

THEME B: COMPUTATIONAL THINKING · B2.2

Data structures — Mark scheme

34 marks · covers B2.2.1–B2.2.4

How to mark. Each creditworthy point is on its own line and ends with a semicolon (;). A line beginning with ... is an *expansion* of the line above it, so the pair is worth the marks shown. A slash (/) separates acceptable alternative wordings. Words in (brackets) are not needed for the mark. Mark positively: credit what the student has got right rather than penalising what is missing, and accept any valid alternative not listed here. Do not award more than the maximum for a part. Where a later part uses a wrong answer from an earlier part correctly, award follow-through marks (FT). HL marks parts that assess higher level only content.

Q	ANSWER	MARKS
1(a) B2.2.1	Award [4] max Award [1] per point up to [4] max. Both types must be addressed and the contrast made clear. A static structure has its size fixed when it is declared, whereas a dynamic structure can grow and shrink while the program runs; Memory for a static structure is allocated once, as a single contiguous block, whereas a dynamic structure allocates memory as it is needed; A static structure cannot be resized: to store more items a new, larger structure must be created and the contents copied across; A dynamic structure resizes itself automatically when items are added or removed; A static structure wastes memory if fewer items are stored than the size declared, and fails if more are needed; A dynamic structure uses extra memory per item for its bookkeeping, and resizing costs time when it happens;	4
1(b) B2.2.1	Award [1] max Any one, [1] max. Access to any element is fast, because the position of each item can be calculated directly; It uses less memory per item, with no overhead for pointers or capacity tracking; The size is known in advance, so behaviour is predictable and there is no cost from resizing;	1
2(a) B2.2.2	Award [4] max Award [1] for initialising a variable to hold the highest value; [1] for traversing every element; [1] for a correct comparison and update; [1] for printing the result. Accept either language. JAVA <pre>int highest = rainfall[0]; for (int i = 1; i < rainfall.length; i++) { if (rainfall[i] > highest) highest = rainfall[i]; } System.out.println(highest);</pre> PYTHON <pre>highest = rainfall[0] for value in rainfall:</pre>	4

Q	ANSWER	MARKS
	<pre>if value > highest: highest = value print(highest)</pre> Both print 21. Accept initialising <code>highest</code> to a very small number instead of the first element. Accept Python's built-in <code>max(rainfall)</code> for [2] max only, since the question asks the candidate to construct the search.	
2(b) B2.2.2	Award [2] max Award [1] for the correct method and [1] for correct syntax on a dynamic structure. JAVA <pre>rainfallList.add(8); // an ArrayList<Integer></pre> PYTHON <pre>rainfall.append(8)</pre> A fixed-size Java array cannot be appended to, so the Java answer must use an <code>ArrayList</code>. Do not award marks for an answer that assigns to an index beyond the end of a static array.	2
3(a) B2.2.2	Award [1] max JAVA <pre>marks[3][1]</pre> PYTHON <pre>marks[3][1]</pre> Indexing starts at 0, so the fourth student is row 3 and the second test is column 1. Do not accept <code>marks[4][2]</code>.	1
3(b) B2.2.2	Award [5] max Award [1] for a counter initialised to 0; [1] for an outer loop over the students; [1] for an inner loop over the tests; [1] for a correct comparison against 40; [1] for incrementing the counter and printing the total. Accept either language. JAVA <pre>int below = 0; for (int row = 0; row < 5; row++) { for (int col = 0; col < 4; col++) { if (marks[row][col] < 40) below = below + 1; } } System.out.println(below);</pre> PYTHON <pre>below = 0 for row in range(5): for col in range(4): if marks[row][col] < 40:</pre>	5

Q	ANSWER	MARKS
	<pre> below = below + 1 print(below) </pre> Accept Python's for row in marks: for mark in row: form. Do not award the comparison mark for ≤ 40, which would include a mark of exactly 40. Award [4] max if the counter is printed inside the loop rather than after it.	
4(a) B2.2.3	Award [3] max Award [1] per point up to [3] max. The contrast with a queue must be made. A stack is last in, first out, so the item removed is always the most recently added one; The page the user wants when pressing back is the one they visited most recently; ... (so) popping the stack returns exactly the right page, and repeated presses walk back through the history in reverse order; A queue is first in, first out, so it would return the <i>first</i> page of the session instead; ... (which) is not the behaviour a back button needs;	3
4(b) B2.2.3	Award [2] max Award [1] for both correct pages and [1] for the correct order (B on top). B (top), then A; A is pushed when the user leaves it for B, and B is pushed when the user leaves it for C. C is the current page and is not on the stack. Award [1] for 'A and B' with the order wrong or unstated.	2
4(c) B2.2.3	Award [1] max peek();	1
5(a) B2.2.4	Award [3] max Award [1] per point up to [3] max. The contrast with a stack must be made. A queue is first in, first out, so jobs are printed in the order they were sent; This is fair: a user who submitted first does not wait behind someone who submitted later; A stack is last in, first out, so the most recent job would print first; ... (and) an early job could sit at the bottom indefinitely while new jobs keep arriving, which is starvation;	3
5(b) B2.2.4	Award [1] max Report;	1
5(c) B2.2.4	Award [2] max Award [1] for what front() returns and [1] for the difference. front() returns the item at the head of the queue, which here is Report; ... (but) it leaves the queue unchanged, whereas dequeue() removes that item as well as returning it;	2
6(a) B2.2.3 B2.2.4	Award [3] max Award [1] per point up to [3] max. push and pop always act on one end of the structure, so the position is already known; The work done does not depend on how many items are stored, so the operation takes constant time, $O(1)$; Searching a list for a particular value may have to examine every item until it is found; ... (so) the time taken grows with the number of items, $O(n)$ for an unsorted list; The trade-off is that a stack only gives access to the top item, so it cannot be searched efficiently;	3

Q	ANSWER	MARKS
6(b) B2.2.3	Award [2] max Award [1] for the correct structure and [1] for the justification. No mark for the structure alone. A stack; Undo must reverse the most recent action first, which is exactly last in, first out: each action is pushed as it happens and popped when the user undoes it;	2

Total for this paper: 34 marks