Chhetri Academy

chhetriacademy.com

GCSE COMPUTER SCIENCE

2.2

OCR J277 | Specification point 2.2.3

Additional programming techniques

Untiered

Answers and examiner comments

2	2018-2023	Untiered
question groups	available papers	tier

Using this booklet

1. Answer numbers match the companion question booklet.
2. Follow the original question number on marking tables.
3. Examiner comments are included where available.

Name _____ **Date** _____

Some entries use full source marking material in an appendix. Any unavailable mark schemes are identified beside the relevant answer.

Question			Answer	Mark	Guidance

4	(a)	(i)	1 mark per filled gap, max 3
			
			`01 function librarycode(title, year)`
			
			`02  parta = title.substring(0, 3)`
			
			`03  partb = year.substring(2, 2)`
			
			`04  return parta.upper + partb`
			
			`05 endfunction`
				3	Ignore capitalisation.
					
					Allow **librarycode =** for 3rd mark – this is an equivalent in some languages for returning a value (eg. Visual Basic).

Question			Answer	Mark	Guidance

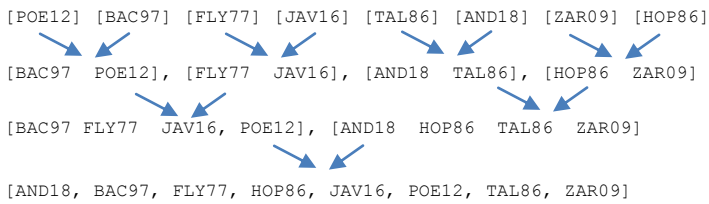
4	(a)	(ii)	1 mark per bullet, max 6
			
			• Input title and year from user
			• Open bookcodes.txt
			• Call the librarycode() function…
			• … with the two parameters that match input values
			• … write out **code obtained** to the text file
			• Close text file
				6	Example algorithm
					
					`title = input("enter title")`
					`year = input ("enter year")`
					`code = librarycode(title, year)`
					`myFile = openWrite("bookcodes.txt")`
					`myFile.writeLine(code)`
					`MyFile.close()`
					
					Note, pseudocode shown above is an example – candidates may answer very differently, but award marks if intention can be seen.
					
					Bullet points 3,4 and 5 could be done in one line:
					`myFile.writeLine(librarycode(title, year))`
					
					Do not award bullet point 3 if candidate is defining the function rather than calling it.
					
					Allow bullet point 2 (opening text file) if correctly referred to during write operation.
					
					Bullet point 3 must include brackets () to signify it is the function being called or indication that is being called.

4	(b)	(i)	1 mark per bullet, max 2. • Function **returns** a value • Procedure **does not return** a value	2	Allow "does not" for second mark if intention is clear (ie if it is obvious that the "not" refers to not returning a value). Allow discussion of how returned value in a function can be used (e.g. to assign to a variable or to use this returned value in some way).

J276/02 Mark Scheme June 2018

Question			Answer	Mark	Guidance
4	(b)	(ii)	1 mark per bullet, max 4. Mark in pairs. e.g. • Breaks down / decomposes / modularises the problem / program // structures the program • …making it easier to design/create/test • …each subroutine can be tested separately • Reuse code (in different programs) • …quicker to develop (new) programs • …build on existing work / use of a library of subroutines • Avoid repetition of code (in the same program) • …makes program shorter / smaller • … subprogram called instead of copying/pasting. • … quicker to develop (new) programs • Easier to maintain …as code is easier to understand/read • …as code is shorter • Easier to debug • …as code is shorter • …same bugs will not have been copied to other areas of the program. • Work can be split up in a team • ...to suit developers' skill set • …to work on different subprogram at the same time / develop separately • Allows for abstraction / removes complexity • …subprograms can be used by programmers who do not need to understand how they work.	4	Maximum of two benefits with expanstions to be marked as per question. Allow other sensible expansions. Allow expansions which cross over to other benefits (e.g. breaks down the problem / to make it easier to maintain). Allow "can be called multiple times" Allow "file size is smaller". Do not allow "more efficient" without further explanation.

J276/02 Mark Scheme June 2018

Question			Answer	Mark	Guidance
4	(c)	(i)	1 mark per bullet, max 4. • List split into individual elements (may be done over several steps or just as a starting point) • **Merge** individual elements into **sorted** lists of size 2 • **Merge** lists of size 2 into **sorted** lists of size 4 • **Merge** lists of size 4 into final **sorted** list.	4	Candidates **can** describe how the merge sort would work rather than showing output values at each stage. Ignore intermediate steps. **Do not** give final mark for simply showing the list sorted. Must have the (correct) idea of where it being merged from previous lists. Candidates' answers describing / showing other sorting algorithms (e.g. bubble sort, insertion sort) are worth 0 marks. 
4	(c)	(ii)	1 mark per bullet, max 2. • Faster/quicker (to sort)… • …for large lists // for lists that are **more unordered** • Has a consistent running time (for a lists of same length)… • …doesn't depend on how ordered original list is	2	Accept (correct) reference to big O notation for 2nd mark on either mark point although this is beyond scope of GCSE specification. Allow "more efficient" for BOD on first bullet point.

