating	19.99
Grass Seed	Gardening	1.99
Lawn Mower	Gardening	129.99

Fig. 2

```
Matt Pink Paint £6.99
Floral Wallpaper £7.99
Magnolia Gloss Paint £5.49
Weed Killer £2.99
-£0.30 discount
Picture Frame £8.99
Plug Socket £6.99
Doorbell £15.99
Matt White Paint £4.99
Tiles £19.99
Grass Seed £1.99
-£0.20 discount
Lawn Mower £129.99
-£13.00 discount
----------------
TOTAL: £198.89
```

Fig. 3

Write an algorithm in pseudocode, using the array `purchases`, to:

- determine which items are given a discount
- calculate the total price to pay
- present this information on a receipt in the format shown in Fig. 3. **[6]**

Question			Answer	Marks	Guidance
7			• Prints receipt with item name and price on each line. (AO3.2) • Applies a 10% discount to gardening purchases. (AO3.2) • If decorating spend is £20 or more. (AO3.2) • Displays each discount on the receipt. (AO3.2) • Displays the correct total. (AO3.2) • Correct addressing of a 2D array (A02.1)	6 **A02.1 (1)** **AO3.2 (5)**	Example `decoratingSpend = 0.0` `for i = 0 to purchases[].size()` `  if purchases[i,1] == ”Decorating” then` `  decoratingSpend = decoratingSpend +` `  float(purchases[i,2])` `  endif` `next i` `total = 0.0` `disc = 0.0` `for i = 0 to purchases[].size()` `  print(purchases[i,0] + ” £” + purchases[i,2])` `  total = total + float(purchases[i,2])` `  if decoratingSpend >= 20 AND purchases[i,1] ==` `  ”Gardening” then` `  disc = round(float(purchases[i,2]) * 0.1, 2)` `  print(“-£” + str(disc) + ” discount”)` `  total = total - disc` `  endif` `next i` `print(“------------“)` `print(“TOTAL: £“ + str(total))`

Binary Search

(b) A programmer has been tasked with writing a function that uses a binary search to return a Boolean value. The function should return `true` if the target integer is found in a list of integers. Using pseudocode, write an algorithm for the function.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

[8]

Question			Answer/Indicative content	Mark	Guidance
	b		• Finding `midpoint` and correctly checking if `midPoint` value is target value … • … and if so returning `true` • Correctly checking that all elements have been checked … • … and if so returning `false` • Identify top or bottom of list … • … if top then `leftPtr` set/passed as `midPoint + 1` … • … if bottom then `rightPtr` set/passed as `midPoint - 1` • Correct use of indentation (AO2.1)	**8** **AO3.2 (7)** **AO2.1 (1)**	Max 8 marks Note: candidates may have given a recursive algorithm and this should is perfectly acceptable.
			Example iterative example `function findItem (numberArray integer[2000], targetNumber:integer, leftPtr:integer, rightPtr:integer): boolean` `  while (leftPtr <= rightPtr)` `      midPoint = (leftPtr + rightPtr) DIV 2` `      if (numberArray[midPoint] == targetNumber)` `        return true` `      else if (numberArray[midPoint] < targetNumber)` `        leftPtr = midPoint + 1` `      else` `        rightPtr = midPoint - 1` `      endif` `  endwhile` `  return false` `endfunction`		

ByVal & ByRef

(b) The code below uses a procedure:

```
name = "Sam"
addMessage(name)
print(name)

procedure addMessage(inText:byVal)
     inText = "Hello " + inText
endprocedure
```

Explain why this program outputs `Sam` rather than `Hello Sam`.

.....

.....

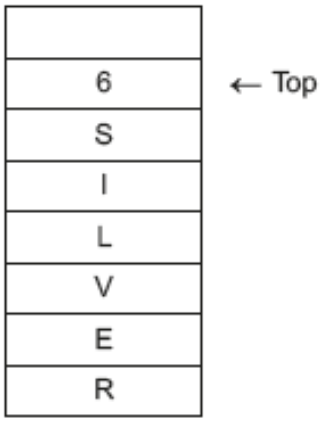
.....

..... **[2]**

Question			Answer/Indicative content	Mark	Guidance
6	b		Parameter / name is passed by value… …rather than by reference / by value does not change the original variable value		
					Maximum 2 marks

Stack

(d) A stack, in shared memory, is being used to pass a single variable length ASCII string between two sub-systems. The string is placed in the stack one character at a time in reverse order with the last byte holding the number of characters pushed i.e the text "SILVER" would be held in the stack as:



Use pseudocode to write a procedure that will take a text string passed to it and push it to the stack in the format defined above. You may assume any given input will fit in the stack.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

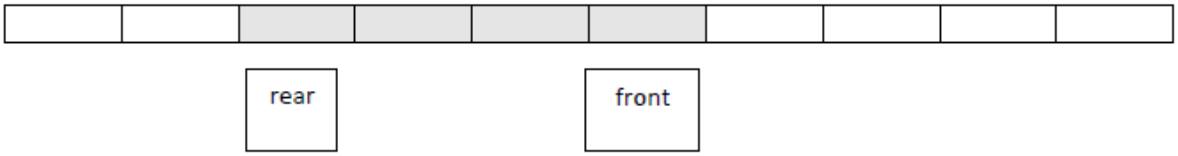
..... **[6]**

Question			Answer/Indicative content	Mark	Guidance
	d		• String length calculated (1) • Correct number of characters from passed string taken … (1) • … in reverse order (1) • Characters placed in stack in correct order (1) • String length placed in stack at correct point (1) • Meaningful variable names used (1) (AO2.1) Example program `procedure passToStack(passString)` `  stringLen = passString.Length()` `  if stringLen == 0 then` `  stack[0]=0` `  else` `  stackPtr = 0` `  stringPtr = stringLen - 1` `  for i = 1 TO stringLen` `  stack[stackPtr] = passString[stringPtr]` `  stackPtr = stackPtr + 1` `  stringPtr = stringPtr -1` `  next i` `  stack[stackPtr] = stringLen` `  endif` `endprocedure`	**6** **AO3.2 (5)** **AO2.1 (1)**	Allow `StackPtr` to be used instead of `i` in loop, as we would not expect them to know that some compilers do not always increment "loop counter" when they exit loops (i.e. loop counter on exit is undefined) Accept candidates using built-in stack methods e.g. `stack.push(word.substring(i,1))` Do not penalise for syntax errors if the logic can clearly be followed. Max 6 mark

Queue

2 The array queue shown below is set up to hold a small queue. Assume that there is sufficient storage to hold all necessary additions to the queue.

Queue



The table below shows variables that are used to maintain the queue:

Variable	Type	Purpose
front	integer	pointer to the front element of the queue
rear	integer	pointer to the rear element of the queue
queue_full	Boolean	indicates whether the queue is full
max	integer	the maximum size of the queue

Shown below is an algorithm that is intended to add an item to the queue.

```
procedure add_to_queue (item)

    if rear==max then
      queue_full=true

    else
       front=front + 1
       queue[front]=item

     endif

endprocedure
```

(a) Identify the parameter that is passed to this procedure.

.....**[1]**

(b) Describe the logical decision that is made.

.....

.....

.....**[2]**

(c) (i) This algorithm contains a logic mistake. Explain what the mistake is.

.....

.....

.....**[2]**

(ii) Rewrite the algorithm to correct the mistake.

.....

.....

.....

.....

