2.3 Stacks

Name: ____________________

Class: ____________________

Date: ____________________

Time: **202 minutes**

Marks: **141 marks**

Comments:

Q1.

To evaluate an expression in Reverse Polish notation, you start from the left hand side of the expression and look at each item until you find an operator (eg + or −).

This operator is then applied to the two values immediately preceding it in the expression. The result obtained from this process replaces the operator and the two values used to calculate it. This process continues until there is only one value in the expression, which is the final result of the evaluation.

For example `5 2 7 + +` would change to `5 9 +` after the first replacement.

Explain how a stack could be used in the process of evaluating an expression in Reverse Polish notation.

____

____

____

____

____

____

(Total 3 marks)

Q2.

Two frequently completed actions when using a particular piece of software are **undo** and **repeat**.

The **undo** action results in the state changing from the current state to the state previous to the user's most recent action, eg if the last action the user completed was to change the font of a selected piece of text from `Courier New` to Chiller then if the **undo** action is selected the result will be to change the font of that text back to `Courier New`.

The user is able to keep using the **undo** action to go back through all previous states.

The **repeat** action results in the user's most recent action being applied again, eg if the last action the user completed was to change the font of a piece of text to Chiller then if the **repeat** action is selected the result will be to change the font of the currently selected text to Chiller.

The user is able to keep using the **repeat** action to apply the most recent action multiple times. The **repeat** action will only work when there is a most recent action that can be applied again.

(a) Explain how a single stack can be used in the implementation of the repeat action **and** the undo action.

____

____

(5)

(b) State the type of error that occurs if the user tries to complete an undo action before they have completed any other actions.

(1)
(Total 6 marks)

Q3.

A graph can be drawn to represent a maze. In such a graph, each graph vertex represents one of the following:

- the entrance to or exit from the maze
- a place where more than one path can be taken
- a dead end.

Edges connect the vertices according to the paths in the maze.

Diagram 1 shows a maze and **Diagram 2** shows one possible representation of this maze.
Position 1 in **Diagram 1** corresponds to vertex 1 in **Diagram 2** and is the entrance to the maze. Position 7 in **Diagram 1** is the exit to the maze and corresponds to vertex 7.

Dead ends have been represented by the symbol ——┤ in **Diagram 2**.

Diagram 3 shows a simplified undirected graph of this maze with dead ends omitted.

Diagram 1

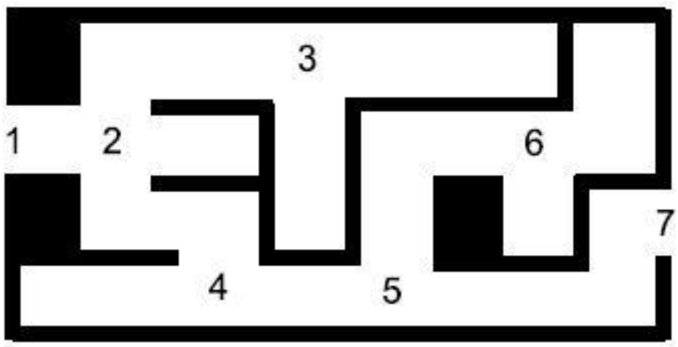
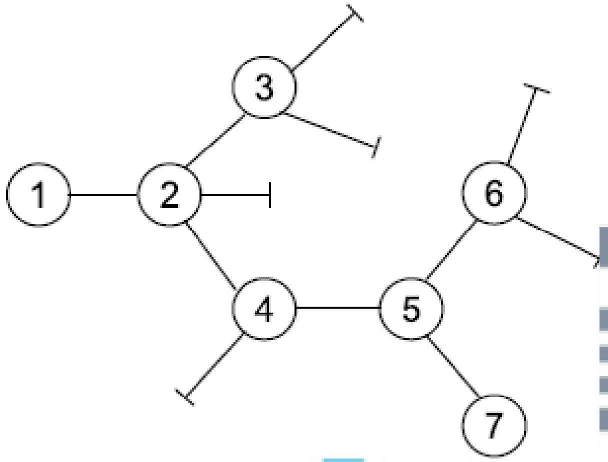
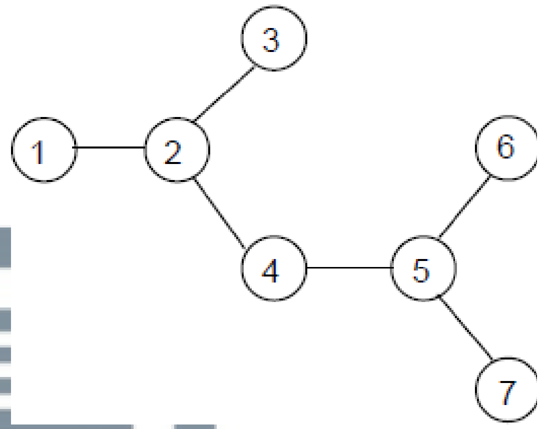


Diagram 2



Representation of maze including dead ends

Diagram 3



Graph representing maze with dead ends omitted

(a) The graph in **Diagram 3** is a tree.

