CS 2001 - Go Test

Fall 2016

Name: _____

Instructions:

1. Read all instructions or suffer the consequences.
2. No materials other than your pencil and eraser are allowed or required for this test.
3. Do not cheat. Don't even try.
4. Securely store all Gophers, carts, programming manuals, etc. during the test.
5. Place answers in blanks if they are provided. If you are asked to write a short answer, make it literate.
6. If you have a question about a problem/question, be sure to ask.
7. If you are asked to give code for a problem, give all pertinent code.
8. Unless otherwise indicated, you may assume all code snippets are embedded in otherwise-compiling programs.
9. Good luck.

Problem 0

Choose the best option for each of the following. Clearly indicate your selection.

1. (3 pts) If included in an otherwise compilable program, this snippet will cause a compiler error.

```
x := 12
if x < 10
{
        fmt.Println("Our toaster is also confused. .")
        fmt.Println("It doesn't know why we put bagels in it.")
}
```

   a. True
   b. False

2. (3 pts) The `export` keyword is used to export items (e.g., functions, variables, constants, types) from a package.

   a. True
   b. False

3. (3 pts) An otherwise uninitialized string variable (e.g., `x` in `var x string`) is implicitly initialized to the empty string.

   a. True
   b. False

4. (3 pts) Accessing a non-existent key for a map raises an exception. *Note that this code snippet is syntactically valid.*

```
m := map[string]int{"dog": 42}
fmt.Println(m["kittycat"]) // No such key. Exception!
```

   a. True
   b. False

5. (3 pts) A variable of type empty interface (`interface{}`) can be used to store a value of **any** type.

   a. True
   b. False

6. (3 pts) Go will implicitly convert types to aid with comparison. For example, `x` of type `int` and `y` of type `float64` can be compared as such: `x <= y`

   a. True
   b. False

7. (8 pts) Match 'em up.

   A(n) `___________` statement is an idiomatic alternative to a big ol' `if-else` chain.

   A(n) `___________` statement compares types instead of values.

   A(n) `___________` statement chooses which of a set of send/receive operations will proceed.

   A(n) `___________` statement can be used to continually receive from a non-closed channel.

   a. `select`
   b. `expression switch`
   c. `for`
   d. `type switch`

Problem 1

Consider the following (syntactically correct) program snippet.

```
package main

import "fmt"

type T struct {
        x int
}

func (t T) V()  { t.x++ }
func (t *T) P() { t.x++ }

func main() {
    var t T
    fmt.Print(t, " ") // The output for this single-field struct looks like this: {0}
    t.P()
    fmt.Print(t, " ")
    t.V()
    fmt.Print(t)
}
```

1. (3 pts) Write the **complete** output in the space below. Note that there are three calls to `fmt.Print` (it works just like `fmt.Println`, but without the trailing newline).

2. (5 pts) Fill in the blanks.

   For the above type `T`, the method `V()` has a _____ receiver.

   For the above type `T`, the method `P()` has a _____ receiver.

Problem 2

Consider the following Go program snippet:

```
func f(x []int) {
    x[0] = 10
}

func g(y []int) {
    y = append(y, 10)
}

func main() {
    s := []int{1, 2, 3, 4} // A slice of length 4 and capacity 4
    f(s)
    fmt.Println(s)

    t := []int{1, 2, 3, 4} // A different slice of length 4 and capacity 4
    g(t)
    fmt.Println(t)
}
```

1. (4 pts) What is the output of the above program snippet?

2. (6 pts) Explain how you reached your answer above. Draw diagrams of slices as necessary to show how slices are modified by function calls.

Problem 3

Consider the following program snippet:

```
c := make(chan int)
c <- 50
fmt.Println(<-c)
```

1. (3 pts) What, if anything, is output from running the above snippet? What, if any, errors are generated?

2. (3 pts) Briefly describe how you reached your above answer.

Problem 4

Consider the following program snippet:

```
c := make(chan int, 3)
c <- 1
c <- 2
c <- 3
fmt.Println(<-c)
fmt.Println(<-c)
fmt.Println(<-c)
```

1. (4 pts) What, if anything, is output from the above program? What, if any, errors are generated?