.....**[2]**

2	(a)		• Item (1).	1 AO2.1 (1)	For 1 mark.
	(b)		• The queue is checked to determine if it is full (1), if it is, then stop (1) otherwise continue (1).	2 AO2.1 (2)	Up to 2 marks for valid description that demonstrates application of knowledge and understanding to given context.
	(c)	(i)	• The front of the queue is incremented (1), an item is placed at (near) the front of the queue (1), it should be placed at the rear (1).	2 AO2.2 (2)	Up to 2 marks for a valid explanation.
		(ii)	`front=front + 1` changed to `rear=rear-1` (1) `queue[front]=item` changed to `queue[rear]=item` (1)	2 AO3.2 (2)	For 2 marks.
			rear=rear-1 queue[rear]=item		

Selection, Comparative Operators

4 Below are extracts from the ASCII and EBCDIC character sets.

ASCII

Denary Value	65	66	67	68	69	70	71	72	73	74	75	76	77
Character	A	B	C	D	E	F	G	H	I	J	K	L	M
Denary Value	78	79	80	81	82	83	84	85	86	87	88	89	90
Character	N	O	P	Q	R	S	T	U	V	W	X	Y	Z

EBCDIC

Denary Value	193	194	195	196	197	198	199	200	201	...	209	210	211	212
Character	A	B	C	D	E	F	G	H	I	...	J	K	L	M
Denary Value	213	214	215	216	217	...	226	227	228	229	230	231	232	233
Character	N	O	P	Q	R	...	S	T	U	V	W	X	Y	Z

(b) Write a function that given the denary value of an EBCDIC uppercase letter, returns the denary value of an ASCII uppercase letter. If a value is entered that doesn't correspond to an uppercase EBCDIC letter the function should return -1

e.g.

`convert(201)` returns 73

`convert(209)` returns 74

`convert(78)` returns -1

```
function convert(ebValue)
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
.....
endfunction
```

[5]

	b		- Value between 193 and 201 returns respective ASCII value between 65 and 73 - Value between 209 and 217 returns respective ASCII value between 74 and 82 - Value between 226 and 233 returns respective ASCII value between 83 and 88 - Values less than 193 and greater than 233 return -1 - Values between 202 and 208, and 218 and 225 return -1. (1 per -)	**AO3.2** **5**	`function convert(ebValue)` `  if ebValue >= 193 and ebValue <= 201 then` `  return ebValue - 128` `  elseif ebValue >= 209 and ebValue <= 217 then` `  return ebValue - 135` `  elseif ebValue >= 226 and ebValue <= 233 then` `  return ebValue - 143` `  else` `  return -1` `  endif` `endfunction` A program that returns a value 128 less for values between 193 and 208 would receive the first mark but not the last one. (The same principle applies for points 2 and 3)

Function, Iteration, Selection, String Manipulation

(b) The algorithm will make use of a function, `contains`, that compares two strings and checks if the second string contains the first string. For example, calling the function with (`"fox"`, `"foxhound"`) this would return `true`.

The function needs to:

- Take two strings as parameters, `string1` and `string2`
- return `true` if `string1` is contained within `string2`, or both strings are identical
- return `false` if `string1` is not contained within `string2`

Write, using pseudocode, the function `contains`.

Annotate your pseudocode with comments to show how it solves the problem.

[7]

4	b		1 mark per bullet to max 7 • Correct function declaration, taking string1 and string2 as parameters • Use of a flag	7 AO2.1 (1) AO2.2	Allow reversed true and false
			• Looping through string2 by some means • Using string manipulators to check either letters or substrings of string2 • Correctly setting return value to `true` • Returning `true` or `false` accordingly • Comments that explain how the algorithm works e.g. `function contains(string1, string2)` `  wordInside = false` `  for i = 0 to (string2.length - string1.length)` `  if string2.substring(i, string1.length) == string1 then` `  wordInside=true` `  endif` `  next i` `  return wordInside` `endfunction`	(2) AO3.2 (4)	

Casting, MOD, DIV, Concatenation, Selection, Iteration

5 A procedure is shown below.

```
01   procedure fun1(x)
02       y=""
03       if x < 0 then
04           flag = true
05           x = x * -1
06       endif
07       while (x > 0)
08           y = str(x MOD 2) + y
09           x = x DIV 2
10       endwhile
11       if flag == true then
12           y = "1" + y
13       else
14           y = "0" + y
15       endif
16       print(y)
17   endprocedure
```

`flag` is a local variable and has a default value of `false`.

(a) Explain why `str` is needed in line `08`.

.....

.....

.....

.....

..... **[3]**

(b) **(i)** Show the result of `y` when the procedure is called with: `fun1(10)`. Show your working.

y: ..... **[4]**

(ii) Show the result of `y` when the procedure is called with `fun1(-13)`. Show your working.

y: ..... **[4]**

5	a		max 1 mark for purpose and max 2 for application to line 8. • Case/convert numeric value to a string • Result of `x MOD 2` will be a number... • ...needs to be concatenated to string `y`	3 AO1.2 (1) AO2.2 (2)	**Accept "avoid type mismatch error" for 1 mark as part of application to line 8**

5	b	i	1 mark for `flag = false`. 1 mark for showing y's values correctly through loop 1 mark for showing x's values correctly through loop 1 mark for the correct answer • `10 > 0` (1) • `y = "0"` `x = 5` `y = "10"` `x = 2` `y = "010"` `x = 1` `y = "1010"` (1 for y at each stage) `x=0` (1 for x at each stage) • `01010` (or `y = "01010"`)	4 AO1.2 (1) AO2.1 (3)	If only the result is shown, 1 mark only. Award bod if no "" with y Accept any suitable answer, e.g. trace table
5	b	ii	1 mark for `flag = true` and `x=13` 1 mark for showing y's values correctly through loop 1 mark for showing x's values correctly through loop 1 mark for the correct answer • `flag = true` `x = 13` • `y = "1"` `x = 6` `y="01"` `x=3` `y="101"` `x=1` `y="1101"` (1 for y at each stage) `x=0` (1 for x at each stage) • `result=11101` (or `y="11101"`)	4 AO1.2 (1) AO2.1 (3)	If only the result is shown, 1 mark only. Award bod if no "" with y Accept any suitable answer, e.g. trace table

Iteration, Built in functions, XOR

8 It is possible to use XOR together with a key to encrypt a plain text message.

For example, to encrypt the bit pattern 111100001111 using the number 5 as a key, the key is repeated as often as necessary to match the length of the message. An XOR operation is performed to generate the encoded message.

111100001111 <- Message to be encrypted

101101101101 <- Key: repeated as necessary

010001100010 <- Encrypted message

Write an algorithm to accept a text message and a key that would use the method above to generate an encoded message. Annotate your pseudocode with comments to show how it solves the problem.

[8]

8			Individual steps in pseudocode: • Input text and key • Get a length of text • Method to convert text to binary such as • ASCII/lookup table • Convert key to binary or accept key as binary • Use of loop to iterate through message duplicate key element • Perform XOR operation • Convert back to text Programming marks to be awarded as follows • pseudocode shows the input of text and the length being established (1-AO3.2) • pseudocode shows a method to convert text and key to binary and binary to text with the need for the key to be duplicated (1-AO3.2) • pseudocode shows a loop iterating through the message (1-AO3.2) • pseudocode shows the XOR operation being correctly applied (1-AO3.2) Possible annotated comments: • The pseudocode will prompt for an input for a message and key and store it as a variable. (1 – AO 2.2)	8 AO2.2 (4) AO3.2 (4)	Up to 4 marks for valid pseudocode (AO3.2). Up to 4 marks for annotated comments used (AO2.2). Example pseudocode: `msg=(message)` `key=int(key)` `msglength=len(msg)` `keystring=str(bin(key))` `keystring length=len(keystring)` `for i=1 to msglength` `asc msg=asc msg+asc(msg(i,i))` `end for` `asc msg length=len(asc msg)` `for i=1 to asc msg length/8` `complete key=complete key+keystring` `end for` `for i=1 to len(asc msg)` `encryptmsg=encryptmsg+complete key(1,i) XOR asc msg(1,i)` `end for`

