3.3 Reverse Polish Mark Scheme

Mark schemes

Q1.

Mark is for AO2 (apply)

5 3 -

[1]

Q2.

All marks AO2 (apply)

3 4 2 * + 1 –

Mark as follows:

1 mark correct order for values and + and – either side of the 1
1 mark * directly after 4 2

Max 1 if any errors

[2]

Q3.

All marks AO1 (understanding)

Simpler for a machine / computer to evaluate; **A.** easier **R.** to understand

simpler to code algorithm;

Do not need brackets (to show correct order of evaluation/calculation); **A.** RPN expressions cannot be ambiguous as **BOD**

Operators appear in the order required for computation;

No need for order of precedence of operators;

No need to backtrack when evaluating;

Max 2

[2]

Q4.

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]

Q5.

(a)

Reverse Polish Notation	Equivalent Infix Expression
18 9 -	18 - 9 **A.** (18 - 9) **R.** 9 – 18
10 4 - 12 ×	(10 - 4) × 12 **R.** 10 - 4 × 12 **A.** * for ×

1 mark per correct infix expression

2

(b) Simpler/quicker for a machine/computer to evaluate // simpler to code algorithm **A.** Easier as BOD **R.** To understand
Do not need brackets (to show correct order of evaluation/calculation);
N.E. Does not use brackets
T.O. No brackets so less storage space used
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

[3]

Q6.

(a) **All marks AO2 (apply)**

```
3 * 4
```

1

(b) **All marks AO2 (apply)**

```
(12 + 8) * 4;
```

1

(c) **Mark for AO1 (understanding)**

1 mark: Simpler / easier for a machine / computer to evaluate / / simpler / easier to code algorithm
R Simpler / easier 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 Benefit Of Doubt (BOD)

1

[3]

Q7.

(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]

Examiner reports

Q1.

This question was about reverse Polish notation (RPN) and stacks. Almost all students were able to convert the simple infix expression into RPN.

Q2.

This question was about reverse Polish notation (RPN) and stacks. Less than a fifth were able to get any marks for the more complex conversion

Q3.

This question was about reverse Polish notation (RPN) and stacks. The majority of students were able to state one advantage of RPN with no need for brackets being by far the most common correct answer.

Q4.

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.

Q5.

This question was about Reverse Polish Notation (RPN). The vast majority of candidates were able to convert both of the expressions from RPN to infix notation for part (a).

Part (b) asked candidates to explain an advantage of RPN. The majority of candidates were able to do this, although some responses were rather superficial. It is true that RPN does not require the use of brackets in expressions, but few candidates went on to explain that this was the case because the order of evaluation was determined entirely by the order in which the operators appeared. Some candidates revealed a lack of understanding of why the lack of the use of brackets was significant, suggesting that it would save memory. Candidates need to be careful to avoid using human-oriented terms such as “understand” and “easier” in the context of answers relating to computer programs. Responses such as “A computer can understand it more easily” were not markworthy.

Q7.

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.