2.3 Producing Robust Programs: Test Pack

OCR J277 · Calculators not allowed · about 25 minutes

Total Marks

Name: _____ Date: ____ / ____ / _____

Answer ALL questions. Show all your working.

1 A software company is developing a new app. The lead developer tells the team that the app must be 'robust'.

(a) State what is meant by a robust program. **(1)**

(b) Give two examples of situations, other than invalid data entry, that a robust program should be able to handle without crashing. **(2)**

(Total for Question 1 is 3 marks)

---

2 A bank wants to make its online banking login more secure than a password alone.

(a) Identify two authentication methods, other than a password, that the bank could add to its login process. **(2)**

(b) Explain how using two-factor authentication reduces the risk of a criminal accessing a customer's account, even if the criminal has stolen the customer's password. **(2)**

(Total for Question 2 is 4 marks)

© Revision Library | revisionlibrary.co.uk

3 Binary, hexadecimal and denary conversions (no calculator).

(a) Convert the 8-bit binary number 10111010 to hexadecimal. **(2)**

(b) Convert the hexadecimal number 3F to denary. **(2)**

(c) State one reason why programmers often use hexadecimal, rather than binary, to represent values such as colour codes or MAC addresses. **(1)**

(Total for Question 3 is 5 marks)

© Revision Library | revisionlibrary.co.uk

4 Study the following algorithm, written in OCR Exam Reference Language:

```
count = 0
do
mark = int(input("Enter a mark"))
count = count + 1
if mark < 0 OR mark > 100 then
    print("Invalid, try again")
endif
until mark ≥ 0 AND mark ≤ 100
print("Valid mark entered after " + str(count) + " attempt(s)")
```

A user enters the following values, in order, when the program runs: -5, then 150, then 68.

(a) Complete a trace table to show the value of mark, count and any output produced for each iteration of the loop. **(4)**

(b) State the final message that the program outputs. **(1)**

(Total for Question 4 is 5 marks)

---

5 An online shop is developing a website where customers enter their payment card details to make a purchase. Discuss the defensive design techniques the programmers should use, and explain why each technique is important for producing a robust and secure program.

(Total for Question 5 is 6 marks)

© Revision Library | revisionlibrary.co.uk

Mark scheme · 2.3 Producing Robust Programs: Test Pack

Question 1

(a) A program that continues to run / does not crash even when given unexpected or invalid input (oe) **B1**

(a) Answer: A program that keeps running correctly (does not crash) even if the user enters unexpected or invalid data.

(b) First valid example, e.g. a required file is missing **B1**

(b) Second valid example, e.g. a network connection is lost / division by zero / peripheral not connected **B1**

(b) Answer: Any two of: a required file is missing; a network connection is lost; a division by zero is attempted; a peripheral (e.g. a printer) is not connected.

Question 2

(a) First valid method, e.g. biometrics (fingerprint/facial recognition) **B1**

(a) Second valid method, e.g. a one-time PIN/code sent by text or app (two-factor authentication) / security question / physical token **B1**

(a) Answer: Any two of: biometrics (e.g. fingerprint or facial recognition); a one-time code sent by text/app (two-factor authentication); a memorable/security question; a physical security token or card reader.

(b) The criminal would also need the second factor, e.g. a code sent to the customer's own phone or a biometric feature **M1**

(b) The criminal is very unlikely to also have access to this second factor, so access is still denied even with the correct password **A1**

(b) Answer: Even with the correct password, the criminal would also need the second piece of evidence, such as a one-time code sent to the customer's own phone or a fingerprint, which they are very unlikely to have, so access is still blocked.

Question 3

(a) Correct method: splitting into nibbles 1011 and 1010, or converting via denary 186 **M1**

(a) BA cao **A1**

(a) Answer: BA

(b) Correct method: 3x16 and F=15 identified **M1**

(b) 63 cao **A1**

(b) Answer: 63

(c) Hexadecimal is a shorter/more compact way of representing the same value than binary, so it is quicker to read/write and less prone to transcription errors (oe) **B1**

(c) Answer: Hexadecimal represents the same value using far fewer digits than binary, so it is quicker and easier for humans to read, write and check.

Question 4

(a) Iteration 1: mark=-5, count=1, outputs 'Invalid, try again' **B1**

(a) Iteration 2: mark=150, count=2, outputs 'Invalid, try again' **B1**

(a) Iteration 3: mark=68, count=3, no 'Invalid' output produced **B1**

(a) Loop correctly shown to end after iteration 3, since $0 \leq 68 \leq 100$ is true **B1**

(a) Answer: Iteration 1: mark=-5, count=1, outputs 'Invalid, try again'. Iteration 2: mark=150, count=2, outputs 'Invalid, try again'. Iteration 3: mark=68, count=3, condition true, loop ends.

(b) 'Valid mark entered after 3 attempt(s)' cao **B1**

(b) Answer: Valid mark entered after 3 attempt(s)

Question 5

Level 1 Basic. Identifies one or two defensive design techniques with little or no explanation of their purpose. Answer lacks structure or detail. **1-2**

Level 2 Good. Identifies several relevant defensive design techniques with some explanation of why each helps produce a robust or secure program. Some structure evident. **3-4**

Level 3 Excellent. Identifies and explains a range of relevant defensive design techniques in detail, with clear links to why each reduces the risk of crashes or security breaches for this specific scenario (payment data). Answer is well structured and uses correct technical terminology throughout. **5-6**

Indicative content:

- Input validation (e.g. presence, format, length, range and check digit checks) on the card number, expiry date and CVV fields, to reject data that is missing, the wrong length or the wrong pattern before it is processed.
- Input sanitisation, to remove or escape unexpected characters, helping to prevent malicious input such as SQL injection being used to attack the underlying database.
- Authentication (e.g. requiring the customer to log in and/or a one-time code for two-factor authentication), to confirm the person entering the payment details is who they claim to be.
- Maintainability techniques (e.g. clear comments, indentation, meaningful identifier names) make the code easier for other programmers to review and spot flaws in.
- Thorough testing, using normal, boundary and erroneous test data, checks that the validation and sanitisation routines work correctly before the site goes live.
- Overall justification: because card details are highly sensitive, failing to use these techniques could allow invalid data to crash the program or allow fraud/data theft, so defensive design directly reduces both technical failure and security risk.

© Revision Library | revisionlibrary.co.uk