Question			Answer	Marks	Guidance
			• The length of the message variable is checked and then converted to binary (1 – AO 2.2). • The key is then converted to binary if needed (1 – AO 2.2). • The loop then duplicates the key and performs a XOR operation, the output of this is then converted back to text which is now encrypted (1 – AO 2.2).		

xor encryption.py - C:\Users\ad\Desktop\xor encryption\xor encryption.py (3.4.4)

File Edit Format Run Options Window Help

```
#message=============================================================================
msg = ("cat")

msg_ascii = []
for letter in msg:
    msg_ascii.append(ord(letter))    #ord() = convert char to ascii value

msg_bin = []
for letter in msg_ascii:
    letter = bin(letter)[2:]    #convert to binay
    letter = letter.zfill(8)    #pad out 0s
    msg_bin.append(letter)      #add to binary list

str_msg = ""
for i in msg_bin:
    str_msg += i    #convert list to single string


#key==================================================================
key = 5
key_bin = bin(key)[2:]      #convert to binay
key_bin = key_bin.zfill(8)  #pad out 0s

key_bin_rep = []
for letter in msg_ascii:
    key_bin_rep.append(key_bin) #repeat key for len of message

str_key = ""
for i in key_bin_rep:
    str_key += i  #convert list to single string


#XOR==================================================================
encrypt = ""
length = len(str_msg)

for i in range(length):
    if str_msg[i] == str_key[i]:  #see if both 1 or 0
        encrypt += "0"
    else:
        encrypt += "1"


print(str_msg)
print(str_key)
print(encrypt)
```

Search, Parameters, Recursion v Iteration

2 Consider the following algorithm in Fig.2, expressed in pseudocode, as a function S:

```
function S(A[0..N-1], value, low, high)

     if (high < low)then
           return error_message
      endif

      mid = (low + high) / 2

     if (A[mid] > value)then
            return S(A, value, low, mid-1)
     elseif (A[mid] < value)then
           return S(A, value, mid+1, high)
     else
           return mid
     endif

endfunction
```

Fig.2

(a) State the name of the algorithm implemented in Fig.2.

.....[1]

(b) Describe the purpose of this algorithm.

.....

.....

.....

.....[2]

(c) Parameters are passed to this function. Complete the following table to identify these parameters and the purpose of each.

Parameter name	Purpose

[8]

(d) (i) Describe what is meant by recursion.

.....

.....

.....

.....[2]

(ii) Identify one example of where recursion occurs in this algorithm.

.....

.....[1]

(f) Rewrite the algorithm in Fig.2 without using recursion. Annotate your pseudocode with comments to show how it solves the problem.

[8]

Question			Answer	Marks	Guidance
2	(a)		• Binary search (1).	1 AO1.1 (1)	For 1 mark.
	(b)		• To locate an item (1) in a list (1). The list is in some order (1).	2 AO1.2 (2)	Up to 2 marks for a valid description.
	(c)		• A (1 AO – 1.2) the list to be searched (1 – AO 2.1). • Value (1– AO 1.2) the item being searched for (1 – AO 2.1). • Low (1 AO – 1.2) the lower end of the list/sublist (1 – AO 2.1). • High (1 AO – 1.2) the upper end of the list/sublist (1 – AO 2.1).	8 AO1.2 (4) AO2.1 (4)	Up to 4 marks for each correct identification (AO1.2). Up to 4 marks for each purpose identified (AO2.1).
	(d)	(i)	• The function calls itself (1) from within the function.	2 AO1.1 (2)	Up to 2 marks for a valid description.
		(ii)	• Return S(A, value, low, mid-1) (1) return S(A, value, mid+1, high) (1).	1 AO1.2 (1)	For 1 mark (either point). Accept if point in the algorithm is unambiguously referenced. There are two recursive calls in the program. Either is acceptable.
	(f)		Individual steps in pseudocode: • Function declaration • Parameters all given correctly • Found flag used • Mid-point found • Check if value found is greater than looked for	8 AO2.2 (4) AO3.2 (4)	Up to four marks for valid pseudocode (AO3.2). Up to four marks for annotated comments used (A02.2).

2D array, Subroutines, Recursion, Recursion v Iteration

2 The layout for a 2-player board game is shown in Fig 2.1

START	1	2	3	4	5	6	7
15	14	13	12	11	10	9	8
16	17	18	19	20	21	22	23
31	30	29	28	27	26	25	24
32	33	34	35	36	37	38	39
47	46	45	44	43	42	41	40
48	49	50	51	52	53	54	55
END	62	61	60	59	58	57	56

Fig 2.1

The game is played by rolling two 6-sided dice and moving that number of spaces. Both players start on the START space. If a player lands on a space occupied by the other player, they move to the next available space.

The board is to be stored as a 2-dimensional array.

(b) Each time a player moves, a series of obstacles are to be added to the board.

On their turn, each player rolls two dice. The smaller number from the two dice is taken, and that many obstacles will appear on the board in random locations.

For example, if a 3 and 6 are rolled, then 3 obstacles will appear.

A recursive function is written in pseudocode to perform this task.

```
01 function generateObstacle(diceNumber)
02     if diceNumber == 0 then
03         return true
04     else
05         x = randomNumber(0, 7)
06         y = randomNumber(0, 7)
07         board(x, y) = new obstacle()
08         generateObstacle(diceNumber-1)
09     endif
10 endfunction
```

The code `new obstacle()` generates an instance of the object obstacle.

(i) Explain the purpose of the code in line `01` in the algorithm.

.....

.....

.....

..... **[2]**

(ii) Identify the line of code where recursion occurs.

..... **[1]**

(iv) Rewrite the function in part (b) so it uses iteration instead of recursion.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [4]

(v) If a position on the board is not occupied, its value is set to a blank string ("").

The current algorithm does not check if the random space generated is currently occupied.

Write a subroutine that takes the generated position of the board, checks if it is free and returns `true` if free, or `false` if occupied.

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... [3]

2	b	i	1 mark per bullet to max 2 • Declares a function called `generateobstacle`(1) • Has parameter `diceNumber` (1)	2 AO1.1 (1) AO2.1 (1)	
2	b	ii	• 08(1)	1 AO2.1 (1)	

2	b	iv	1 mark per bullet • Loop start and end in correct positions(1) • With correct number of iterations(1) • Returns a value(1) • All other code correct, in the right place(1) e.g. `function generateobstacle(diceNumber)` `  for count = 1 to diceNumber` `  x = randomNumber(0, 7)` `  y = randomNumber(0, 7)` `  board(x, y) = new obstacle()` `  next count` `  return true` `endfunction`	4 AO2.2 (1) AO3.2 (3)	
2	b	v	1 mark per bullet, to max 3 • Appropriate declaration of function, taking 2 parameters(1) • Checks position in board against "" correctly(1) • Returns `false` and true correctly(1) e.g. `function checkFree(x, y)` `  if board(x, y) == "" then` `  return true` `  else` `  return false` `  endif` `endfunction`	3 AO2.1 (1) AO3.2 (2)	

User defined function, Hash Table

4 A bus runs between two cities. There are a number of stops on the bus route labelled `StopA`, `StopB` and so on. The timetable for the route is represented as a hash table. For each entry in the hash table the key is the bus stop code and the data attached to it is a (zero indexed) array of the times a bus arrives at the stop. The times are stored as strings.