State **one** property of the graph in **Diagram 3** that makes it a tree.

____

____

(1)

(b) The graphs of some mazes are not trees.

Describe a feature of a maze that would result in its graph **not** being a tree.

____

____

(1)

(c) Complete the table below to show how the graph in **Diagram 3** would be stored using an adjacency matrix.

(2)

(d) (i) What is a *recursive routine?*

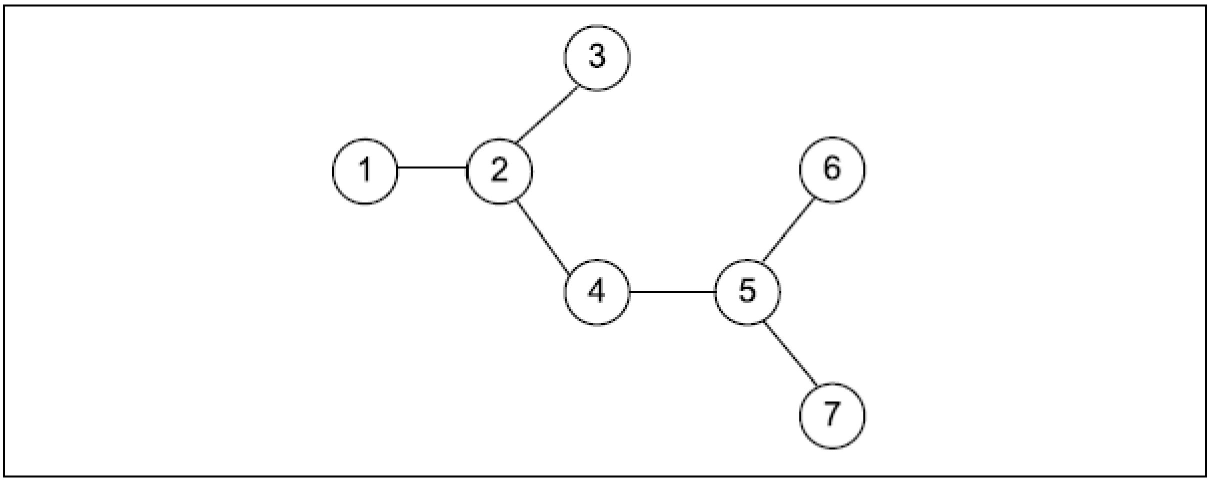
(1)

(ii) To enable the use of recursion a programming language must provide a stack.

Explain what this stack will be used for and why a stack is appropriate.

(2)

Diagram 3 is repeated here so that you can answer Question (e) without having to turn pages.



(e) A recursive routine can be used to perform a depth-first search of the graph that represents the maze to test if there is a route from the entrance (vertex 1) to the exit (vertex 7).

The recursive routine in the diagram below is to be used to explore the graph in **Diagram 3**. It has two parameters, `V` (the current vertex) and `EndV` (the exit vertex).

```
Procedure DFS(V, EndV)
    Discovered[V] ← True
    If V = EndV Then Found ← True
    For each vertex U which is connected to V Do
        If Discovered [U] = False Then DFS(U, EndV)
    EndFor
    CompletelyExplored[V] ← True
EndProcedure
```

Complete the trace table below to show how the `Discovered` and `CompletelyExplored` flag arrays and the variable `Found` are updated by the algorithm when it is called using `DFS(1,7)`.

The details of each call and the values of the variables `V`, `U` and `EndV` have already been entered into the table for you. The letter `F` has been used as an abbreviation for `False`. You should use `T` as an abbreviation for `True`.

