CERTLABZ CERTIFICATION BOARD

Certified Smart Contract Engineer (CSCE)

Ten Sample Questions

These ten questions are written at the difficulty of the live exam. Six carry a figure, a data table, a configuration or a code listing; four are plain scenario questions, as on the live paper. Every one is a scenario a working smart contract engineer would meet, and none of them appears in the live question bank.

The live exam is multistage adaptive: how you score on one stage selects the difficulty of the next, so no two candidates sit the same paper. It is a performance-based exam: each form can carry up to 5 to 12 performance-based questions alongside the multiple-choice questions, which means you build, sequence or complete something rather than only choosing from a list. Answers and full explanations follow each question here; the live exam does not disclose item-level answers.

Question 1

Domain 5: Security and Common Vulnerabilities

CONFIG

Emma Wilson reviews a withdraw function that sends funds before updating state.

```
function withdraw() public {
  uint amount = balances[msg.sender];
  (bool ok, ) = msg.sender.call{value: amount}("");
  balances[msg.sender] = 0;   // state set AFTER send
}
```

Listing 1.1 The withdraw function.

What is the flaw, and the fix?

A. No flaw exists here, because setting the balance to zero after the transfer is a perfectly safe and standard ordering pattern.

B. It is reentrancy; set the balance to zero before the external call, following the checks-effects-interactions ordering.

C. The flaw is high gas cost, so the fix is to remove the balance update and let the caller track their own balance instead.

D. The flaw is the return value, so the fix is to ignore whether the call succeeded and always continue the function anyway.

Correct answer: B
Sending funds before zeroing the balance lets the recipient re-enter withdraw and drain the contract. The fix is checks-effects-interactions: update state before the external call. It is not safe, the issue is not gas, and ignoring the call result makes things worse.

Question 2 — Domain 3: Standards and Patterns — DATA

James Carter matches each need to the token standard family.

Need	Standard family
Interchangeable currency-like units	(to choose)
Unique one-of-a-kind items	(to choose)
Many token types in one contract	(to choose)
Upgrade logic without moving state	(to choose)

Table 2.1 Needs and standards.

Which set of choices is correct?

A. Non-fungible, fungible, proxy pattern and multi-token, so unique items use the fungible standard and currency uses non-fungible.

B. Fungible, non-fungible, multi-token and proxy pattern, matching interchangeable units, unique items, many types, and upgradeable logic.

C. Multi-token, proxy pattern, fungible and non-fungible, applied so the proxy pattern is what represents unique one-of-a-kind items.

D. Proxy pattern, multi-token, non-fungible and fungible, applied so the fungible standard is what enables upgrading logic in place.

Correct answer: B
Interchangeable units use a fungible standard, unique items use a non-fungible standard, many token types in one contract use a multi-token standard, and upgrading logic without moving state uses a proxy pattern. Only option B pairs each need correctly.

Question 3 — Domain 6: Gas, Deployment and Operations — CONFIG

Olivia Davis sees a loop that writes to storage on every iteration.

```
for (uint i = 0; i < items.length; i++) {
  total = total + items[i];   // storage write each pass
}
// total is a storage variable
```

Listing 3.1 The loop.

What is the most effective gas optimization here?

A. Accumulate into a local memory variable inside the loop and write to storage once after the loop finishes running.

B. Add more storage writes inside the loop, because writing more often reduces the total gas the loop consumes overall.

C. Remove the loop bound check, because reading the array length is what makes this loop expensive to run on chain.

D. Convert every number to a string first, since string math is cheaper than integer math on the virtual machine.

Correct answer: A
Storage writes are among the most expensive operations, so accumulating in a local (memory) variable and writing to storage once after the loop cuts gas sharply. Adding writes costs more, the length read is minor, and string math is not cheaper than integer math.

Question 4

Domain 4: Testing and Tooling

CONFIG

Liam Anderson lists the steps to ship a contract safely, shown out of order.

```
A. Deploy to mainnet and verify the source
B. Write unit and fuzz tests
C. Develop the contract
D. Deploy and test on a testnet
```

Listing 4.1 Steps to order.

What is the correct order?

A. A, then C, then B, then D, deploying to mainnet before the contract has been written, tested or tried on a testnet at all.

B. C, then B, then D, then A, developing, then testing, then trying on a testnet, then deploying to mainnet and verifying source.

C. D, then A, then C, then B, going to a testnet and mainnet before the contract is developed or any tests are written for it.

D. B, then A, then D, then C, writing tests, deploying to mainnet, then a testnet, and developing the contract last of all.

Correct answer: B
The safe order is develop the contract, write unit and fuzz tests, deploy and test on a testnet, then deploy to mainnet and verify the source. The other orders deploy to mainnet before the contract exists or is tested, which risks real funds.

Question 5

Domain 2: Contract Development

DATA

Ava Thompson matches each concept to its meaning.