An extract of the hash table is shown below:

```
times=
{
"StopA":["06:55", "07:25", "07:55", "08:55", "09:55", "11:55", "14:00",
"15:00", "15:30", "16:00"]
"StopB":["06:40", "07:40", "08:40", "09:20", "09:40", "14:00", "15:00",
"16:00", "16:30"]
…
…
}
```

`print(times["StopA"][1])` displays `07:25`

(b) Write a function called `timeValue` that given a time stored in a string, returns the equivalent integer (using thousands and hundreds for the hours and tens and units for the minutes). The given string should be assumed to represent the time in the 24-hour clock in the format HH:MM

`timeValue("07:55")` should return `755`
`timeValue("15:30")` should return `1530`

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... **[3]**

(c) Write code for a function that takes in the name of a stop (`stopName`) and the current time as an integer (`currentTime`) in the format described in part (b) (using thousands and hundreds for the hours and tens and units for the minutes). It should return the time of the next available bus in the string format. If there are no more buses available that day it should return the string "No buses".

Example `nextBus("StopA", 1013)` should return `"11:55"`

```
function nextBus(stopName, currentTime)
```

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

```
endfunction
```

[5]

				(AO1.2)	
	b		Correctly named function that takes in time as a parameter and returns a value. (1) Minutes element is correct (1) Hours element is correct (1)	3 (AO 3.2)	Returned value needn't be correct for first mark Example solution: `function timeValue(givenTime)` `  intTime=int(givenTime.substring(3,2))` `  intTime=intTime+int(givenTime.substring(0,2))*100` `  return intTime` `endfunction`

	c		Correct stop array extracted/referenced in code (1) Sensible attempt to iterate through the array (1) Program returns time of next bus (1) Program returns No Buses when no more buses left. (1) Program runs without an index out of bounds error (You may assume short circuit evaluation i.e. if the array is in the second part of an and condition it won't be checked if the first half evaluates to false.) (1)	5 (A03.2)	Marks 1-2 can be awarded even if the program doesn't exhibit behaviour needed for marks 3-5 Marks 1-4 can be awarded even if mark 5 can't be awarded as program would in reality crash. `count = 0` `timesLeft = true` `timesList = times[stopName]` `while timesLeft == true and` `timeValue(timesList[count]) < currentTime` `  count = count + 1` `  if count == timesList.length then` `  timesLeft = false` `  endif` `endwhile` `if timesLeft == true then` `  return timesList[count]` `else` `  return "No Buses"` `endif`

```
count = 0
timesLeft = true
timesList = times[stopName]

while timesLeft == true and timeValue(timesList[count]) < currentTime
        count = count + 1
        if count == timesList.length then
                timesLeft = false
        endif
endwhile

if timesLeft == true then
        return timesList[count]
else
        return "No Buses"
endif
```

Iteration, String manipulation, Array/Stack

6 A company is writing a syntax checker to be used when writing HTML.

(a) The first thing the program does is add every tag in a piece of text to the data structure `dataStructureA`.

The string `X` is added to `dataStructureA` with the code

```
dataStructureA.add("X")
```

The string type variable `htmlCode` holds the code that is to have its tags added.

If `htmlCode` were to contain:
`<html><head><title>My Page</title></head><body>Hello</body></html>`

`dataStructureA` would have the following contents:

<html>	<head>	<title>	</title>	</head>	<body>	</body>	</html>

Write the code to fill `dataStructureA` with the tags in `htmlCode`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....[7]

6	a		- Adds the tag name… - Includes the opening < - Includes the closing > and nothing further - Tags are added to data structure. - Adds all tags in the string. - Sensible variable names used - Correct use of indentation (1 per -)	**AO3.2** **7**	`tagStartPos = 0` `insideTag = false` `i = 0` `while i < htmlCode.length` `  if htmlCode.substring(i,1) == "<" and insideTag == false then` `  tagStartPos = i` `  insideTag = true` `  elseif insideTag == true and htmlCode.substring(i,1) == ">" then` `  dataStructureA.add(htmlCode.substring(tagStartPos, i-tagStartPos+1))` `  insideTag = false` `  endif` `  i = i + 1` `endwhile`

Data structures

(b) Part of the program checks that the HTML tags are well formed.
Well formed HTML has tags that are nested but never overlapping.

e.g.

`<p>The cat <strong>sat</strong> on the mat.</p>` is well formed.

Whereas `<p>The cat <strong>sat</p> on the mat.</strong>` is not well formed as `p` closes before the `strong` inside it has been closed.

All comments and single tags (e.g. `img`, `br` etc) are removed from `dataStructureA`.
All attributes are removed from the within the tags.

(i) The contents of `dataStructureA` may look similar to below:

`<html>`	`<head>`	`<title>`	`</title>`	`</head>`	`<body>`	`<h1>`	`</h1>`	`</body>`	`</html>`

Tags are removed from `dataStructureA` in the same order they were added.

Identify what type of data structure `dataStructureA` is.

.....

.....[1]

`dataStructureB` is given a closing tag and gives the corresponding opening tag.

e.g.

```
openingTag=dataStructureB.get("</head>")

openingTag is "<head>"
```

(ii) Identify what type of data structure `dataStructureB` is.

.....

.....[1]

The following code is used to check if the tags are well formed.

```
function checkTags(dataStructureA)
{
   valid=true
   //loops while code is still valid
   //and dataStructureA has tags
   while valid==true and dataStructureA.isEmpty()==false
      tag=DataStructureA.remove()
      //Next, check if closing tag
      if tag.substring(1,1)=="/" then
           if dataStructureC.remove()!=dataStructureB.get(tag)then
            valid=false
           endif
      else
           dataStructureC.add(tag)
      endif
   endwhile
   return valid
}
```

(iii) Identify what type of data structure `dataStructureC` is.

.....

.....[1]

(iv) Explain why `dataStructureC` is suited to checking if HTML is well formed.

.....

.....

.....

.....[2]

	b	i	Queue	**AO2.1** **1**	
		ii	Hashtable	**AO2.2** **1**	Accept Hashmap/Associative Array/Dictionary
		iii	Stack	**AO2.2** **1**	
		iv	Stack uses a last in first out approach… … and the last HTML tag to be opened should be the first to be closed.	**AO2.2** **2**	

Object Orientated Programming (OOP)

4 Livid Lizards is a computer game in which players get to fire lizards from a cannon to knock down walls. Players get to pick different types of lizards, each with qualities and special powers.

The game is coded using an object-oriented language. Below is the code for the lizard class:

```
class Lizard

      private speed
      private mass
      private size

      public procedure new(givenSpeed, givenMass, givenSize)
            speed=givenSpeed
            mass=givenMass
            size=givenSize
      endprocedure

      public function breakBlock(brick)
            if speed*mass>=brick.getStrength() then
                  speed=((speed*mass)-brick.getStrength())/mass;
                  return true
            else
                  return false
            endif
      endfunction

      ...
      ...
      ...

endclass
```

(a) Lizard is a class. Describe what is meant by a class.

.....

.....

.....[2]

(b) Identify an attribute in the Lizard class.

.....[1]

4	(a)		• A template (1) defining methods and attributes (1) used to make objects (1).	2 AO1.1 (2)	Up to 2 marks for a valid description.
	(b)		• Speed (1)/mass (1)/size (1).	1 AO1.1 (1)	For 1 mark.

OOP (encapsulation, inheritance, attributes and methods)

2 Mobile Treasure Hunt is a game played on a mobile phone. The game shows the user's position on a map of their local area. Treasure randomly appears on the map and users must move to the appropriate area to collect the treasure before it disappears.

Below is part of the code from Mobile Treasure Hunt.

```
class Treasure

    private value
    private weight
    private name

    public procedure new(givenName)
        name=givenName
        weight=20
        value=randomInteger(1,20)
    endprocedure

    public procedure changeName(givenName)
        name=givenName
    endprocedure

endclass

class TreasureChest inherits Treasure

    private locked

    public procedure new(givenName)
        super.new(givenName)
        locked=false
        value=randomInteger(1,100)
        weight=randomInteger(80,120)
    endprocedure

    public procedure pickLock()
        if getRandomNumber()>0.5 then
            locked=false
        endif
    endprocedure

endclass
```

(b) Explain what is meant by the term 'encapsulation' with reference to the attribute called `name`.