				Discovered							CompletelyExplored							
Call	V	U	EndV	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	Found
	-	-		F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(1,7)	1	2	7															
DFS(2,7)	2	1	7															
		3	7															
DFS(3,7)	3	2	7															
DFS(2,7)	2	4	7															
DFS(4,7)	4	2	7															
		5	7															
DFS(5,7)	5	4	7															
		6	7															
DFS(6,7)	6	5	7															
DFS(5,7)	5	7	7															
DFS(7,7)	7	5	7															
DFS(5,7)	5	-	7															
DFS(4,7)	4	-	7															
DFS(2,7)	2	-	7															
DFS(1,7)	1	-	7															

(5)

(Total 12 marks)

Q4.

Reverse Polish Notation is an alternative to standard infix notation for writing arithmetic expressions.

(a) Convert the following Reverse Polish Notation expressions to their equivalent infix expressions.

Reverse Polish Notation	Equivalent Infix Expression
45 6 +	
12 19 + 8 *	

(2)

(b) State **one** advantage of Reverse Polish Notation over infix notation.

____

____

(1)

(c) The pseudo-code algorithm below can be used to calculate the result of evaluating a Reverse Polish Notation expression that is stored in a string. The algorithm is designed to work only with the single digit denary numbers 0 to 9. It uses procedures and functions listed in the table below, two of which operate on a stack data structure.

```
StringPos ← 0
Repeat
  StringPos ← StringPos + 1
  Token ← GetCharFromString(InputString, StringPos)
  If Token = '+' Or Token = '-' Or Token = '/' Or Token = '*'
    Then
      Op2 ←Pop()
      Op1 ← Pop()
      Case Token Of
        '+': Result ← Op1 + Op2
        '-': Result ← Op1 - Op2
        '/': Result ← Op1 / Op2
        '*': Result ←Op1 * Op2
      EndCase
      Push(Result)
    Else
      IntegerVal ←ConvertToInteger(Token)
      Push(IntegerVal)
  EndIf
Until StringPos = Length(InputString)
Output Result
```

Procedure/Function	Purpose	Example(s)
GetCharFromString (InputString:String , StringPos:Integer): Char	Returns the character at position StringPos within the string InputString. Note that the leftmost letter is position 1, not position 0.	GetCharFromString ("Computing", 1) would return the character 'C'. GetCharFromString ("Computing", 3) would return the character 'm'.
ConvertToInteger (ACharacter: Char): Integer	Returns the integer equivalent of the character in ACharacter.	ConvertToInteger('4') would return the integer value 4.
Length (AString: String): Integer	Returns a count of the number of characters in the string AString.	Length("AQA") would return the integer value 3.
Push (ANumber: Integer)	Puts the number in ANumber onto the stack.	Push(6) would put the number 6 on top of the stack.
Pop (): Integer	Removes the number from the top of the stack and returns it.	X ← Pop() would remove the value from the top of the stack and put it in X.

(d) Complete the table below to trace the execution of the algorithm when

`InputString` is the string: `64+32+*`

In the `Stack` column, show the contents of the stack once for each iteration of the `Repeat..Until` loop, as it would be at the end of the iteration.

The first row and the leftmost column of the table have been completed for you.

StringPos	**Token**	**IntegerVal**	**Op1**	**Op2**	**Result**	**Stack**
0	-	-	-	-	-	
1						
2						
3						
4						
5						
6						
7						

(5)

Final output of algorithm: ____

(1)

(e) A programmer is going to implement the algorithm above in a programming language that does not provide built-in support for a stack data structure.

The programmer intends to simulate a stack by using a fixed length array of 20 integers named `StackArray` with indices running from `1` to `20` and an integer variable `TopOfStackPointer` which will be initialised to `0`.

Write a pseudo-code algorithm for the `Push` operation to push a value stored in the variable `ANumber` onto the stack.

Your algorithm should cope appropriately with any potential errors that might occur.

____

____

____

____

____

____

____

____

(4)

(Total 13 marks)

Q5.

(a) State the principle of operation of a set of data values which behave as a stack.

____

____

(1)

(b) Memory locations 600 to 605 are to be used as a stack area to store character data, and the first value added to the stack is to be stored at address 600.

Figure 1 shows the initial empty state of the stack.

Figure 1

600	

601	
602	
603	
604	
605	

(i) Show on **Figure 2** the state of the stack after the characters ‘A’, ‘V’, ‘E’, ‘R’ and ‘Y’ join the stack.

Figure 2

600	
601	
602	
603	
604	
605	

(1)

(ii) Two items are removed from the stack. Show on **Figure 3** the state of the stack.

Figure 3

600	
601	
602	
603	
604	
605	

(1)

(iii) Two new characters ‘S’ and ‘P’ join the stack. Show on **Figure 4** the final state of the stack.

