2.3 Stacks Mark Scheme

Q1.

All marks AO1 (understanding)

(Starting at LHS of expression) push values/operands onto stack; **R.** if operators are also pushed onto stack

Each time operator reached pop top two values off stack (and apply operator to them);

Add result (of applying operator) to stack;

Max 2 if any errors **Max 2** if more than one stack used
Note for examiners: award 0 marks if description is not about a stack / LIFO structure even if the word "stack" has been used

[3]

Q2.

(a) **Marks are for AO2 (analyse)**

1. Stack / data structure is used to store the (user's) actions; **A**. by implication

2. Each time an action is completed it is pushed/added onto the **top** of the stack;

3. unless it is an undo (or repeat) action;

4. When repeat action is used the top item from the stack is used to indicate the action to complete // when repeat action is used the result of peek function is used to indicate the action to complete; **R**. implication that top item of stack is popped/deleted from stack – unless it is clear it is subsequently pushed/added back to the stack **A**. when repeat action is used a copy of the top item from the stack is pushed/added to the top of the stack

5. When undo action is used the top item is popped/removed from the stack of actions;

5

(b) **Mark is for AO1 (understanding)**

Stack empty (error) // (stack) underflow;

1

[6]

Q3.

(a) **Connected** // There is a path between each pair of vertices;
Undirected // No direction is associated with each edge;
Has no cycles // No (simple) circuits // No closed chains // No closed paths in which all the edges are different and all the intermediate vertices are different // No route from a vertex back to itself that doesn't use an edge more than once or visit an intermediate vertex more than once;
A no loops

Alternative definitions:
A simple cycle is formed if any edge is added to graph;
Any two vertices can be connected by a unique simple path;

Max 1

(b) No route from entrance to exit / through maze;
Maze contains a loop/circuit ;
A more than one route through maze;
Part of the maze is inaccessible / enclosed;
R Responses that clearly relate to a graph rather than the maze

Max 1

(c)

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2	1	0	1	1	0	0	0
3	0	1	0	0	0	0	0
4	0	1	0	0	1	0	0
5	0	0	0	1	0	1	1
6	0	0	0	0	1	0	0
7	0	0	0	0	1	0	0

(allow some symbol in the central diagonal to indicate unused)

or

	1	2	3	4	5	6	7
1	0	1	0	0	0	0	0
2		0	1	1	0	0	0
3			0	0	0	0	0
4				0	1	0	0
5					0	1	1
6						0	0
7							0

(with the shaded portion in either half – some indication must be made that half of the matrix is not being used. This could just be leaving it blank, unless the candidate has also represented absence of an edge by leaving cells blank)

1 mark for drawing a 7x7 matrix, labelled with indices on both axis and filled only with 0s and 1s, or some other symbol to indicate presence/absence of edge. e.g. T/F. Absence can be represented by an empty cell.
1 mark for correct values entered into matrix, as shown above;

2

(d) (i) Routine defined in terms of itself // Routine that calls itself;
A alternative names for routine e.g. procedure, algorithm
NE repeats itself

1

(ii) Stores return addresses;
Stores parameters;
Stores local variables; NE temporary variables
Stores contents of registers;
A To keep track of calls to subroutines/methods etc.

Max 1

Procedures / invocations / calls must be returned to in reverse order (of being called);
As it is a LIFO structure;
A FILO
As more than one / many return addresses / sets of values may need to be stored (at same time) // As the routine calls itself and for each call/invocation a new return address / new values must be stored;

Max 1

2

(e)