2. (6 pts) Name two **different** ways to cause a deadlock in the above program using only the channel `c`.

Problem 5

(8 pts) Add to the program below, so that the `Name` type implements the `fmt.Stringer` interface. Name components should be space-separated[1] in the string representation. For example, `t` would be output by `fmt.Println` as `Tina Ruth Belcher`.

```
package main

import "fmt"

// From the fmt package...
// type Stringer interface {
//         String() string
// }

type Name struct {
    First, Mid, Last string
}
```

```
func main() {
    t := Name{"Tina", "Ruth", "Belcher"}
    fmt.Println(t)  // Tina Ruth Belcher
}
```

[1] Remember that you can concatenate strings with the `+` operator.

Problem 6

Code Point	UTF-8 (# bytes)	UTF-16 (# 16-bit code-units)	UTF-32 (# 32-bit code-units)
o	0x6f (1)	0x6f (1)	0x0000006f (1)
p	0x70 (1)	0x70 (1)	0x00000070 (1)
y	0x79 (1)	0x79 (1)	0x00000079 (1)
ñ	0xc3 0xb1 (2)	0xf1 (1)	0x000000f1 (1)
🐸[2]	0xf0 0x9f 0x90 0xb8 (4)	0xd83d 0xdc38 (2)	0x0001f438 (1)

For each of the following prompts, be sure to…

- Draw your boxes, so that it is clear where one byte / code-unit ends and where the next one begins.
- Indicated which box(es) correspond to which code point.
- Indicate the width (in **8-bit bytes**) of the boxes.

1. (3 pts) Draw Go's representation of "🐸poñy🐸" if it was stored as a `[]rune`.

2. (3 pts) Draw Go's representation of "🐸poñy🐸" if it was encoded using UTF-16 and stored as a `[]uint16`.

3. (3 pts) Draw Go's representation of "🐸poñy🐸" if it was encoded using UTF-8 and stored as a `string`.

4. (3 pts) Fill in the table based on your responses to questions 1 through 3.

Type	Memory required to store "🐸poñy🐸" (in **8-bit bytes**)
UTF-8	
UTF-16	
UTF-32	

[2] It's a frog.

Problem 7

```
package main

import "fmt"

func count(start, stop int, output chan int) {
        for i := start; i <= stop; i++ {
                output <- i
        }
        close(output)
}

func add(input1, input2, output chan int) {
        for {
                i1, ok1 := <-input1
                i2, ok2 := <-input2
                if !ok1 || !ok2 {
                        close(output)
                        break
                }
                output <- i1 + i2
        }
}

func main() {
        nums := make(chan int)









        for x := range nums {
            fmt.Println(x)
        }
}
```

1. (8 pts) Finish writing the body of `main()`, so that …
    a. the goroutines and channels are setup as shown above.
    b. the program output matches the output above.

2. (4 pts) Briefly describe how and why the **output** of the program changes if `add()` doesn't check for closed channels and is defined as:

```
func add(i1, i2, o chan int) {
        for {
                o <- (<-i1 + <-i2)
        }
}
```

Hint: What happens when you receive from a closed channel?

Problem 8

Consider the following program:

```
package main

import (
        "fmt"
        "time"
)

func main() {
        c := make(chan int)

        // Starts a timer in a separate goroutine and returns a chan time.Time
        // Sends the current time over the returned channel after the specified delay
        quit := time.After(5 * time.Millisecond) // type <-chan time.Time

        go func() {
                for i := 0; ; i++ {
                        c <- i
                }
        }()

        for {
                select {
                case val := <-c:
                        fmt.Println(val)
                case <-quit:
                        return
                }
        }
}
```

1. (4 pts) Draw a diagram to the right of the program showing how `main()` and its goroutines are connected with channels.
2. (4 pts) Briefly describe (in words) the output of this program. When (if ever) does it terminate?

Bonus

1. (5 pts) Draw a hand turkey.