CS245 Midterm 2

Name:

Start Time:

Finish Time:

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

Accommodations: If you have given me an accommodation letter, please remind me here

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

Typography: In the test all code is in blue. All program output appears in brown. Question text is in black.

There are 10 questions in this test in two parts. The first part is short answer; the second is programming. Answer 6 of questions 1-7 (part 1) and 1 of questions 8-10 (part 2). For a total of 7 answers. There is NO extra credit for extra answers.

All questions have equal value.

(Part 1 – answer 6 of 1-7)
1. The type of a function in Go is considerably longer and more detailed than in the type of a function in Elixir. Describe function typing in each language and explain why Go and Elixir are so different in this respect. Examples will help.

(Part 1 – answer 6 of 1-7)
2. Scott lists the following as characteristics of most functional languages:

```
      lists
      recursion
      first class functions
      garbage collection
```

Why are each associated with functional programming languages.

(Part 1 – answer 6 of 1-7)
3. The output of these two nearly identical programs (one in Go, one in Elixir) is different. Explain.

Go

```go
package main

import "fmt"

func  main() {
    r:=makeFunn(5)
    fmt.Printf("%v\n", r())
    fmt.Printf("%v\n", r())
    fmt.Printf("%v\n", r())
}

func makeFunn(inp int) func() int
{
    f:=func() int {
        inp=inp+inp
        return inp
    }
    return f
}
```

Output:

```
10
20
40
```

Elixir

```elixir
defmodule Main do


def main() do
    r = makeFunn(5)
    IO.puts(r.())
    IO.puts(r.())
    IO.puts(r.())
  end

 def makeFunn(inp) do
    fn () ->
      inp=inp+inp
      inp
    end
  end
end

Main.main
```

Output:

```
10
10
10
```

(Part 1 – answer 6 of 1-7)

4. Consider the following (copied from the elixir interpreter)

```
iex(1)> a=["This",17,"That",43.5, 1712]
["This", 17, "That", 43.5, 1712]
iex(2)> aa=a
["This", 17, "That", 43.5, 1712]
iex(3)> b=[{1,2} | a]
[{1, 2}, "This", 17, "That", 43.5, 1712]
iex(4)> c=aa++[{1,2}]
["This", 17, "That", 43.5, 1712, {1, 2}]
```

Why is the appending method from line 3 preferred over the method from line 4? (The answer has nothing to do with the order of the items.) Be specific.

What values are stored in a,aa,b,c after line 4? Explain.

(Part 1 – answer 6 of 1-7)
5. Elixir appears to be (and claims to be) dynamically typed. On the other hand, variables in Elixir are certainly immutable. Question: does immutability make Elixir, at least effectively, statically typed? Explain. (A good explanation will receive full credit, regardless of a yes or no answer.)

(Part 1 – answer 6 of 1-7)
6. Consider the following two Java classes. One does not compile (ALCov), the other throws a runtime exception. Both problems are related to co-variance. Describe, exactly what the problems with each class are and why they exist in Java.

```
import java.util.ArrayList;
public class ALCov {
    private void changeNAL(ArrayList<? extends Number> al) {
        for (int i = 0; i < al.size(); i++)
            al.set(i,Double.valueOf(i+13.2));
    }
    public void testArrayList() {
        ArrayList<Number> nArr = new ArrayList<>();
        ArrayList<Integer>iArr = new ArrayList<>();
        for (int i = 0; i < 5; i++) {
            nArr.add(i + 10.5);
            iArr.add(i, i);
        }
        changeNAL(nArr);
        changeNAL(iArr);
        for (int i = 0; i < nArr.size(); i++) {
            System.out.println("N " + nArr.get(i));
            System.out.println("I " + iArr.get(i));
        }
    }
    public static void main(String[] args) {
        new ALCov().testArrayList();
    }
}
```

```
public class ArraCov {

    private void changeN(Number[] arr) {
        for (int i = 0; i < arr.length; i++)
            arr[i] = i+13.2;
    }
    public void testArray() {
        Number[] nArr = new Number[5];
        Integer[] iArr = new Integer[5];
        for (int i = 0; i < nArr.length; i++) {
            nArr[i] = i + 10.5;
            iArr[i] = i;
        }
        changeN(nArr);
        changeN(iArr);
        for (int i = 0; i < nArr.length; i++) {
            System.out.println("N " + nArr[i]);
            System.out.println("I " + iArr[i]);
        }
    }

    public static void main(String[] args) {
        new ArraCov().testArray();
    }

}
```

(Part 1 – answer 6 of 1-7)
7. Describe the differences between deep vs shallow equality checking and structural vs name type equivalence. A complete answer will need to define all four terms and will probably be helped along by some well chosen examples.