.....

.....

.....

.....

.....

..... **[3]**

(c) Describe what is meant by the term 'inheritance', referring to the code in Fig. 2.1.

.....

.....

.....

.....

.....

..... **[3]**

(d) Identify all attributes and methods in the `TreasureChest` class.

Methods: .....

.....

Attributes: .....

..... **[2]**

	b		When an attribute is made private (so it can't be directly accessed or changed from outside the class) (1) Public methods are used to read / amend the attribute's value (1) The attribute name's value can only be amended through the method `changeName`. (1)	3 **(AO 1.2)**	
	c		When a class has the attributes and methods of its parent class. (1) It may also have methods and attributes of its own (1) `TreasureChest` inherits from the class `Treasure` (1)	3 **(AO 1.2)**	
	d		Methods: (constructor/`new`), `changeName`, `pickLock` (1) Attributes: `value`, `weight`, `name`, `locked` (1)	**2** **(AO 1.2)**	Do not penalise for not including constructor. Only give method mark if both other methods are listed Only give attributes mark if all four attributes are listed.

OOP (graph, queue, breadth first)

6 A salesman travels around the country, stopping at specific places, and then returning to the starting place.

Fig 6.1 shows an example map of places that the salesman visits.

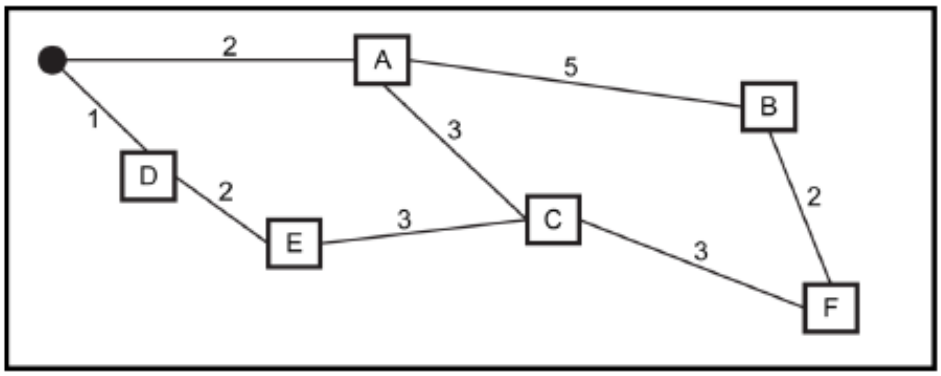


Fig 6.1

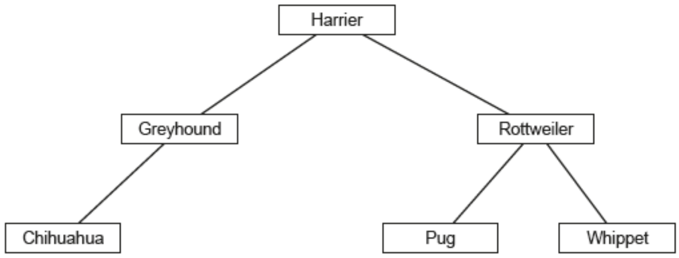
The filled in circle represents the start and end point. The letters represent the places to visit. The lines are the routes available and the numbers are the length of time each route takes to travel.

(ii) The pseudocode below shows part of an algorithm which uses a queue to traverse the graph breadth-first. Complete the missing elements of the algorithm.

```
markAllVertices (notVisited)

createQueue()

start = ..............

markAsVisited(..............)

pushIntoQueue(start)

while QueueIsEmpty() == ..............

    currentNode = removeFromQueue()

    while allNodesVisited() == false

        markAsVisited(..............)

        //following sub-routine pushes all nodes connected to

        //currentNode AND that are unvisited

        pushUnvisitedAdjacents()

    endwhile

endwhile
```

6	c	ii	1 mark each markAllVertices (notVisited) createQueue() start = **currentNode** (1) markAsVisited(**start**) (1) pushIntoQueue(start) while QueueIsEmpty() == **false** **(1)** popFromQueue(currentNode) while allNodesVisited() == false markAsVisited(**currentNode**) (1) //following sub-routine pushes all nodes connected to //currentNode AND that are unvisited pushUnvisitedAdjacents() endwhile endwhile	4 AO1.2 (2) AO2.1 (1) AO3.2 (1)	

7 A binary search tree is used to store the names of dog breeds.



OOP - Binary Search Tree

(d) The tree is coded using object oriented programming.

Each dog breed is represented by an object of class Node.

The Node class has the methods:
`getLeftNode()` – returns the left hand child node or null if there is no left hand child.
`getRightNode()` – returns the right hand child node or null if there is no right hand child.
`getBreed()` – returns the name of the breed stored in that node.

The program allows for a breed name to be entered, and depending on whether the breed is in the tree or not, displays either:

<*breed name*> `is not in the tree.`

or

<*breed name*> `is in the tree.`

Complete the program below. Credit will be given for readability of code.

```
name=input("Enter the name of a breed")
breedNode=tree.root() //breedNode is an object of type Node
                      //representing the root of the tree
```

[6]

	d		Calls `getLeftNode()` when name is less than the value of the current node (1) and calls `getRightNode()` when name is less than the value of the current node. (1) Declares a breed to be in the tree if and only if it exists.(1) Declares a breed not to be in the tree if and only if it doesn't exist (1) Presents output strings in correct format (1) Sensible use of variable names and correctly indented (1)	6 (5 AO 3.2, 1 AO 1.2)	Points 4 and 5 can be awarded even if 1-3 aren't. `notThere = false` `while breedNode.getName() != name and notThere == false` `  if name < breedNode.getName() then` `  if breedNode.getLeftNode() != null then` `  breedNode = getLeftNode()` `  else` `  notThere = true` `  endif` `  else // must be greater` `  if breedNode.getRightNode() != null then` `  breedNode = getRightNode()` `  else` `  notThere = true` `  endif` `  endif` `endwhile` `if notThere == true then` `  print(name+ " is not in the tree.")` `else` `  print(name+" is in the tree")` `endif`

Section B: 2D Array, Iteration, Selection, Functions

Section B

Answer all questions

7 Four in a Row is a game where two players drop coloured discs into a grid, with the aim to get four of their own colour in a row.
Each player is given a set of coloured discs, red (R) or yellow (Y). The players take it in turns to drop their disc into a column in the grid. The disc drops down to the lowest available space in that column.

The grids below (Fig 7.1 and 7.2) show what happens when the yellow player drops a disc into column 2:

Before

	0	1	2	3	4	5	6
0							
1							
2							
3		R	Y	Y			
4		Y	R	R	Y		
5	R	Y	R	R	Y	R	

Fig 7.1

After

	0	1	2	3	4	5	6
0							
1							
2			Y				
3		R	Y	Y			
4		Y	R	R	Y		
5	R	Y	R	R	Y	R	

Fig 7.2

The game continues until one player has got four discs of their colour in a straight row in any direction i.e. vertical, horizontal, or a diagonal.

(b) A 2-dimensional array, `grid`, is used to hold the game grid.

Using pseudocode, write a function that takes as input the player whose turn it is, and the column number they select as their turn. The function either:

- returns 999 (i.e. the column is already full), or
- stores the player's move in the array and returns the row the disc has been placed in.

Annotate your pseudocode with comments to show how it solves the problem.