				Discovered							Completely Explored							
Call	V	U	End V	1	2	3	4	5	6	7	1	2	3	4	5	6	7	F
	-	-	7	F	F	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(1,7)	1	2	7	**T**	F	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(2,7)	2	1	7	T	**T**	F	F	F	F	F	F	F	F	F	F	F	F	F
		3	7	T	T	F	F	F	F	F	F	F	F	F	F	F	F	F
DFS(3,7)	3	2	7	T	T	**T**	F	F	F	F	F	F	**T**	F	F	F	F	F
DFS(2,7)	2	4	7	T	T	T	F	F	F	F	F	F	T	F	F	F	F	F
DFS(4,7)	4	2	7	T	T	T	**T**	F	F	F	F	F	T	F	F	F	F	F
		5	7	T	T	T	T	F	F	F	F	F	T	F	F	F	F	F
DFS(5,7)	5	4	7	T	T	T	T	**T**	F	F	F	F	T	F	F	F	F	F
		6	7	T	T	T	T	T	F	F	F	F	T	F	F	F	F	F
DFS(6,7)	6	5	7	T	T	T	T	T	**T**	F	F	F	T	F	F	**T**	F	F
DFS(5,7)	5	7	7	T	T	T	T	T	T	F	F	F	T	F	F	T	F	F
DFS(7,7)	7	5	7	T	T	T	T	T	T	**T**	F	F	T	F	F	T	**T**	**T**
DFS(5,7)	5	-	7	T	T	T	T	T	T	T	F	F	T	F	**T**	T	T	T
DFS(4,7)	4	-	7	T	T	T	T	T	T	T	F	F	T	**T**	T	T	T	T
DFS(2,7)	2	-	7	T	T	T	T	T	T	T	F	**T**	T	T	T	T	T	T
DFS(1,7)	1	-	7	T	T	T	T	T	T	T	**T**	T	T	T	T	T	T	T

1 mark for having the correct values changes in each region highlighted by a rectangle and no incorrect changes in the region. Ignore the contents of any cells that are not changed.

A alternative indicators that clearly mean True and False.
A it is not necessary to repeat values that are already set (shown lighter in table)

5

[12]

Q4.

(a)

Reverse Polish Notation	Equivalent Infix Expression
45 6 +	*45 + 6* ***R*** *6 + 45*
12 19 + 8 *	*(12 + 19) * 8* ***R*** *12+19*8, (19+12)*8* ***A*** *x for **

2

(b) Simpler for a machine / computer to evaluate // simpler to code algorithm
A easier **R** to understand
Do not need brackets (to show correct order of evaluation/calculation);
Operators appear in the order required for computation;
No need for order of precedence of operators;
No need to backtrack when evaluating;
A RPN expressions cannot be ambiguous as **BOD**

1

(c)

String Pos	Token	Integer Val	Op1	Op2	Result	Stack
0	-	-	-	-	-	
1	6	6				6
2	4	4				4 6
3	+		**6**	**4**	10	10
4	3	3				3 10
5	2	2				2 3 10
6	+		**3**	**2**	5	5 10
7	*		**10**	**5**	50	50

Output : **50**

1 mark for each of rows 1–3
1 mark for rows 4 and 5 together
1 mark for rows 6 and 7 together
1 mark for correct final output
Values of Op1 and Op2 MUST be assigned in rows 3, 6 and 7 to award the marks for these rows. They cannot be inferred from incorrectly entered previous values.

***I** values in empty cells, even if they are incorrect.*

6

(d)
```
If StackArray is full
  Then Stack Full Error
  Else
    Increment TopOfStackPointer
    StackArray [TopOfStackPointer] ←
        ANumber
EndIf
```

1 mark for appropriate `If` structure including condition (does not need both `Then` and `Else`) – Do not award this mark if `ANumber` is put into `StackArray` outside the `If`.
1 mark for reporting error in correct place
1 mark for incrementing `TopOfStackPointer`*
1 mark for storing value in `ANumber` into correct position in array*
** = if the store instruction is given before the increment instruction OR the `If` structure then award **Max 1** of these two marks UNLESS the item is inserted at position `TopOfStackPointer+1` so the code would work.*

I initialisation of `TopOfStackPointer` to 0
A `TopOfStackPointer=20/>=20` for Stack is full
A Logic of `if` structure reversed i.e. If stack is not full / `TopOfStackPointer<20 / <>20/!=20` and `Then, Else` swapped
A Any type of brackets or reasonable notation for the array index
DPT If candidate has used a different name any variable then do not award first mark but award subsequent marks as if correct name used.
Refer answers where candidate has used a loop to find position to insert item into stack to team leaders.

4

[13]

Q5.