Shallow vs. Deep equality checking:
Shallow equality checking sees that the pointers of the two values are the same. For

example, in Java, == does a shallow check. For example, in the following code: Int[] a = {1, 2}

int [] b = {1, 2}

A == b // this will be false, because the pointers of the two objects are different places in memory.

Deep equality checking follows the pointer to the place in memory and checks that the actual VALUES are the same. For example, Equivalent code in elixir, a language for which == does a deep comparison:

A= [1, 2]
B = [1, 2]
A == b # will return true, because elixir performs a deep comparison of the lists.

Name type equivalence vs. Structural equivalence

Name equivalence is the norm in Java and Go, as it is easy to do and be checked by a compiler. If two types are to be compared, it first checks to see that the names of the types are the same. For example, in Java-- user-defined types (classes) will be considered equivalent if they have the same type name. If you defined two classes with different names, but the exact same insides, they would not be considered equivalent in the eyes of the compiler.

ClassA {
Int test = 0;

}
Class B {

Int test = 0; }

For a function defined as:
Public static func(A param) { }

If you were to input an object of class B, then java will throw a big hissy fit. This means that java is looking at the name of the class, not the things inside the class, to determine if they are the same thing.

The only real place that we see Structural equivalence come up is in Go, when talking about structs:

If these two structs were defined in Go:

Type A struct { A int32

B string }

Type B struct { A int32

B string }

These two classes would be considered equivalent by the program. Note that if the order of A and B were swapped in the struct, then they would not be considered structurally equivalent.

(Part 2 – answer 1 of 8-10)
8. Write a program in Elixir to find the second largest number in a list of numbers. You may assume that the input will always consist of a single list and that the list will contain only numbers. Do not use anything in Enum. If the list is empty, return 0. If the list has only one item, return that item.

```
defmodule Sec do
  @doc """
  Find the smallest, in a rather imperative way.
  Just go through the list and use if to check.
  At least use tail recursion to do the work
  first parm is the current smallest item
  second is the list
  """
  def sst(cs,[]) do
    # return the smallest
    cs
  end
  def sst(cs, [h|r]) do
    if cs<h do
      sst(cs,r)
    else
      sst(h,r)
    end
  end

  @doc """
  Must the same as sst, but does the work using matching
  """
  def sst2(cs,[]) do
    cs
  end
  def sst2(cs, [h|r]) when h<cs do
    sst2(h,r)
  end
  def sst2(cs, [_|r]) do
    sst2(cs, r)
  end

  @doc """
  Here we pass the smallest and the second smallest along in the
  recursion, so can find second smallest in a single pass
  through the list.  But there is a lot more work and it does not
  scale to find the nth smallest.  Note th use of both sst3/1 and
  sst3/3
  """
  def sst3([]) do
    0
  end
  def sst3([h|[hh|r]]) do
    if h<hh do
      sst3(r,h,hh)
    else
      sst3(r,hh,h)
    end
  end
  def sst3([h|r]) do
    h
  end
  def sst3([], smllest, next_small) do
    next_small
  end
  def sst3([h|r], smllest, _next_small) when h<smllest do
    sst3(r, h, smllest)
  end
  def sst3([h|r], smllest, next_small) when h<next_small do
    sst3(r, smllest, h)
  end
  def sst3([_|r], smllest, next_small) do
    sst3(r, smllest, next_small)
  end

  @doc """
  The top-level code to get the second smallest
  using only a smallest finder.
  """
  def smll([]) do
    0
  end
  def smll([h]) do
    h
  end
  def smll([h|r]) do
    smallone = sst2(h,r)
    [sh|sr] = [h|r] -- [smallone]
    sst2(sh, sr)
  end
end
IO.inspect Sec.smll([2,3,4,5,6,7])
IO.inspect Sec.smll([-2,-4,-9,-8,-7,-3])
IO.inspect Sec.sst3([2,3,4,5,6,7])
IO.inspect Sec.sst3([-2,-4,-9,-8,-7,-3])
```

