GCST

2.3 PRODUCING ROBUST PROGRAMS

TOPIC WISE EXAM QUESTIONS

ANSWERS

GCSE OCR

GCST

2023

5	(a)	(i)	1 mark each to max 2 • Check the program works (as intended) • meets user requirements. • gives the correct **output** / **result** • Find / detect / check for errors / bugs • Check the program does not crash // is robust // executes / runs • To try and break the program // destructive testing • Test for / improve usability / user experience / performance // check user feedback is **suitable** • Allow any errors to be fixed // make changes / improvements as a result of testing • Ensure no problems / issues fixed **when released**. • Defensive design considerations / anticipating misuse / so cannot be misused	**2** (AO1 1b)	Allow answers that explain what would happen if not tested (e.g. "there might be bugs")
5	(a)	(ii)	1 mark for name, 1 mark for matching description e.g. • Final / terminal testing… • … Completed at the end of development / before release. • … to test the product as a whole. • Iterative / incremental testing… • …completed during development. • …after each module is completed. • … test module in isolation • Normal testing… • …test using data that should be accepted // • …test that is expected to work / pass • Boundary / Extreme testing… • …test using data that is on the edge of being acceptable / unacceptable. • …test highest / lowest value • Invalid / Erroneous testing… • …test using data that should be **rejected** / is not acceptable / causes an error	**2** (1 AO1 1a), (1 AO2 1b)	Allow other sensible descriptive names for testing. Description must match test type. Must be a description and not just an example, but example may support description. Do not accept descriptions that simply repeat type of test without further clarification (e.g. "boundary, testing the boundary"). Allow : • Black box testing… • …testing **without** access / knowledge of a system's workings. • White box testing… • …testing **with** access / knowledge of system's workings. Allow other sensible / valid types of testing. Do not accept examples of validation (e.g. type test, range check) "data that is not expected" is NE for invalid/erroneous unless clarified.

5	(b)		1 mark for method, 1 mark to max 2 for each use e.g. • Range check • … checks **upper/max / lower/min / boundaries** • … make sure the players **answer / input** is between sensible limits (e.g. 20 or less, between 2 and 20 inclusive) // not negative • …by example of program code • Type check • … making sure the data inputted is of the correct **data type** • … make sure **answer / input** is an **integer** (or equivalent e.g. whole number) • Presence check • … making sure a value is inputted / not blank • … reference to **answer / input** • …by example of program code • Length check • … **limit** number of **characters** // check **maximum / minimum string** length • … **answer / input** must be **1 or 2 characters**	6 (4 AO2 1b) , (2 AO1 1a)	Validation must be applied to the rules of the game as given; **do not** accept uses related to input not asked for (e.g. names, passwords, etc). Do not accept uses that simply repeat the name of the check (e.g. "range check, checks a range of numbers") For range check, values must be sensible (e.g. 1 to 50), and allow input of 2 to 20. Do not allow 1 / 10 (answer could be over this). For length check, must be clear that the **string** version of the data input is being checked to award use marks (e.g. characters not digits). Accept alternative names or descriptions (e.g. existence check, boundary check) but name of check must be sensible. Mark each answer as a whole, ignore method/use headings. Do not accept defensive design elements (e.g. input sanitisation, authentication)
			• …by example of program code • Format check • … making sure the data inputted follows a **set pattern** • … checking **answer / input** consists of **only** 1 or 2 numeric digits • …by example of program code • Look up / table check • … making sure the data inputted is one from an **allowed set of values** • … checking that **answer / input** is one of **[2, 3…20]** inclusive • …by example of program code		Examples of program code can be actual code (e.g. `if inp>=2 and inp<=20`) or identification of technique (e.g. "use IF statement / Case statement to limit values to between 1 and 20"). Do not accept code just showing casting.