(a) Last (item) in, is the first (item) out / first (item) in is the last (item) out ;
R LIFO / FILO

1

(b) (i)

600	'A'
601	'V'
602	'E'
603	'R'
604	'Y' ;

605	

All items in the correct locations

1

(ii)

599	
600	‘A’
601	‘V’
602	‘E’ ;
603	
604	
605	

Correct three items // ft from an incorrect (i) including 605 as the first location used ;
A ‘R’ and ‘Y’ entries indicated in some way as ‘deleted’

1

(iii)

600	‘A’
601	‘V’
602	‘E’
603	‘S’
604	‘P’ ;
605	

Correct list of five items // ft from an incorrect (i) + a correct ft (ii) including 605 as the first location used ;

1

(c) (i) Queue ;
A First In – First Out FIFO / LILO

1

(ii) Items are removed/popped from the stack (one at a time) (and items are then added to the queue);

1

(iii) Items leave the queue on a ‘first in-first out’ basis ; **A** from the front of the queue

1

(iv) ‘Y’, ‘R’, ‘E’, ‘V’, ‘A’ on the queue ;

Y', 'R', 'E', 'V', 'A' on the final stack ;
A using 701 for the first queue location

2

[9]

Q6.

(a) A procedure that is defined in terms of itself;
A A procedure that calls itself
R re-entrant

1

(b) Store return addresses;
Store parameters;
Store local variables/ return values;

Max 1

(c)

Number	Entry	Output
11	1	
11	2;	
11	3;	
11	4;	4;

4

(d) A linear search//
To find/output the position/index of Number in Items;

1

(e) Number is not an entry in Items// Stack overflows;

1

(f) Test for reaching the end of Items;

1

(g) Binary Search;
An iterative solution;

Max 1

[10]

Q7.

(a) (i) Empty entries <u>waste</u> space // Maximum/fixed/static size
A stack may overflow

1

(ii) Space used by <u>pointers</u> // more complex to <u>program</u>;

1

(b) (i) The size of the stack /amount of data is known/limited/predictable
Memory saved since no pointers (if not given in a (ii))

R easier to program

1

(ii) The size of the stack is unknown//
The stack is volatile/ number of items fluctuates widely;

1

[4]

Q8.

(a) (i) First In First Out;
or by description

(ii) Last In First Out;
or by description

2

(b)

	FIFO	LIFO
Queue	✓	
Stack		✓

2

(c) Reverse the contents of a queue/list;
Push all contents of queue/list onto stack then pop them off into a new queue/list;
Procedure/function calls;
Local variables;
Parameters;
Return Address;
Volatile environment; **A** register contents State 1 Describe 1

2

(d) List of elements inserted into tree;
To allow rapid/fast searching of the data;
To output sorted/ordered data;

2

[8]

Q9.

(a) It calls itself / is defined in terms of itself / contains within its body a reference to itself;

Ensure 'it' refers to procedure, if meaning program or object no mark

1

(b) The current state of the machine is saved/preserved;
So can return correctly (to previous invocation/call of **Process)**;
OR
Return address / procedure parameter / status register / other register values / local variables must be saved/preserved;
So can return correctly to "*correctly' can be implied* (previous invocation of Process);

2

(c) Printed Output:
1; 3; 5, Bird; Bremner; 4, Fortune, Jones; 2, Smith;

Mark from left and stop marking when error encountered
Ignore punctuation.

6

(d) (in-order) traversal of a tree; **A** printing of tree (elements in order)
I wrong order

1

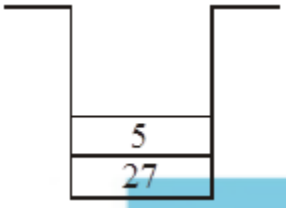
[10]

Q10.

(a) Last In First Out;

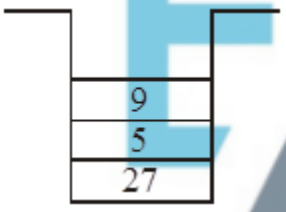
1

(b) (i)



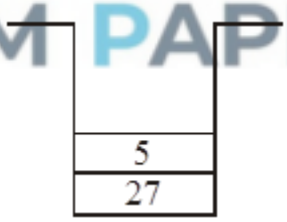
1

(ii)



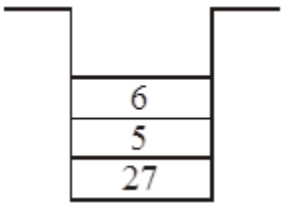
1

(iii)



1

(iv)



1

(c) To reverse elements/ pass parameters/ store volatile environment;
A store return address

1

[6]

Q11.