[6]

(c) After a player makes their move, the program needs to check if that player has won (i.e. the player has four discs in a row).

Subroutines have already been written to check if the player has won vertically, or diagonally.

Using pseudocode, write a procedure that reads appropriate parameters and checks if the player has won horizontally. If the player has won, display an appropriate message identifying which player has won.

[6]

(d) (i)* The programmer is writing a new version of the game, where each player removes one disc from the bottom row of the grid before a new move is made.

In the example below, player R removes one disc from column 2 (Before) and places one in column 4 (After).

Before

	0	1	2	3	4	5	6
0							
1							
2							
3		R	Y	Y			
4		Y	R	R	Y		
5	R	Y	R	R	Y	R	

Fig 7.4

After

	0	1	2	3	4	5	6
0							
1							
2							
3		R		Y	R		
4		Y	Y	R	Y		
5	R	Y	R	R	Y	R	

Fig 7.5

(d) (iii) A procedure needs to be written to remove the disc from the chosen column. The procedure will:

- have the column the disc is being removed from as a parameter
- move each disc in that column down to the bottom of the grid
- replace the top space with an empty string ("")

Complete the algorithm below.

```
procedure playDisc (removeColumn)
```

[3]

7	b		Programming steps to award marks for, max 6 • Function declaration taking 2 parameters(1) • Looping through all 6 elements in the array…(1) • …in the correct order (bottom to top, 5 to 0) (1) • Place player in correct position…(1) • … return the position(1) • Return 999 if no space in that column(1) Example pseudocode e.g. `function gameMove(player, column)` `  for x = 5 to 0 step -1` `  if grid(x,column) == "" then` `  grid(x,column) = player` `  return x` `  endif` `  next x` `  return 999` `endfunction`	6 AO2.2 (2) AO3.2 (4)	

7	c		The algorithm can be tackled by either a) Checking all possible positions, b) Just checking what is around the last move c) checking the entire row. Full marks can be awarded for all possible methods, if correct. Programming steps to be awarded as follows, to max 6 marks. • Appropriate procedure declaration… • … taking at least player parameter(1) • Checking each element in the row(1) • Only checking valid options (e.g. if checking row-3, row-2, row-1, 0 then they need to check all rows are within the grid) (1) • Updating a counter/checking for four-in-a-row(1) • Appropriate output message(1) Example pseudocode: `procedure checkHorizontal (player, row)` `  counter = 0` `  for x = 0 to 6` `  if grid(row, x)== player then` `  counter = counter + 1` `  if counter >= 4 then` `  print "Player " + player + " has won"` `  endif` `  else` `  counter = 0` `  endif` `  next x` `endprocedure`	6 AO2.2 (2) AO3.2 (4)	

7	d	iii	1 mark per bullet to max 3 • Appropriate loop to move through the column (bottom to top, so 5 to 1) • Replacing the grid with the value above (5 with 4 etc.) • Replacing row 0 with "" e.g. `procedure playDisc (removeColumn)` `  for x = 5 to 1 step -1` `  grid(x, removeColumn) = grid(x-1, removeColumn)` `  next x` `  grid(0, removeColumn) = ""` `endprocedure`	3 AO2.2 (1) AO3.2 (2)	

Section B: Hanoi Tower (OOP)

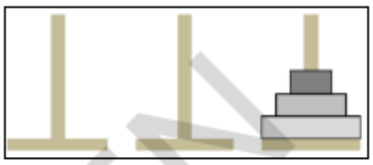
Section B

Answer all questions.

10 The Towers of Hanoi is a classic puzzle. Disks are placed in order on a pole, the biggest disc at the bottom of the pole, the smallest disk at the top of the pole, on the first of three poles. The challenge is to get them to the third pole in the same order.

First step

Final step



The disks can only be moved under the following rules:

- only one disk can be moved at a time
- a disk can only ever be placed on an empty pole or on top of a larger disk
- a larger disk can never be placed on a smaller disk.

This is a valid move.

This would be a valid second move.

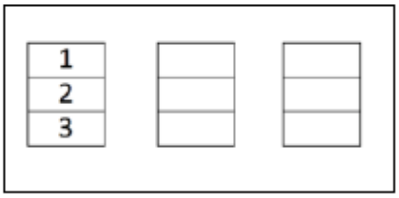
This would **not** be a valid third move (you can't put a bigger disk on a smaller one.)

This **would** be a valid third move.

(a) Each disk can be represented by an integer denoting its size.

So

Can be represented by



Stack (push)

(ii) The tower class has the method push. It takes in the value of the disk to be pushed. It adds it to the top of the stack if it is a valid move. If it is not a valid move, the value of the disc is not added and the message 'Invalid move' is printed to the screen.

The stack is implemented using an array called pole and an integer called pointer. Pointer represents the index of the array position at the top of the stack.

```
class Tower
      private array pole[10]
      private pointer

      public procedure new()

            pointer=0

      endprocedure

      public procedure push(diskValue)
            //Code for push method

      endprocedure
endclass
```

Write the pseudocode to go in the push method. Annotate your pseudocode with comments to show how it solves the problem. You are not expected to test for overflow.

[6]

(c) Rather than using a tree, the following iterative algorithm can be used to play the perfect game where the number of disks is odd. A similar algorithm exists for an even number of disks.
A program is required to solve the Towers of Hanoi puzzle using the iterative algorithm below.

Iterative algorithm to solve Towers of Hanoi for an odd number of disks.

Cycle through the following three steps until the puzzle is solved (which may be after any of the steps):

- make the valid move between tower1 and tower3
- make the valid move between tower1 and tower2
- make the valid move between tower3 and tower2

NB: The valid move might be in either direction but there will only be one possibility each time.

In this program there are three objects of the class Tower; `tower1`, `tower2` and `tower3`

The Tower class has the methods `push`, `peek` and `pop`

The method `push` adds a disk to the tower, for example `tower1.push(3)` adds a disk of size 3 to tower1.

The method `peek` returns the value of the disk on top of the tower but does not remove it. It returns the value 999 if the tower is empty.

The method `pop` removes a disk from a tower and returns the value of that disk.

e.g. `x=tower1.peek()` would make x equal to 2 and the towers stay the same.
`x=tower1.pop()` would make x equal to 2 and remove 2 from tower1.

Stack (push, pop, peek)

Complete the pseudocode program below so when given an odd number of disks, below 100, on tower1 they will be moved to tower3 using the iterative algorithm. Annotate your pseudocode with comments to show how it solves the problem.

[10]

```
noOfDisks=5 //Can be set to any odd number below 100
      tower1 = new Tower()
      tower2 = new Tower()
      tower3 = new Tower()

      i=noOfDisks
      while i>0
            tower1.push(i)
            i=i-1
      endwhile

//add code to solve puzzle
```