Concept	Meaning
Modifier	(to choose)
Event	(to choose)
storage vs memory	(to choose)
require	(to choose)

Table 5.1 Development concepts.

Which set of meanings is correct?

A. A logged signal for off-chain use, reusable precondition logic, a guard that reverts on false, and persistent versus temporary data.

B. Reusable precondition logic, a logged signal for off-chain use, persistent versus temporary data, and a guard that reverts on false.

C. Persistent versus temporary data, a guard that reverts on false, reusable precondition logic, and a logged signal for off-chain use.

D. A guard that reverts on false, persistent versus temporary data, a logged signal for off-chain use, and reusable precondition logic.

Correct answer: B
A modifier packages reusable precondition logic, an event is a logged signal for off-chain consumers, storage versus memory is persistent versus temporary data, and require is a guard that reverts when its condition is false. Only option B maps each concept correctly.

Question 6

Domain 5: Security and Common Vulnerabilities

CONFIG

A privileged function is written as shown.

```
function setFee(uint newFee) public {
  fee = newFee;   // no access check
}
```

Listing 6.1 The setter.

What is the problem, and the correct fix?

A. No problem exists, because any account being able to change the fee is a normal and intended behavior for a public setter.

B. It has no access control; restrict it to an authorized role, for example with an owner or role check that reverts otherwise.

C. The problem is the parameter name, so renaming newFee to something shorter is the change that secures this function properly.

D. The problem is that it is public, so marking it external without any access check fully resolves the security issue here.

Correct answer: B
Any account can call setFee and change the fee because there is no access control. The fix is to restrict it to an authorized owner or role that reverts for everyone else. Open access is not intended, the parameter name is irrelevant, and external without a check is still unrestricted.

Question 7 Domain 1: Smart Contract Foundations

A developer assumes a deployed contract can be freely edited later like ordinary server code. How should you correct this?

A. The developer is right, because deployed contract code can be edited in place at any time exactly like a normal web server.

B. Deployed contract code is generally immutable; to change behavior you plan for it, for example with an upgradeable proxy pattern.

C. Deployed code can be edited only by raising the gas limit, which unlocks in-place editing of the live contract bytecode.

D. Deployed code changes automatically each block, so the developer never needs to plan for upgrades or migrations at all.

Correct answer: B
Once deployed, contract code is generally immutable, so changing behavior must be planned for up front, commonly through an upgradeable proxy pattern or a migration. Code cannot be edited in place, gas limits do not unlock editing, and code does not change on its own.

Question 8 Domain 6: Gas, Deployment and Operations

Before deploying a contract that will hold significant funds to mainnet, what is the most responsible step?

A. Deploy straight to mainnet immediately, because testing only wastes time when the code appears to compile without any errors.

B. Have it independently audited and thoroughly tested first, since a mainnet contract holding funds cannot easily be patched later.

C. Skip testing and rely on users to report bugs, since real users will find problems faster than any test suite ever could.

D. Raise the gas limit as high as possible, because a higher gas limit removes the need to audit or test the contract at all.

Correct answer: B
A contract holding significant funds is hard to patch once live, so independent audit and thorough testing before mainnet are the responsible steps. Deploying untested code, relying on users to find bugs, or raising the gas limit does nothing to reduce the risk of loss.

Question 9 Domain 3: Standards and Patterns

Why should a smart contract engineer favor established, audited token standards over a bespoke implementation for a fungible token?

A. Because bespoke code is always faster, so a custom token will outperform any established standard in every situation on chain.

B. Established standards are widely reviewed and interoperate with wallets and tools, reducing bugs and integration effort.

C. Because standards are legally required, and writing any custom token contract is prohibited on public ledgers by protocol rules.

D. Because custom tokens cannot hold any value at all, so only tokens using an established standard are able to store a balance.

Correct answer: B
Established standards are heavily reviewed and interoperate with existing wallets, exchanges and tooling, which lowers both bug risk and integration effort. Bespoke code is not inherently faster, standards are not legally mandated, and custom tokens can technically hold value but lose interoperability and assurance.

Question 10 Domain 2: Contract Development

An engineer wants a function to reject calls that send an incorrect payment amount. What is the idiomatic approach?

A. Let the function continue silently on a wrong amount, because reverting on bad input frustrates users and should be avoided.

B. Validate the amount with a require or revert so the whole transaction reverts and state changes are rolled back on failure.

C. Log the wrong amount to an event and proceed, since an event by itself prevents any incorrect state from being written.

D. Store the wrong amount and correct it in a later transaction, because on-chain input cannot be validated as it arrives.

Correct answer: B
The idiomatic approach is to validate with require or revert so an incorrect amount reverts the entire transaction and rolls back state, keeping the contract consistent. Continuing silently corrupts state, an event alone does not prevent writes, and input can and should be validated on arrival.

Published by the CertLabz Certification Board, a division of CertLabz LLC. Companion to objectives document CSCE-001. https://certlabz.com/certifications/certified-smart-contract-engineer.html