(Part 2 – answer 1 of 8-10)
9. Horner’s method is a technique for converting a string into a number that is fairly unique from the number derived from other strings. The idea is to take the string, and work with the ASCII value of each character. First, choose a prime number – call it P. Working from either the beginning or end of the string, multiply the ASCII value of the first character by P^(n-1), add that to the second ASCII value multiplied by P^(n-2) etc.
So, for instance, if the word is “Act” and P=3 and we are working from the front of the string the calculation would be
(3^2)*’A’+(3^1)*’c’+(3^0)*’t’
9*65+3*99+116
998
Another way of phrasing this calculation is:
((‘A’*3)+’c’)*3+’t’

To transform a String into a list of chars in Elixir use to_charlist(string). For example:

```
iex(11)> to_charlist("Act")
‘Act'
```

Note that that return value is in single quotes. This is how Elixir chooses to render character lists.

Implement Horner’s method in Elixir. You may hard-code the value P=3

```
defmodule ESum do
  def main(lst) do
    Enum.reduce(lst, 0, fn(elem, acc) ->
(acc*3+elem) end)
  end
end
IO.puts ESum.main(to_charlist("Act"))
```

(Part 2 – answer 1 of 8-10)
10. The function split245 takes two parameters, a value on which to split, and a list to be split.
It returns two lists contained in a tuple. The first list is items less than the value, the second is items greater than the value. Items equal to the value should be thrown out. Order in the returned lists is not significant.
For example (given that split245 is defined within the module Ss):
Ss.split245(6, [2,4,7,8,6,9,5,2,4])
would return
{[2,4,5,2,4], [7,8,9]}
Note that this example shows order preserved; reversed or anything else is fine.
Write the split245 function. It should be tail recursive and have a complexity of O(n) when n is the length of the provided list. A full credit answer will have NO if expressions. (each "if" will loose 1 point)

```
defmodule Spl do
@moduledoc “””
NOTE: this is essentially the splitter used by Quicksort.
Three(!) different splitting functions.  They all work!
"""
  @doc """
  A not-tail recursive solution to the splitting problem.
  This version does use an if, which I could avoided by
  the recursive pair of functions trick in rr (below), or with a guard
clause, but given
  that I am not doing tail recursion I choose to use an if and
  loose a point.
  """
  def ntr([], _spl) do
    {[], []}
  end
  def ntr([spl|r], spl) do
    ntr(r, spl)
  end
  def ntr([h|r], spl) do
    a=ntr(r, spl)
    if h<spl do
      {[h|elem(a,0)], elem(a,1)}
    else
      {elem(a,0), [h | elem(a,1)]}
    end
  end

  @doc """
  Split the list into two parts. This kind of cheats on the
  "no if" requirement in that it uses a "when" guard clause.
  """
  def ss([], _, lss, mre) do
    {lss, mre}
  end
  def ss([h|r], h, lss, mre) do
    ss(r, h, lss, mre)
  end
  def ss([h|r], spl, lss, mre) when h<spl do
    ss(r, spl, [h|lss], mre)
  end
  def ss([h|r], spl, lss, mre) do
    ss(r, spl, lss, [h|mre])
  end
  def spl(lst, spl) do
    ss(lst, spl, [], [])
  end

@doc """
Do the splitting completely without if statements
(and no guards either!) so ONLY matching.  I did this
by using a pair of recursive functions, with differ in that
one of the pair has an extra parameter. That extra param is a boolean
to determine whether the item on the head of the the list should
go into the smaller or the larger of the splits. With that handled
by matching, then call the first function again.
Question, can you have a tail-recursive pair of functions.
Probably not.
"""
  def rr([], _, lss, mre) do
    {lss, mre}
  end
  def rr([h|r], h, lss, mre) do
    # case where split value is in list, throw it out
    rr(r, h, lss, mre)
  end
  def rr([h|r], spl, lss, mre) do
    rr3(h, r, spl, lss, mre, h<spl)
  end


  def rr3(h, r, spl, lss, mre, true) do
    rr(r, spl, [h|lss], mre)
  end
  def rr3(h, r, spl, lss,mre, _) do
    rr(r, spl, lss, [h|mre])
  end

  def splr(lst, spl) do
    rr(lst, spl, [], [])
  end

end

IO.inspect Spl.ntr([-1,-2,-3,-4,-5,-6,-7,-6,-5,-6,-7,-8,-9], -6)
IO.inspect Spl.ntr(~w[this is a test of the dead sty eye], "m")
```