(a) It calls itself / is defined in terms of itself / is re-entrant / contains within its body a reference to itself;

Ensure 'it' refers to procedure, if meaning program or object no mark

1

(b) The current state of the machine must be saved/preserved so can return correctly to previous invocation of B;
OR
Return address / procedure parameter / status register / other register values / local variables must be saved/preserved so can return correctly to previous invocation of B**)**;

1

(c)

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

;
;
;

Printed Output: 1 1 0 1 0 1;;;

1 mark for each correct pair of bits
Mark from left and stop marking when error encountered ignore punctuation. if more than 6 bits give a max of 2 marks

6

(d) Conversion (of a denary number) into binary;

1

[9]

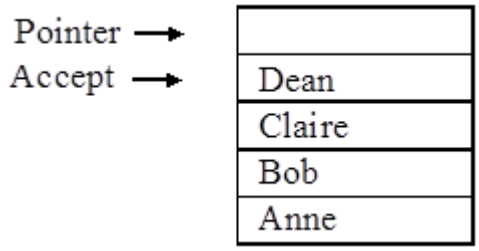
Q12.

Queue is FIFO ; (1)
Stack is LIFO; (1)
Given that:
Process of taking elements from queue to stack(1)
Process of popping stack(1)

[4]

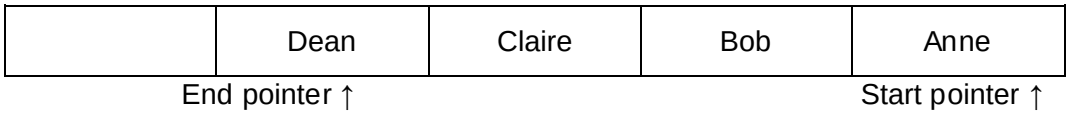
Q13.

(a)



2

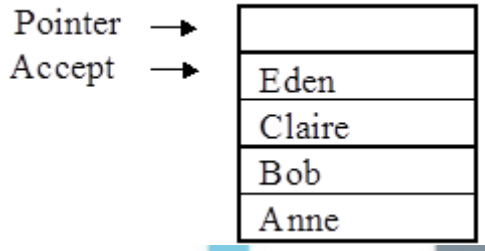
Dean accessed first
Pointer / arrow



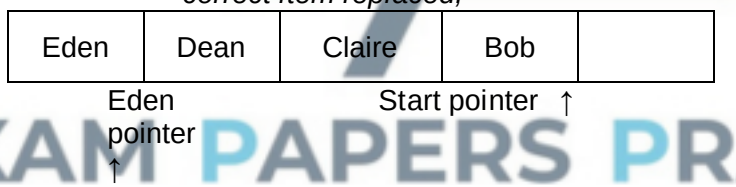
Anne accessed first;
Named pointers correct;

2

(b)



correct item replaced;



Correct item retrieval & input;
Correct moving of pointers

3

(c) In a linear queue data is static, so queue 'moves' through storage/
In a FIFO structure storage locations are only used once;(1)
In a circular queue, the locations will be re-used;(1)
Thus a circular queue has a more efficient use of memory;(1)

1 mark for each of 2 points

2

[9]

Q14.

(a) Tail(Ports) - [Barcelona, Athens, Alexandria, Tunis, Lisbon]

square brackets needed

1

Head(Tail(Tail(Ports))) – Athens (2)

[Athens] (1)

2

Empty(T(T(T(T(T(T(Ports)))))))) – True (2)

True [ ] (1)

[True] (0)

2

(b) **Recursively defined**
A definition which is defined in terms of itself/contains within its body a reference to itself/calls itself ;
A re-entrant; (In specimen papers 2001/2, but refers specifically to a procedure)

1

(c) **Stack necessary**
The state of the machine/contents of appropriate registers/ return address // saved each time the procedure is called (1) and retrieved in reverse order from the stack as control is progressively returned (1)
OR
Different value of parameters /local variables (1) must be available each time procedure is called (1)
OR
P must be re-entrant (In specimen papers 2001/2)(2)

2

(d) Lisbon first (1)
Southampton last(1)
All 6 in order (1)
No punctuation (1)

i.e. Lisbon Tunis Alexandria Athens Barcelona Southampton;

4

(e)

1	Southampton	5	
2	Barcelona	6	Head Pointer
3	Athens	2	4
4	Alexandria	3	
5	Tunis	0 - terminator	
6	Lisbon	1	

2

[14]

Q15.

