AQA Computer Science A-Level

4.3.3 Reverse Polish

Past Paper Mark Scheme

June 2011 Comp 3

5	(a)		<table><tr><th>Reverse Polish Notation</th><th>Equivalent Infix Expression</th></tr><tr><td>45 6 +</td><td>45 + 6 R 6 + 45</td></tr><tr><td>12 19 + 8 *</td><td>(12 + 19) * 8 R 12+19*8, (19+12)*8 A x for *</td></tr></table> **1 mark** per correct expression **A** extra brackets around complete expressions	**2**

5	(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**

Table 1

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.

5	(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

5	(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**

Spec Qs Paper 1

05	1	**All marks AO2 (apply)** `3 * 4`	1

05	2	**All marks AO2 (apply)** `(12 + 8) * 4;`	1

05	3	**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;	1
		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)	