Figure 4

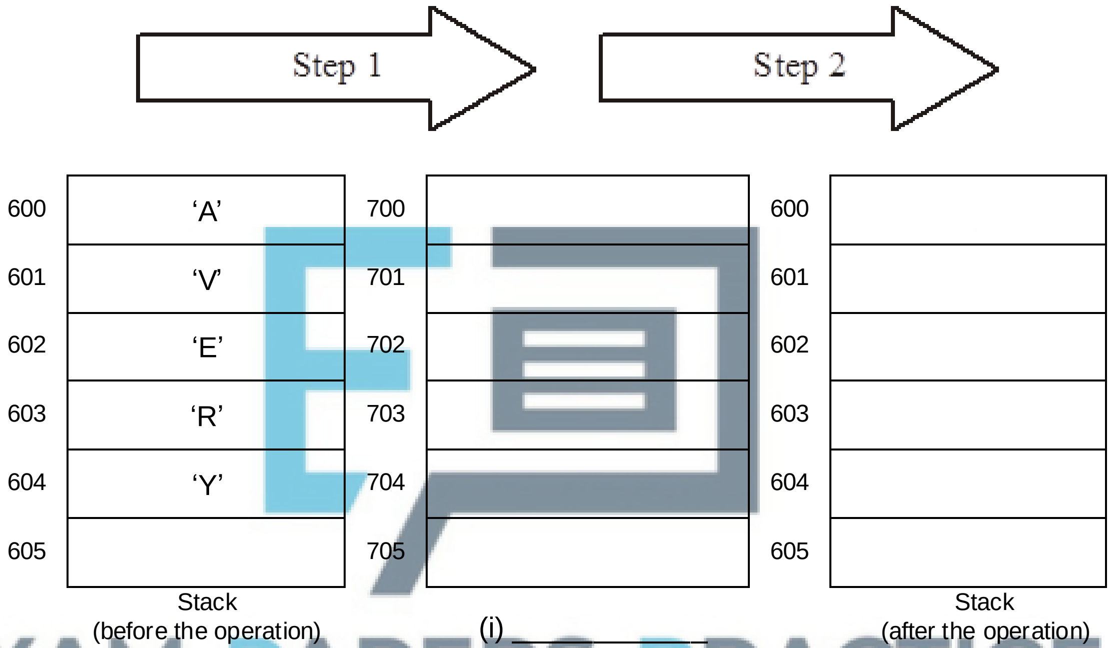
600	
601	

602	
603	
604	
605	

(1)

(c) The original items in this stack are to be reversed. This can be done using a second data structure which uses locations 700 to 705 respectively. The first item added to the stack was character ‘A’.

Figure 5



(i) Name the second data structure. Label **Figure 5.**

____

(1)

(ii) Describe **Step 1** in **Figure 5.**

____

____

(1)

(iii) Describe **Step 2** in **Figure 5.**

____

____

(1)

(iv) Show on **Figure 5** the final contents of all the memory locations.

(2)

(Total 9 marks)

Q6.

A *recursively-defined* procedure **ProcA** that takes two integers as parameters is defined below.

(a) What is meant by a recursively-defined procedure?

____

____

(1)

(b) What is the role of the stack when a recursively-defined procedure is executed?

____

____

(1)

(c) Dry run the procedure call **ProcA(11,1)** using the data in the array, **Items**, by completing the trace table below.

```
Procedure ProcA (Number,Entry)
      If Number <> Items[Entry]
      Then ProcA (Number,Entry+1)
      Else Output (Entry)
      EndIf
EndProc
```

Items

[1]	4
[2]	5
[3]	8
[4]	11
[5]	15
[6]	19
[7]	21
[8]	28
[9]	33

Number	Entry	Output
11	1	

(4)

(d) What is the purpose of this algorithm?

____

(1)

(e) Give a situation where this algorithm will fail.

____

____

(1)

(f) Suggest a modification to the algorithm that will prevent it from failing.

____

____

(1)

(g) With an ordered array, Items, of many more entries, what more efficient algorithm could be used to achieve your expressed purpose in part (d)?

____

____

(1)

(Total 10 marks)

Q7.

A stack may be implemented by using either an array or a linked list.

(a) Give a disadvantage of:

(i) an array implementation;

____

(1)

(ii) a linked list implementation.

____

(1)

(b) Under what circumstances would it be more appropriate to use:

(i) an array;

____

(1)

(ii) a linked list.

____

(1)

(Total 4 marks)

Q8.