Repeat
S → Q
Until Q empty
Queue emptied to a stack
Elements taken from front of queue and placed / pushed on stack

2 marks

Repeat
Q → S
Until S empty
Stack emptied to a queue
Elements popped/taken from top of stack placed in queue

2 marks

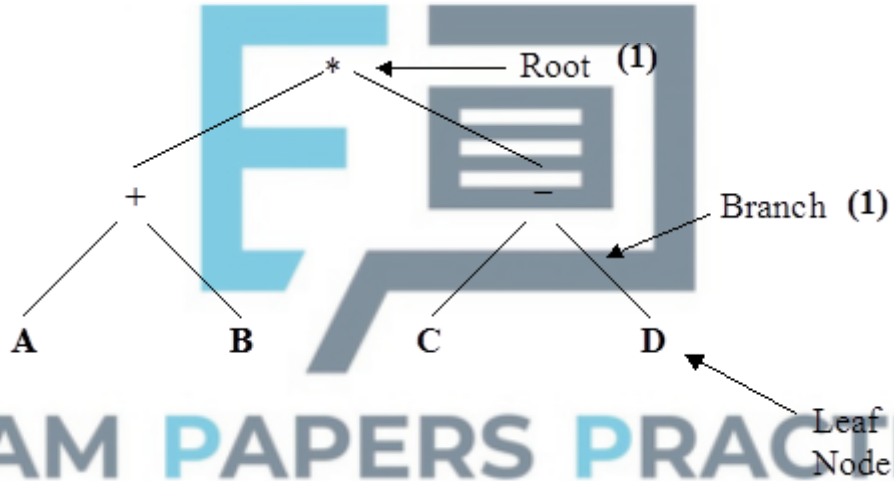
Or suitable diagram

1 mark

[4]

Q16.

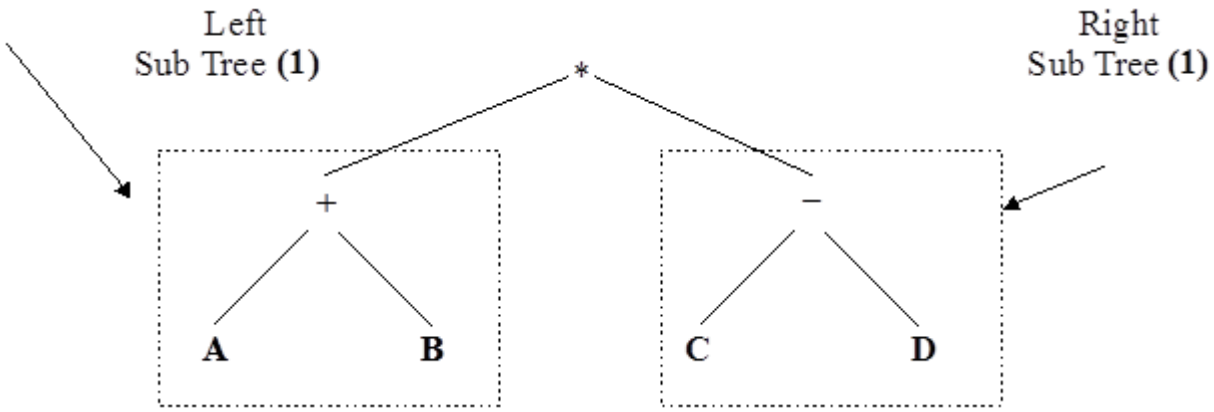
(a)



Labelling must clearly indicate term

3

(b)



