CS245 Midterm 1

Name:

Start Time:

Finish Time:

I have abided by the Honor Code. I have not discussed this test with anyone.
(Sign below)

If you take this test on separate sheets of paper, make the above your first page.

There are 8 questions in this test. All questions have equal weight. Be sure to answer all of the questions.

1. Semicolons – the semicolon “wars” were about whether semicolons separate statements or terminate statements. Distinguish between these (do not just use the example from the article). How are semi-colons used in Java and Go (separator, terminator, or something else?) Explain. Illustrate with examples.

2. In Go, the type interface{} can be used kind of like Object in Java; that is, variables declared to be of “interface{}” can store anything. For instance, the following is a perfectly fine function in Go

```
func mm() {
    var m3 interface{}
    m3=9
    fmt.Println(m3)
    m3="hello"
    fmt.Println(m3)
}
```

Now, consider arrays and slices as in the two code snippets. One of these snippets works (compiles and runs without error). One does not. Explain (if you are not sure, then speculate).

```
func main() {
   bb := make([]interface{}, 2)
    bb[0]="hello"
    bb[1] = 9
    fmt.Println(bb)
}
```

```
func main() {
    var bb [2]interface{}
    bb[0]="hello"
    bb[1] = 9
    fmt.Println(bb)
}
```

3. Go is statically typed. This is true. Yet, in many situations the programmer is not required to identify the type. Explain how Go can relieve programmers of this burden. Finally, do you consider this ability of Go a win; a loss or meh. Why?

4. Arrays in Java and Go behave rather differently. This difference is especially evident when arrays are passed into subroutines, created within subroutines and returned from subroutines. Write two very brief programs – they need actually only be parts of programs – you will not loose points for small syntax errors – that illustrate these differences. Explain how your programs make these differences evident.

5. For the following programs in Java and Go for each variable in the program. state whether that variable is: allocated at compile time, at run time from the stack, and at run time from the heap.

```go
// prints appear below solely to satisfy
// Go usage requirements
package main

import "fmt"
var a int
var pi float64=3.1415
func main(){
   m2(20)
   ht := make(map[string]string)
   z:="17"
   ht[z]=z
   fmt.Printf("%v\n",ht)
}
func m2(mm int) {
   var q []string
   q=append(q,"qq")
   var r [6]int
   r[1]=mm
   fmt.Printf("%v\n", q)
   fmt.Printf("%v\n", r)
}
```

```java
import java.util.HashMap;

public class PP {
   static int p1;
   static String p2 = "this is a test";
   int[] vv = new int[10];
   public PP(int ii) {
      PP.p1=42;
      vv[0] = ii;
   }
   public static void main(String[] args) {
      int aa = 5;
      PP inst = new PP(aa);
      inst.p3(17);
   }

   public void p3(int prm) {
      Integer[] ali = new Integer[6];
      ali[1] = prm;
      HashMap<String, String> hmss = new HashMap<>();
      hmss.put("this", PP.p2+prm);
   }
}
```

6. == is not defined over slices in Go. Write a complete Go program that contains a method named equals that returns true if two slices have exactly the same contents and false otherwise.

What restrictions are there of your equals function.   That is, could it work on a slice slice type (with minimimal modifications) and if not, why?

7. For the following recursive program, show the function stack and all important parts of each stack frame (as discussed in class) in the stack when just before executing line 19 when bval==9.

```
 1  package main
 2
 3  import "fmt"
 4
 5  func main() {
 6          a(0)
 7  }
 8
 9  func a(val int) string {
10          if val < 3 {
11                  return b(val + 1)
12          } else if val < 9 {
13                  return b(val + 2)
14          }
15          return "aa"
16  }
17
18  func b(bval int) string {
19          a(bval + 2)
20          return fmt.Sprintf("b%d", bval)
21  }
```

8. For the following program, identify every scope and the variables that are active in that scope under static scoping. Identify scopes using the line numbers. If there is any possibility of name ambiguity (and there will be) be sure to clear identify the active variables clearly.

```
 1  package main
 2
 3  var ss int = 19
 4
 5  func main() {
 6          aa := "ss"
 7          if aa == "bb" {
 8                  ss := 210
 9                  f2(ss)
10          } else {
11                  f2(ss)
12          }
13  }
14  func f2(ss int) {
15          f := func(ss int) {
16                  if ss > 7 {
17                          qq := ss / 2
18                          println(qq)
19                  }
20          }
21          for qq := ss; qq > 0; qq-- {
22                  f(qq)
23          }
24  }
```