CERTLABZ CERTIFICATION BOARD

Certified Go Programmer (CGOP)

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 Go programmer 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 1: Go Fundamentals and Syntax

CONFIG

James Carter writes a small Go program and checks how it starts.

```
package main

import "fmt"

func main() {
    fmt.Println("Ready")
}
```

Listing 1.1 A minimal Go program.

Which function does the runtime call to start the program?

A. The init function

B. The main function

C. The first import

D. The Println call

Correct answer: B

An executable Go program begins at func main inside package main, which the runtime calls once the program starts. The init function runs first for setup but is not the entry point, and imports and library calls do not start the program.

Question 2 Domain 2: Types, Slices and Maps DATA

Ava Thompson maps each Go type to its zero value, with one entry missing.

Type	Zero value
int	0
bool	(to choose)
string	Empty text
slice	nil

Table 2.1 Types and their zero values.

What is the zero value of a bool?

A nil
B true
C false
D Empty string

Correct answer: C
A declared bool that is never assigned holds its zero value, which is false. Slices and maps zero to nil and numbers to 0, but a bool is simply false until code sets it otherwise.

Question 3 Domain 3: Control Flow and Functions CONFIG

Olivia Davis studies how deferred calls run before a function returns.

```
func main() {
    defer fmt.Print("A")
    defer fmt.Print("B")
    fmt.Print("C")
}
```

Listing 3.1 Two deferred prints.

What does the program print?

A ABC
B CAB
C BAC
D CBA

Correct answer: D
Print("C") runs first, then the deferred calls fire in last in, first out order as the function returns, so B prints before A. The result is CBA, which is why stacking defers reverses their apparent order.

Question 4 — Domain 4: Structs, Interfaces and Methods — CONFIG

Liam Anderson checks what lets a struct satisfy an interface in Go.

```
type Speaker interface {
    Speak() string
}

type Dog struct{}

func (d Dog) Speak() string {
    return "Woof"
}
```

Listing 4.1 An interface and a type.

What makes Dog satisfy Speaker?

A. Implement Speak
B. Embed the interface
C. Import the interface
D. Register with Go

Correct answer: A
Go interfaces are satisfied implicitly, so Dog satisfies Speaker simply by defining the Speak method with the right signature. There is no keyword to register or import an interface, and embedding is not required for a plain method set to match.

Question 5 — Domain 5: Concurrency with Goroutines and Channels — CONFIG

Emma Wilson traces a value sent from a goroutine over a channel.

```
ch := make(chan int)
go func() {
    ch <- 42
}()
fmt.Println(<-ch)
```

Listing 5.1 A goroutine and a channel.

What does the program print?

A. 0
B. deadlock
C. 42
D. nil

Correct answer: C
The unbuffered send in the goroutine pairs with the receive in main, so the value 42 passes across and prints. There is no deadlock because the receiver is ready, and a channel carries the sent value rather than a zero or nil.

Question 6 Domain 6: Packages, Errors, Testing and Tooling DATA

David Brown maps each go command to its purpose, with one purpose missing.

Command	Purpose
go build	Compile packages
go test	(to choose)
go fmt	Format source
go mod	Manage modules

Table 6.1 Common go commands.

What does go test do?

A. Publish a release
B. Run tests
C. Deploy the binary
D. Lint every file

Correct answer: B
The go test command builds and runs the test functions in a package and reports pass or fail. It does not deploy or publish anything, and formatting or linting are handled by separate tools such as go fmt.

Question 7 Domain 1: Go Fundamentals and Syntax

A function defined in one package cannot be called from another package, even though the code compiles. The team wants it visible outside its package. What makes a Go identifier exported?

A. An export keyword
B. A public modifier
C. A go.mod entry
D. A capital letter

Correct answer: D
Go exports an identifier when its name starts with a capital letter, so renaming the function to begin with an uppercase letter makes it visible to other packages. Go has no export keyword or public modifier, and go.mod tracks modules rather than visibility.

Question 8 Domain 3: Control Flow and Functions

A function that reads a file needs to tell the caller when the read fails, following idiomatic Go. What should the function return on failure?

A. A thrown exception
B. A false boolean
C. A non-nil error
D. A panic call

Correct answer: C
Idiomatic Go returns a non-nil error as the last return value so the caller can check it and respond. Go has no exceptions, and panic is reserved for unrecoverable situations rather than ordinary failures like a missing file.

Question 9 Domain 5: Concurrency with Goroutines and Channels

Several goroutines update the same counter variable at once, and the total comes out different from run to run. What best fixes this data race?

A. Guard with a mutex
B. Add more goroutines
C. Sleep between writes
D. Ignore the warning

Correct answer: A
A data race happens when goroutines touch shared state without coordination, so guarding the counter with a mutex or moving updates to a single owner makes the total reliable. Adding goroutines or sleeping does not remove the race, it only hides it sometimes.

Question 10 Domain 6: Packages, Errors, Testing and Tooling

A team starts a new Go project and wants its dependency versions tracked and reproducible across machines. Which step sets up module based dependency management?

A. Copy a vendor dir
B. Edit the PATH var
C. Install a linter
D. Run go mod init

Correct answer: D
Running go mod init creates a go.mod file that records the module path and pins dependency versions for reproducible builds. Editing PATH or installing a linter does not manage dependencies, and copying folders by hand skips the version tracking that modules provide.

CGOP-001. https://certlabz.com/certifications/certified-go-programmer.html