Must clearly indicate subsets

2

(c) A procedure which is defined in terms of/ calls itself /re-entrant

1

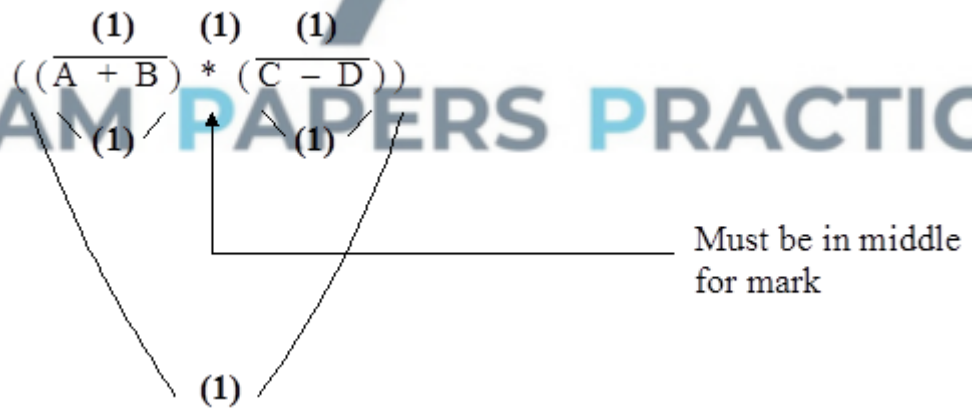
(d) State of machine/return address/parameter (1) needs to be stored/held (1) to enable a previous execution of T to be resumed (1)

Or
So that each call to T(1) can pass(1) a new value of the parameter(1)

3

(e)

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	**tree**('B',empty,empty),	(1)
5	**tree**('-', **tree**('C',empty,empty),**tree**('D',empty,empty))	(1)
6	**tree**('C',empty,empty),	(1)
7	**tree**('D',empty,empty),	(1)



10

(f) In-order traversal

1

[20]

Q1.

This question was about reverse Polish notation (RPN) and stacks. It required students to combine their knowledge of RPN and stacks. A number of clearly-written accurate explanations were seen. Some answers were about converting infix to RPN instead of evaluating an RPN expression; another common error was to use multiple stacks even though the question specified that just one stack should be used. Some students did not fully understand how a stack operates and wrote about accessing items that were not at the top of the stack.

Q2.

Most students had a general understanding of what a stack was but answers were often about how stacks work rather than about the use of stacks in the context given in the question. A common misconception was that the top pointer could be moved down to point to items lower down in the stack without popping data from the top of the stack.

Q3.

Part (a): Two thirds of students were able to identify one property that a graph must have to be a tree. A small number confused a tree with a rooted tree and made assertions such as that a tree must have a root, which is incorrect.

Part (b): This question part tested students' understanding of the method being used to represent a maze as a graph. The majority of students correctly identified a feature of the maze that would stop its graph being a tree. The most commonly seen correct response identified that there could be a loop in the maze. Other possibilities included that part of the maze could be inaccessible or that part of the maze might only be traversable in one direction. Some students failed to achieve the mark because they re-answered part (a), discussing a feature of a graph that would stop it being a tree, rather than a feature of a maze.

Part (c): Students were asked to represent the graph of the maze as an adjacency matrix. Three quarters of students scored both marks for this question part. Responses where symbols other than 0s and 1s were used in the matrix were accepted, as long as they could be viewed as an accurate representation of the graph.

Part (d)(i): The vast majority of students were able to identify that a recursive routine would call itself. A small number asserted that a recursive routine would repeat itself, which was not considered to be enough for a mark as this could equally have been a description of iteration.

Part (d)(ii): Most students scored some marks for this question part, but less than a fifth achieved both. The most widely understood point was that the data would need to be removed from the stack in the reverse of the order that it was put onto it so that the recursion could be unwound. Less well understood was the types of data that would be stored, such as return addresses and local variables.

Part (e): Most students achieved some marks on this question part and around a quarter achieved all five for a fully complete trace. The most commonly made mistake was to update, incorrectly, the Completely Explored array as the recursive calls were made, as opposed to when the recursion unwound.

Q4.

Part (a): This question part was very well answered with the majority of candidates getting both marks. The only common mistake was to miss out the brackets in the expression that should be (12+19)*8.