<table>
<tr><th colspan="3">Question</th><th>Answer</th><th>Mark</th><th>Guidance</th></tr>
<tr><td>1</td><td>(a)</td><td></td><td>1 mark per row

<table>
<tr><th>Statement</th><th>Low-level</th><th>High-level</th></tr>
<tr><td>The same language can be used on computers that use different hardware</td><td></td><td>✓</td></tr>
<tr><td>It allows the user to directly manipulate memory</td><td>✓</td><td></td></tr>
<tr><td>It allows the user to write English-like words</td><td></td><td>✓</td></tr>
<tr><td>It always needs to be translated into object code or machine code</td><td></td><td>✓</td></tr>
</table></td><td>4
(AO1 1b)</td><td>No mark if more than 1 tick for that row.

Allow other indications of choice (e.g. cross) as long as clear.</td></tr>
<tr><td>1</td><td>(b)</td><td></td><td><u>total</u> = num1 + num2</td><td>1
(AO3 2b)</td><td>Allow other logically valid responses that result in total storing the correct value. Accept other suitable assignment operators (e.g. ←)

e.g.
total = sum(num1, num2)

total = num2 + num1

x = num1 + num2
total = x

Ignore any values given to the variable. Ignore capitalisation and minor misspelling. Ignore superfluous code that does not affect outcome.</td></tr>
</table>

Question			Answer	Mark	Guidance
1	(c)	(i)	`print(12 ^  2)`	1 (AO2 1a)	Accept ** or other sensible operator that indicates raising to a power. If pseudocode operator given, must be a single word/symbol (e.g. `pow`), not containing spaces.
1	(c)	(ii)	`if number MOD 2 == 0 then`	1 (AO2 1a)	Accept **%** or other sensible operator that indicates modulus If pseudocode operator given, must be a single word/symbol (e.g. `modulo`), not containing spaces.
1	(c)	(iii)	`difference = measurement1 - measurement2`	1 (AO2 1a)	Accept other sensible operator that indicates subtraction. If a pseudocode operator given, must be a single word/symbol (e.g. `minus`), not containing spaces.

<table>
<tr><th colspan="3">Question</th><th>Answer</th><th>Mark</th><th>Guidance</th></tr>
<tr><td>1</td><td>(d)</td><td></td><td>1 mark each:
• Start is set to 3 on line 01 and 3 is output on line 03.
• 2, 1 and 0 are output on next 3 iterations with start updated to 2, 1, 0, -1 on correct line numbers.
• Finished is output on line 06

<table>
<tr><th>Line</th><th>start</th><th>Output</th></tr>
<tr><td>01</td><td>3</td><td></td></tr>
<tr><td>03</td><td></td><td>3</td></tr>
<tr><td>04</td><td>2</td><td></td></tr>
<tr><td>03</td><td></td><td>2</td></tr>
<tr><td>04</td><td>1</td><td></td></tr>
<tr><td>03</td><td></td><td>1</td></tr>
<tr><td>04</td><td>0</td><td></td></tr>
<tr><td>03</td><td></td><td>0</td></tr>
<tr><td>04</td><td>-1</td><td></td></tr>
<tr><td>06</td><td></td><td>Finished</td></tr>
</table>
BP1
BP2
BP3</td><td>3
(AO3 2c)</td><td>Ignore lines 02 and 05 in answer unless these change or output any values.

Candidate may repeat start value when unchanged, this is acceptable.

Penalise incorrect or missing line numbers or additional output once only then FT. This includes where variable change and output appear on the same line.

-1 must not be output for BP2

Penalise missing or incorrect output once only for BP1 and FT for missing or incorrect output for BP2.

Finished may be with or without quotes. Ignore case or minor spelling error.

Max 2 marks if any incorrect output or changes to <code>start</code>.

Do not accept calculated values of start (e.g. <code>3-1</code>)</td></tr>
</table>

Question 1 (a)

1 (a) The table contains four statements about programming languages.

Tick (✓) **one** box in each row to identify whether each statement describes a low-level programming language or a high-level programming language.

Statement	Low-level	High-level
The same language can be used on computers that use different hardware		
It allows the user to directly manipulate memory		
It allows the user to write English-like words		
It always needs to be translated into object code or machine code		

[4]

This question was answered well by many candidates. The strongest responses showed a good understanding of the difference between low-level and high-level languages. Incorrect responses tended to be on the first row related to portability. A high-level language such as Python is portable, with translators available for many different types of processors. A low-level language is specific to one type of processor.

Question 1 (b)

(b) The variables `num1` and `num2` store integers.

Write pseudocode to add the integers stored in `num1` and `num2`. Store the result in a variable with the identifier `total`

.....

..... **[1]**

Candidates appear to be getting more confident at answering simple programming/pseudocode questions such as this. The majority of responses included code written to produce the required outcome. The use of multiple steps was allowed.

Question 1 (d)

(d) Read the following pseudocode algorithm:

```
01  start = 3
02  do
03    print(start)
04    start = start - 1
05  until start == -1
06  print("Finished")
```

Complete the following trace table for the given algorithm.

Line number	start	Output

[3]

This question assessed candidates' ability to trace through and understand the steps taken by an algorithm. This also tested their understanding of condition-controlled loops. Many responses were very successful with this and achieved full marks.

Mistakes tended to be with identifying the line number where each change occurred or outputting values that were not actually output (e.g. -1).

Examiners were instructed to only penalise a misunderstanding once. Where (for example) line numbers were incorrect, this would still have allowed 2 out of 3 marks to be achieved.