Question			Answer	Marks	Guidance
		(ii)	Individual steps in pseudocode: • Checks if pointer is 0 (i.e. pointer is 0) • Checks if the disk being added is smaller than the one at the top of the pole • Puts diskValue at the top of stack • Move the pointer up one • If it's not a valid move, no disk is added • …and prints Invalid move Programming marks to be awarded as follows: • Checks if pointer is 0 and checks if the disk being added is smaller than the one at the top of the pole (1 – AO 3.2). • Puts diskValue at the top of stack and moves the pointer up one (1 – AO 3.2). • If it's not a valid move, no disk is added and prints Invalid move (1 – AO 3.2). Possible annotated comments: • The IF statement checks the pointer value is 0 so we can assume the disk is available to be moved (1 – AO 2.2). • If these conditions are met then pole value is set to the diskValue and incremented by 1 so the disk can move to the next pole (1 – AO 2.2). • If these conditions are not met then the else prints Invalid move (1 – AO 2.2).	6 AO2.2 (3) AO3.2 (3)	Up to 3 marks for valid pseudocode (AO3.2). Up to 3 marks for annotated comments used (AO2.2). Example pseudocode: `If pointer==0 or pole[pointer]>diskValue then` `  pole[pointer]=diskValue` `  pointer=pointer+1` `else` `  print("Invalid move")` `endif` Example of pseudocode with comments in code for guidance: `If pointer==0 or` `pole[pointer]>diskValue then` `  pole[pointer]=diskValue` `  pointer=pointer+1` `else` `  print("Invalid move")` `endif`
	(c)		Individual steps in pseudocode: Makes a swap between 1 and 3 …..and always makes a valid swap. Then makes a swap between 1 and 2. …. And always makes a valid swap ….only if the problem is not solved. Repeats if not solved. Will not attempt to make a swap between 2 empty towers. Programming marks to be awarded as follows: • Makes a swap between pole 1 and 3 (1 – AO 3.2). • Checks for a valid move throughout (1 – AO 3.2). • Makes a swap between pole 1 and 2 (1 – AO 3.2). • Checks whether problem is solved throughout (1 – AO 3.2). • Makes a swap between 3 and 2 (1 – AO 3.2). • Repeats if not solved and will not attempt a swap between two empty towers. (1 – AO 3.2). • Possible annotated comments: • The loop checks if towers 1 or 2 have anything on them. If they do the problem is not solved as all the disks need to be on pole 3. (1 – AO 2.2). • The first selection (if) makes a swap between 1 and 2 and always makes a valid swap only if the problem is not solved. (1 – AO 2.2). • The second selection (if) makes a swap between 3 and 2 and always makes a valid swap only if the problem is not solved. (1 – AO 2.2). • A swap is only valid if the pole is either empty or the disk on which it is to be put is a larger number (1 – AO 2.2).	10 AO2.2 (4) AO3.2 (6)	Up to 6 marks for valid pseudcode (AO3.2) Up to 4 marks for annotated comments used (AO2.2). Example pseudocode: `while tower1.peek()!=999 or` `tower2.peek()!=999` `  if tower1.peek()<tower3.peek() then` `  disk=tower1.pop()` `  tower3.push(disk)` `  elseif tower3.peek()<tower1.peek() then` `  disk=tower3.pop()` `  tower1.push(disk)` `  endif` `if  tower1.peek()!=999 or tower2.peek()!=999` `then` `  if tower1.peek()<tower2.peek() then` `  disk=tower1.pop()` `  tower2.push(disk)` `  elseif tower2.peek()<tower1.peek()` `then` `  disk=tower2.pop()` `  tower1.push(disk)` `  endif` `  endif` `if  tower1.peek()!=999 or tower2.peek()!=999` `then` `  if tower3.peek()<tower2.peek() then` `  disk=tower3.pop()` `  tower2.push(disk)` `  elseif tower2.peek()<tower3.peek()` `then` `  disk=tower2.pop()` `  tower3.push(disk)` `  endif` `  endif` `endwhile`

Section B: Animal Monopoly (OOP)

Section B

5 Kim is writing an object-oriented program for a four player board game. The board has 26 squares that players move around, as shown in Fig. 5.1.

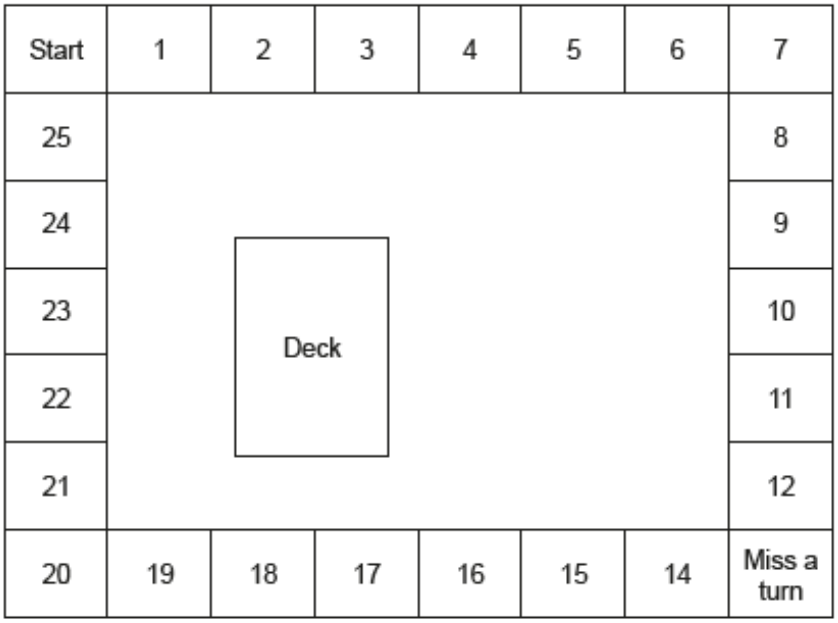


Fig. 5.1

Each player takes it in turn to roll two dice. They then move that number of spaces on the board. If they roll a double (both dice have the same value), they then take a card from the deck. The deck contains 40 cards that each include a sentence (such as "You have won the lottery"). The sentence on the card determines if money is given or taken away from the player.

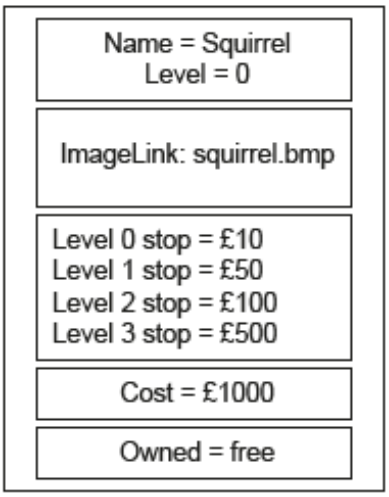


Fig. 5.2

Each square (apart from Start and Miss a turn) has an animal associated with it that the player can purchase, if it has not been purchased already, for example square 6 has a Squirrel. Fig. 5.2 shows an example of one of these animals. Once a player has purchased the animal, any opposing player which subsequently lands on the square/animal has to pay a fine.

Each animal can be upgraded, with each upgrade the game charges more each time a player stops on them. For example, with no upgrade the level 0 squirrel costs £10 when a player stops on it. If £1000 is paid to upgrade, the squirrel is then a level 1 animal and now charges £50 for a stop.

The cost to purchase and upgrade the animal is the same.

Each animal can be upgraded to a maximum of level 3.

When a player lands on, or passes the square 'Start' (position 0), they receive £500. If they land on 'Miss a turn' (position 13), they miss their next turn.

OOP (constructor, instance)

(a) (i) A class, `Player`, stores the player's ID (P1, P2, P3, P4), their current board position and the amount of money they have.

Fig. 5.3 shows a class diagram for `Player`. A class diagram describes a class. It contains the class name, followed by the attributes, then the methods.

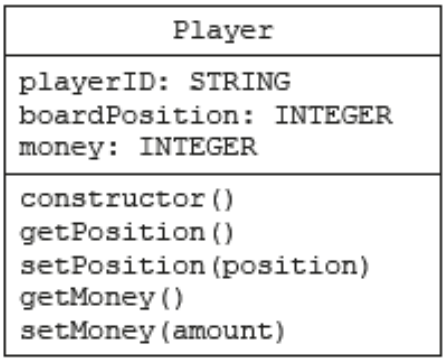


Fig. 5.3

The constructor creates a new instance of `Player`, taking the player's ID as a parameter. The board position is set to 0, and money to £2000.

Write, using pseudocode, the constructor method for the `Player` class.

.....

.....

.....

.....

..... **[3]**

(ii) A class, `Animal`, define the attributes and methods for the animals stored in each square.