Part (b): As with part (a), this question part was also well answered. The most common correct response was that brackets are not required. It would have been nice to see some more detailed explanations of this point, rather than just a brief statement of it. A common incorrect answer was that RPN was easier for a computer to understand. The word "understand" is not appropriate in this context.

Part (c): Responses to this question part were excellent, with relatively few errors made. The majority of candidates got full marks which is unusual for a question involving a trace table. The only two recurring mistakes were to pop the numbers off the stack in the wrong order, resulting in the transposition of the values in Op1 and Op2 and forgetting to push 50 back onto the stack at the very end.

Part (d): This question was well answered, with most candidates getting some marks and a significant number more than half marks. The most common mistake was to increment the `TopOfStackPointer` in the wrong place – either before the If construct which tested for the stack full condition or after the value in `ANumber` was stored into the `StackArray`. Some candidates implemented solutions that used a loop to find the first empty position in the array to insert the number into. These were awarded credit if they would have worked, but many failed to test properly for the stack being full. It is important that candidates use the correct variable names when they are given on the question paper.

Q5.

(a) The majority of candidates were able to describe a stack structure as a 'first in last out' or 'last in first out' operation.

(b) The weaker answers seen here moved values to a different memory location once additions and deletions occurred, or used location 605 as the first available and so qualified for a maximum of two (only) 'follow through' marks.

(c) Many candidates were clear about the basic operation which was taking place but then their communication skills let them down in the descriptions required for (ii) and (iii). For (ii) the answer looked for was the idea that items leave the stack one after the other. For (iii) a description was required for the principle of operation of a queue.

Q6.

Candidates generally scored well on this question. Recursively-defined was well understood although many candidates were unable to describe the use of the stack well enough. It was pleasing to see the majority of candidates obtaining most of the marks on part (c). Candidates often failed to obtain the mark for part (d) due to inadequate descriptions. Although many candidates provided a situation where the algorithm will fail, fewer were able to suggest a suitable modification. Once again this was often due to an inability to express themselves well. A wide range of answers were supplied for part (g) but a substantial number of correct responses were given.

Q7.

Although a short question, it proved difficult for most candidates. Many missed the point

that both part (a) and part (b) were about the *implementation* of a stack, and in part (b) gave answers that were about applications that were suitable for a linked list or an array. However, we can note one particularly lucid answer to part (a)(i): “This is a static data structure with a finite pre-declared capacity.”

Q8.

(a) Many candidates scored very well on this question. The terms FIFO and LIFO were very well understood.

(b) Again, a high scoring part of the question. Most candidates managed to place the ticks in the correct boxes.

(c) Quite a few candidates were able to state a suitable example of the use of a stack but few were able to describe it satisfactorily. It should be stressed that candidates should give computing examples. A significant number of answers referred to stacks of plates, books etc.

(d) Once again, many candidates obtained a mark for stating a suitable example but few were able to describe it satisfactorily.

Q9.

Part (a) and (b) have been asked many times before. However, many candidates were unable to explain how a stack is used in the execution of a recursive procedure. Correct responses included that the return address (held in the program counter) and other register values are saved so control can return correctly to the previous invocation of the procedure. Many candidates gained full marks for stating, correctly, the printed output after dry-running the procedure. However, many others could not even get the first few printed items in the correct order.

Many candidates correctly spotted that the procedure described an in-order traversal of a tree.

Q10.

(a) Many candidates scored very well on this question. The term LIFO was very well understood.

(b) Many candidates managed to perform the pushes and pops correctly. Some candidates failed to perform the operations in sequence, taking each part as a new problem. It is essential that candidates read the question carefully.

(c) Few candidates were able to give a suitable example of the use of a stack.

Q11.

Those candidates who clearly understood recursion scored high marks in this question, but a worrying number of candidates could not explain that a procedure is recursively defined when it is defined in terms of itself. A stack is needed so that register values such as return address and parameter values can be saved and can be returned to in the correct order. Most candidates managed to complete the trace table correctly but many did not give the correct printed output. Many candidates did not provide the correct number of digits, or in reverse order. A large majority wrongly thought that procedure B described a binary search. Interestingly, some of the candidates who got the printed output wrong still stated the correct purpose of the procedure: converting the denary number provided as parameter into binary.