(a) In the context of data structures what is meant by the terms:

(i) FIFO; ____

(ii) LIFO? ____

(2)

(b) Queue and stack are examples of data structures. Tick in the following table to indicate whether they are FIFO or LIFO data structures.

	FIFO	LIFO
Queue		
Stack		

(2)

(c) Describe **one** example of the use of a stack.

____

____

____

(2)

(d) Describe **one** example of the use of a Binary Search Tree.

____

____

____

(2)

(Total 8 marks)

Q9.

A *recursively-defined* procedure **Process**, which takes an integer as its single parameter, is defined below.

(a) What is meant by recursively-defined?

____

____

(1)

(b) Describe how a stack is used in the execution of procedure **Process**?

____

____

____

____

(2)

(c) Dry run the procedure call **Process(1)**, using the data in the table below, showing clearly the order the values are printed.

```
 Procedure Process (P)
  Print (P)
  If Table[P].Left <> 0
     Then Process (Table[P].Left)
  EndIf
  Print (Table[P].Data)
  If Table[P].Right <> 0
     Then Process (Table[P].Right)
  EndIf
EndProcedure
```

		Table	
	Data	Left	Right
[1]	Jones	3	2
[2]	Smith	0	0
[3]	Bremner	5	4
[4]	Fortune	0	0
[5]	Bird	0	0

Printed Output:= ____

____

(6)

(d) What does procedure Process describe?

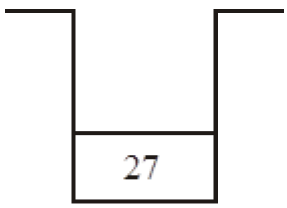
____

(1)

(Total 10 marks)

Q10.

A stack is a type of abstract data type (ADT) that is often known as a LIFO data type. A stack with a single element 27 may be drawn as follows:



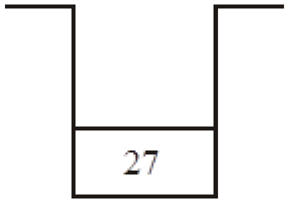
(a) What is the meaning of the term LIFO?

____

(1)

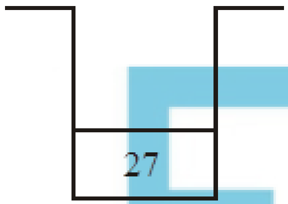
(b) A stack has two operations, **Push** and **Pop**. **Push n** adds item **n** to stack. **Pop** removes one item from the stack. A number of operations are performed, **in sequence**, on the stack drawn above. Using the stack diagrams below show the effect of this sequence of operations.

(i) Push 5



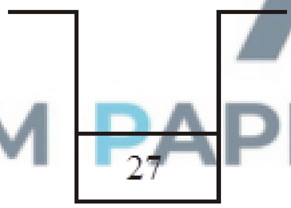
(1)

(ii) Push 9



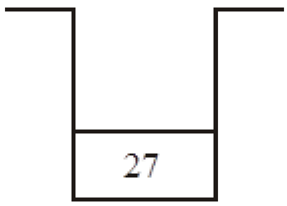
(1)

(iii) Pop



(1)

(iv) Push 6



(1)

(c) Give **one** example of the use of a stack.

____

(1)

(Total 6 marks)

Q11.

A *recursively-defined* procedure B, which takes an integer as its single parameter, is defined below. The operators DIV and MOD perform integer arithmetic.
x DIV y calculates how many times y divides exactly into x. For example 7 DIV 3 = 2
x MOD y calculates the remainder that results. For example 7 MOD 3 = 1.

```
Procedure B (Number)
  If (Number = 0) OR (Number = 1)
    Then Print (Number)
    Else
     B (Number DIV 2)
     Print (Number MOD 2)
  EndIf
EndProcedure
```

(a) What is meant by recursively-defined?

____

____

(1)

(b) Why is a stack necessary to execute procedure B recursively?

____

____

(1)

(c) Dry run the procedure call *B(53)* showing clearly the values of the parameter and the printed output for the six calls of *B*.

Call Number	Parameter
1	53
2	26
3	13
4	
5	
6	

Printed Output: ____

(6)

(d) What process does procedure B describe? ________________________________

(1)

(Total 9 marks)

Q12.

Describe how the elements in a non-empty queue are reversed with the aid of a stack.

(Total 4 marks)

Q13.

The following data is input to a program, in alphabetical order, and is stored.

Anne
Bob
Claire
Dean

(a) Draw a diagram to show how this data is stored for:

(i) a stack;

(ii) a queue.