GCST

2022

4	(a)		Any two bullet points for one mark each: • Add comments • Name variables sensibly • **Put into** subroutine / procedure / function • Use loop / iteration	2 (AO2 1b)	Do not accept indentation (no code to sensibly indent in this example) "Use a subroutine" is not enough. Must be clear that existing code will be put into a new subroutine.
5	(b)	(i)	• Checks that **both** `firstname` and `surname` are not empty… • Checks that `room` is either "basic" or "premium"… • Checks that `nights` is between 1 and 5 (inclusive)… • …Outputs "NOT ALLOWED" (or equivalent) if <u>any</u> of the 3 checks are invalid (must check all three) • …Outputs "ALLOWED" (or equivalent) <u>only</u> if all three checks are valid (must check all three) *Note : output marks are given for if <u>entire system</u> produces the correct output. For example, If a user enters a valid name and room but an invalid number of nights, the system should say "NOT ALLOWED" (or equivalent). If this works and produces the correct response no matter which input is invalid, BP4 should be given.* *The same process holds for the valid output – if (and only if) three valid inputs results in an output saying "ALLOWED" (or equivalent), BP5 should be given. Do not give this if ALLOWED is printed when (for example) two inputs are valid and one is invalid.* *For any output marks to be given, a sensible attempt must have been made at all three checks. These may not be completely correct (and may have been penalised in BPs 1 to 3) but should be enough to allow the FT marks for output.*	5 (AO3 2a)	Must have some attempt at <u>all three checks</u> to give output mark(s). Check for `nights` must check both upper and lower limits. Iteration can be used as validation if input repeatedly asked for until valid answer given. <u>Do not accept</u> logically incorrect Boolean conditions such as `if firstname or surname == ""` Do **not** accept ≥ or ≤ for >=, <=. Ignore capitalisation e.g. `valid = True` `if firstname == "" or surname == "" then` `  valid = False` `end if` `if room != "basic" and room != "premium" then` `  valid = False` `endif` `if nights < 1 or nights > 5 then` `  valid = False` `endif` `if valid then` `  print("ALLOWED")` `else` `  print("NOT ALLOWED")` `endif` BP1 to 3 can check for valid or invalid inputs. . Pay particular attention to use of AND / OR. Only give marks for output if these work together correctly. Example above shows checking for **invalid** data. Checks for **valid** data equally acceptable Examples shown below : • `if firstname != "" and surname != ""` • `if room == "basic" or room == "premium"` • `if nights >= 1 and nights <= 5`
5	(b)	(ii)	• Normal • 1 or 5 *(not 0 or 6 as says allowed)* • Any numeric value except 1 to 5 // any non-numeric input (e.g. "bananas")	3 (AO3 2c)	Allow other descriptions that mean normal (e.g. valid / typical / acceptable) **Test data (number of nights)** | **Type of test** | **Expected output** 2 | Normal | ALLOWED 1 // 5 | Boundary | ALLOWED e.g. 7 | Erroneous/Invalid | NOT ALLOWED

GCST

SAMPLE

7	a		1 mark for naming the example and 1 mark for an example related to that method E.g • Comments/annotation… • …E.g. any relevant example, such as line 4 checks the input is valid • Indentation… • …E.g. indenting within IF statement • Using constants… • …E.g. π	**4 (AO2 1b)**	
8	b	i	• `or` • `>300 // >= 301` • `print`	**3 (AO3 2b)**	**<u>High-level programming language / OCR Exam Reference Language response required</u>** Do not accept pseudocode / natural English. MP2 do not accept 'greater than', must use the HLL syntax > or >= MP3 must be a suitable output command word that could be found in a HLL e.g. `print` (Python), `console.writeline` (VB), `cout` (C++)
	b	ii	• Suitable invalid test data (i.e. > 300, e.g. 350) • "Value accepted" or equivalent	**2 (AO3 2c)**	
8	f		1 mark per bullet • Test data either 0 or less characters, or 20 or more characters • Stating correct output • Test data between 1 and 19 characters (inc) • Stating correct output	**4 (AO3 2c)**	Mark test data first, both must meet different criteria. Then mark output for each.