Fig. 5.4 shows a class diagram for `Animal`.

Animal
name: STRING currentLevel: INTEGER cost: INTEGER L0: REAL L1: REAL L2: REAL L3: REAL imageLink: STRING setSquare: INTEGER owned: STRING
constructor() getCost() upgrade(player) getCurrentLevel() setOwned(player) getOwned() getAmountToCharge() getName()

Fig. 5.4

The constructor takes the required data as parameters and then sets `currentLevel` to 0, and assigns the parameters as the remaining attributes for the new object.

Write, using pseudocode, the constructor method for the `Animal` class. **[4]**

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

(iii) Write, using pseudocode, the code to create an instance of `Animal` for the Squirrel shown in Fig. 5.2, positioned on square number 6, for the constructor function you wrote in part **(a)(ii)**.

.....

.....

..... **[2]**

(b) The board is stored as a 1D array, `board`, of data type `Animal`. The spaces at 0, and 13, are left as empty elements that are checked using separate functions.

(i) Complete, using pseudocode, the function to:

- Roll both dice
- Move the player, the dice number of spaces
- If a double is rolled, calls the procedure `pickDeck`
- Adds £500 if they have passed or landed on Start
- Calls the procedure `missAGo` if they land on space 13 or
- Calls the procedure `checkAnimal`
- Return the new position

```
function playerMove(currentPlayer)

    dice1 = random(1,6)

    dice2 = random(1,6)

    boardPosition = ........................... + dice1 + dice2

    if ..............  == dice2 then

       pickDeck(currentPlayer)

    endif

    if position > 25 then

       currentPlayer.setMoney(currentPlayer.getMoney() +  .............)

       position = position -  ..............................

    endif

    if position == ................................ then

       missAGo(currentPlayer)

    elseif position != 0 then

       checkAnimal(currentPlayer)

    endif

    ..............................................................

endfunction
```

[6]

5	a	i	1 mark per bullet to max 3 • Declaring the procedure and taking a player ID as parameter • Setting `playerID` to parameter • Setting `boardPosition` to 0 and money to 2000 e.g. `public procedure new(thePlayerID)` `  playerID = thePlayerID` `  boardPosition = 0` `  money = 2000` `endprocedure`	3 AO2.2 (2) AO3.2 (1)	

5	a	ii	1 mark per bullet to max 4 • Declaration as public and procedure, named constructor/new and Taking all correct parameters (missing currentLevel) • Sets currentLevel to 0 • Setting all the data… • …to the matching parameters taken e.g. `public procedure new(theName, theCost, theL0, theL1,` `theL2, theL3, theImageLink, theSetSquare, theOwned)` `  name = theName` `  currentLevel = 0` `  cost = theCost` `  L0 = theL0` `  L1 = theL1` `  L2 = theL2` `  L3 = theL3` `  imageLink = theImageLink` `  setSquare = theSetSquare` `  owned = theOwned` `endprocedure`	4 AO2.2 (2) AO3.2 (2)	
5	a	iii	1 mark per bullet to max 2 • Creating new instance e.g. squirrel = new Animal • Parameters matching part (b)(i) e.g.	2 AO2.1 (2)	

5	b	i	1 mark for each correctly completed space `function playerMove(currentPlayer)` `  dice1 = random(1,6)` `  dice2 = random(1,6)` `  position = currentPlayer.getPosition() +` `dice1 + dice2` `  if dice1 == dice2 then` `  pickDeck(currentPlayer)` `  endif` `  if position > 25 then` `  currentPlayer.setMoney(currentPlayer.getMoney() +` `500)` `  position = position - 26` `  endif` `  if position == 13 then` `  missAGo(currentPlayer)` `  elseif position != 0 then` `  checkAnimal(currentPlayer)` `  endif` `  return position` `endfunction`	6 AO2.2 (3) AO3.2 (3)	

OOP (methods, queue)

(c) The deck is stored as a zero-indexed 1D array, named `deck`, of type `Card`.

The class diagram for `Card` is shown in Fig. 5.5.

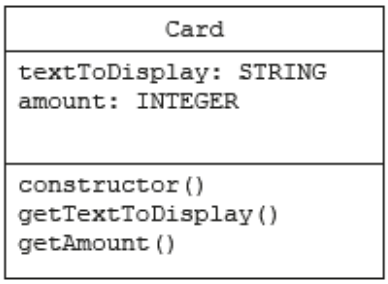


Fig. 5.5

The array, `deck`, is treated as a queue, with a variable, `headPointer`, identifying the first card in the deck. When a card has been used, the head pointer increases to move to the next position. If the end of the deck is reached, the head pointer returns to 0 and starts again.

The procedure `pickDeck`:

- takes the current player as a parameter
- outputs the text to be displayed from the first card in the queue
- adds or subtracts the amount to/from the current player's money
- increases the head pointer

Write, using pseudocode, the procedure `pickDeck`.

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

.....

..... **[6]**

5	c		1 mark per bullet to max 6 • Procedure declaration with correct name and parameter • Outputting the correct text from deck at `headPointer` • Sending to `currentPlayer.setMoney` ... • `getMoney` + `deck` at head pointer amount • Increase the head pointer • Set `headPointer` to 0 if position 40 or greater `procedure pickDeck(currentPlayer)` `  output(deck[headPointer].getTextToDisplay())` `  amount = deck[headPointer].getMoney()` `  currentPlayer.setMoney(currentPlayer.getMoney() + amount)` `  headPointer = headPointer + 1` `  if headPointer == 40 then` `  headPointer = 0` `  endif` `endprocedure`	6 AO2.2 (4) AO3.2 (2)	

OOP (functions, parameters, selection)

(d) The procedure `checkAnimal`:

- Takes the current player as a parameter
- Accesses the data for the animal at the player's position in the array `board`
- If the animal is free, asks the player if they would like to purchase the animal and outputs its `name` and `cost`, if they choose to buy the animal, it calls the procedure `purchase()` with the player and animal as parameters
- If that player owns the animal, and it is not at level 3, it asks if they would like to upgrade the animal
- If they would like to upgrade, it calls the method `upgrade` for that animal with the current player as a parameter
- If a different player owns the animal, it calls the method `getAmountToCharge()` for that animal, sending this value and the current player as parameters to the procedure `chargeStay()`

Write, using pseudocode, the procedure `checkAnimal`. **[10]**

5	d		• Declaring the procedure with correct parameters • Check if the space/animal is free ○ If free, outputting name and cost ○ Checking if they want to buy ▪ Calling purchase with current player and animal • If they own the animal, checking if they can upgrade ▪ If they can, asking if they want to upgrade outputting the cost ▪ If they want to, calling the `upgrade` method • If they don't own the animal ○ Calling `chargeStay` with the amount to charge and the current player	10 AO2.2(5) AO3.2(5)	Allow follow through for incorrect accessing of methods

e.g.

```
procedure checkAnimal(currentPlayer)

   if board[currentPlayer.getPosition()].owned == "free"
      answer = input("Would you like to purchase ",
                board[position].getName(), "? It costs ",
                board[position].getCost()

      if answer = "yes" then
         purchase(currentPlayer, board[position])
      endif

   elseif board[currentPlayer.getPosition()].getOwned() == currentPlayer

       if board[currentPlayer.getPosition()].getCurrentLevel()
!= "L3"
          answer = input("Upgrade? It costs ",
board[position].getCost())

           if answer == "yes" then
              board[currentPlayer.getPosition()].upgrade(currentPlayer)
           endif

        endif

   else
      amount = board[position].getAmountToCharge()
      chargeStay(amount, currentPlayer)
   endif
endprocedure
```

#TODO

- zig zag
- other resources
- old spec
- other boards
- books
- sites