(4)

(b) One item is retrieved from these data structures for processing, and Eden is input.

Draw the diagrams of this new situation for:

(i) the stack;

(ii) the queue.

(3)

(c) Why are queues in computer systems usually implemented as circular queues?

____

____

____

____

(2)

(Total 9 marks)

Q14.

The list Ports contains the following names:

[Southampton, Barcelona, Athens, Alexandria, Tunis, Lisbon]

The table below shows some functions which take a list as their single argument and return a result which is either an element of a list or a boolean value.

Head(list) – If the list is non-empty, it returns the element at the head of the list (e.g. Head (Ports) → Southampton) otherwise it reports an error
Tail(list) – If the list is non-empty it returns a new list containing all but the first element of the original list, otherwise it reports an error
Empty(list) – if the list is the empty list it returns True otherwise it returns False. The empty list is donated by [ ]

(a) What result is returned when the following function calls are made?

(i) Tail(Ports) ____

____

(1)

(ii) Head(Tail(Tail(Ports))) ____

____

(2)

(iii) Empty(Tail(Tail(Tail(Tail(Tail(Tail(Ports)))))) ___________________________

____

(2)

A *recursively defined* procedure P, which takes a list as its single parameter, is defined below.

```
Define Procedure P(list)
     If Not Empty(list)
       Then
         P(Tail(list))
         Print Head(list)
     EndIf
EndDefine
```

(b) What is meant by recursively defined?

____

____

(1)

(c) Explain why a stack is needed to execute procedure P recursively.

____

____

____

(2)

(d) For the procedure call P(Ports) give the PRINTed output in the order in which it is produced.

____

____

(4)

(e) Complete the table to show the list Ports as a linked list so that the ports can be accessed in alphabetical order.

1	Southampton	
2	Barcelona	
3	Athens	
4	Alexandria	
5	Tunis	
6	Lisbon	

Head Pointer

(2)
(Total 14 marks)

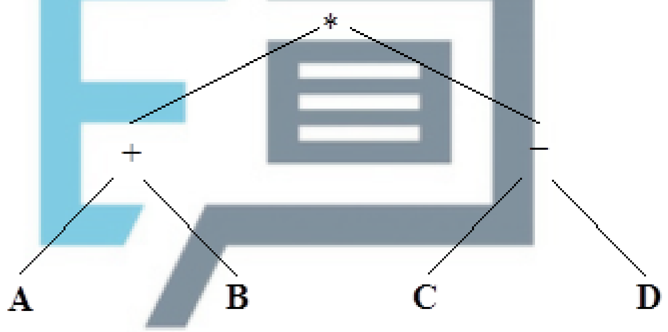
Q15.

Explain how the elements in a non-empty queue may be reversed with the aid of a stack.

(Total 4 marks)

Q16.

An algebraic expression is represented in a binary tree as follows:



(a) Label its *root*, a *branch* and a *leaf node.*

(3)

(b) Mark and label the *left sub-tree* and the *right sub-tree* of this tree.

(2)

A recursively-defined procedure T, which takes a tree structure, tree(x, y, z) as its single parameter, where x is the root, y is the left sub-tree and z is the right sub-tree, is defined below (<> means not equal to).

```
Procedure T (tree(x, y, z))

  If y <> empty
    Then
          PRINT ')'
          T(y)
 EndIf
    PRINT x
    If z <> empty
    Then
          T(z)
         PRINT ')'
```

```
    EndIf
EndProc
```

(c) What is meant by *recursively-defined*?

____

____

(1)

(d) Explain why a stack is necessary in order to execute procedure T recursively.

____

____

____

____

____

____

(3)

(e) Dry run the following procedure call

```
T  (    tree( '*', tree ('+', tree ('A', empty, empty), tree ('B', empty, empty) ),
               tree ('–', tree ('C', empty, empty), tree ('D', empty, empty) )
          )
)
```

showing clearly the PRINTed output and the values of the parameter omitted from the table (rows 4, 5, 6, 7) for the **seven** calls of T.

Call Number	Parameter
1	tree('*', tree('+', tree('A',empty,empty), tree('B',empty,empty)), tree('–', tree('C',empty,empty), tree('D',empty,empty)))
2	tree(' +', tree('A',empty,empty), tree('B',empty,empty))
3	tree('A',empty,empty)
4	
5	
6	
7	

(10)

(f) What tree traversal algorithm does procedure T describe?

(1)
(Total 20 marks)