Q12.

Many candidates failed to score as highly as they should have through their lack of ability 'to organise relevant information clearly and coherently'. The basic argument is that the fact that a queue is a 'First In, First Out 'structure, while a stack is a 'Last In, First Out' structure. This means that if the elements of a queue are pushed onto a stack, their order, when pushed off the stack, is reversed. Incidentally, the word is 'queue', not 'que' or 'cue' – mistakes prevalent in candidates' answers.

Q13.

This focused on stacks and queues. Again, marks were lost through lack of care in the diagrams. Two vertical columns of names, with Dean at the top of one and Anne at the top of the other were insufficient to show how this data would be stored in these two different data structures. Full marks were gained by two groups of locations, either vertical or horizontal, with pointers to show the top of the stack and the front and rear of the queue. Most candidates did successfully remove and input the correct items in part (b). In part (c), many seemed to think that pointers were only used in circular queues, and that queues were usually implemented in this way so that 'once an item is retrieved and processed it can be placed back into the queue by adding it at the rear'. Good answers noted that a queue is a 'Last In First Out' structure. Data is removed from the front and added to the rear, so current data in a linear queue will 'move' through memory. In a circular queue, the locations in the front of the queue, which are no longer current, will be re-used. A circular queue thus makes more efficient use of memory.

Q14.

(a) This part asked what result would have been returned by the specified function calls. So Tail (Ports) would have returned [Barcelona, Athens, Alexandra, Tunis, Lisbon] with the brackets being part of the correct answer. Head(Tail(Tail(Ports))) would have returned Athens, without brackets and the answer to the last part was True, not [True] or True [ ].

(b) Most candidates could say what recursively defined was, although some answers were barely sufficient and 'It calls itself' was deemed insufficient.

(c) Explanations of why a stack was necessary to execute procedure P recursively frequently missed the point.

(d) Full marks were gained for this part by an answer of:
Lisbon Tunis Alexandria Barcelona Southampton,
or even of:
LisbonTunisAlexandriaBarcelonaSouthampton,
as no punctuation was printed. A list of:

- Lisbon
- Tunis
- Alexandria
- Barcelona
- Southampton

was also accepted.

(e) Candidates who did not score full marks for 10(e) mainly numbered the ports in alphabetical order, rather than using the pointers to point to the next port in the list, or gave inadmissible or absent end-of-list markers.

Q15.

Many candidates grasped that the queue should be emptied into the stack, element by element, removing from the front of the queue and adding to the top of the stack. Candidates who answered by diagram lost marks if they did not clearly indicate that removal was from the front of the queue and insertion was at the top of the stack. Candidates then went on to state that the stack was emptied into the queue by popping elements in turn from the top of the stack. However, several candidates lost a mark by failing to reference the queue as the destination for the popped elements.

Q16.

This question was answered successfully by the better candidates with many scoring full marks. Surprisingly, several candidates identified an internal node as a leaf node and others could not indicate a branch, clearly. In the latter case, candidates indicated their uncertainty by writing “branch” level with an internal node and omitting to link it to the tree diagram by arrow. The same identification problem arose with labelling the left and right sub-trees. Marks cannot be given when there is doubt in the mind of the examiner as to whether the candidate really knows the correct answer. The left and right sub-trees should be ringed and clearly labelled to eliminate doubt in the mind of the examiner.

Recursion has been examined frequently in recent years. It is therefore pleasing to note an increase in the number of candidates who can define the term accurately and who are able to explain why a stack is needed. Sadly, many candidates still have difficulty dry-running the execution of a recursively-defined procedure successfully. Many candidates completed the table correctly but a significant number made no attempt to show the printed output and many others produced output that was wrong. Many candidates looked at the first parameter of each call and incorrectly used it to generate the printed output. These candidates described the procedure as a *pre-order* tree traversal algorithm, whereas, in fact, it was an *in-order* traversal. Candidates who dry-ran the procedure successfully had little difficulty in stating in-order. Some candidates recognised in the layout of the procedure the structure of an *in-order* traversal. These candidates gained credit for their knowledge even though some did not complete the dry run successfully or at all.