2021

(c)	(i)	• Parameter values outside index range / larger than 4 / smaller than 0 // -1, 16 is not a valid block	1	Answer must refer to either array or gameboard / grid / block
	(ii)	• Use selection / IF / Switch-Case / range check • …check that **parameters** are >=0 **and** <= 4… • …**Return** error code if invalid // set outcome to error	3	Allow equivalent checks (e.g. <5, between 0 and 4) for BP2 Allow reference to `r` and `c` as parameters. BOD handle error for BP3 (e.g. repeat until valid) Answer must be a description, code by itself is NAQ

GCSE

2020

<table>
<tr>
<td>3</td><td>(a)</td><td>(i)</td>
<td>1 mark per bullet to max 2
e.g.
• Check the program meets the user requirements
• Check the program works (as intended) // detect logic / syntax errors
• Check the program does not crash (under invalid entry) // check error messages are suitable
• …allow these errors to be fixed
• …make sure there are no problems when released
• Any suitable example related to the vending machine e.g. gives correct change</td>
<td>2

AO1
1b(2)</td>
<td>Allow two any suitable examples for two marks

BOD "find errors", "find bugs" for BP2

"fix errors" by itself is one mark (BP4)</td>
</tr>
<tr>
<td>3</td><td>(a)</td><td>(ii)</td>
<td>1 mark per bullet to max 2
• Iterative is during development // repeatedly testing after/while making changes
• Final is when the development is (almost) complete // done after iterative testing</td>
<td>2

AO1
1b(2)</td>
<td>Do not accept just "repeatedly testing" for iterative

BOD "iterative testing tests modules/sections"</td>
</tr>
<tr>
<td>3</td><td>(a)</td><td>(iii)</td>
<td>
<table>
<tr><th>Code entered</th><th>Money inserted</th><th>Expected result</th></tr>
<tr><td></td><td></td><td></td></tr>
<tr><td>C2</td><td></td><td></td></tr>
<tr><td></td><td>£0.49
(or any value less that £0.50)</td><td></td></tr>
<tr><td></td><td></td><td>Invalid Selection
(or any suitable error message)</td></tr>
</table>
</td>
<td>3

AO3
2b(3)</td>
<td>For £0.49 accept any value <£0.50. Must be a specific value, not a description.

Accept any suitable error message for invalid selection</td>
</tr>
</table>

3	d	i	1 mark per bullet to max 2 • Indentation // whitespace • Appropriately named variables / identifiers • Modularisation / use of subroutines	2 AO2 1b(2)	
3	d	ii	• Comments • Use of constants	1 AO2 1b(1)	

GCSE

2019

4	(b)		1 mark per bullet, mark in pairs. Max 2 per point. e.g. • Input sanitisation • …cleaning up input data / removing unwanted data • …by example (e.g. removing special characters / preventing SQL injection) • Validation • …checking whether input data should be allowed / is sensible / follows criteria • …by example (e.g. goals cannot be less than 0) • Verification • … checking whether data has been entered correctly • …by example (e.g. double entry / visual check) • Authentication • …ensuring only allowed / authorised users can gain access • …by example (e.g. usernames /passwords) • Maintainable code • …to allow other programmers to understand the code • …by example(e.g. comments, indentation, meaningful variable names)	4 AO2 1a (2) AO2 1b (2)	Mark first answer only in each section For validation, allow one example of a type of validation (e.g. type check, range check) e.g. question so allow other sensible examples such as audit logging, encryption of data Do not allow "data is correct" as expansion for validation – validation checks that data is sensible or follows rules, NOT that it is correct. Planning for contingencies and anticipating misuse are not examples by themselves, but discussion of these may fit under other points – e.g. input sanitisation, validation.

If you found this useful, drop a follow to help me out!

THANK YOU!

GCST

GCSECOMPUTERSCIENCETUTOR.COM