CSCI 3155: Spring 2025 - Sample Mid-Term 1 (15 points). Time Limit: 60 minutes **A**

Answer all questions in place. Please do not attach extra sheets. If you have anything to tell us or your answer is overflowing, please use the "notes" spaces given next to each problem.

Your Name: ______________________

Your Student Id: ______________________

P1 (4 points) For the scala function below, write down *all* the correct facts from the ones given besides the code.

```
def foo(x: Int): Int = {
    if ( x > 20 ) {
        return 1 + foo(x-1)
    }
    if ( x < 100) {
        return 0
    }
    return foo(x-1)
}
```

[✓ (A)] The function is *not* tail recursive.

(B) Scala will complain at compile time that the function's return type is mismatched.

(C) It compiles successfully without an error but throws an exception at runtime for certain values of input x.

(D) It compiles successfully without an error but fails to terminate for some values of input x.

[✓ (E)] It compiles and runs successfully for all integer values of input x.

Write down all of the options from (A) -(D) that are correct. No justifications needed.

Answers

A, E

P2 (8 points) Below is a a recursive function to compute the count of odd numbers in a list. Note that in scala `l.head` returns the very first element of a list and `l.tail` returns a list that includes all but the very first element from the list l.

```
def countNumOdd(l: List[Int]): Int = {
    if (l.length == 0) 0
    else if (l.head % 2 == 1) 1 + countNumOdd(l.tail)
    else countNumOdd(l.tail)
}
```

Fill up the blanks to complete the tail recursive version of the `countNumOdd` function.

```
def countNumOddTail(l: List[Int], acc: Int = __0____ ): Int = {
    if (l.length == 0) ____acc_______________//Answer here
    else if (l.head % 2 == 1) countNumOddTail(l.tail, acc+1)//Answer here
    else countNumOddTail(l.tail, acc)__________________ //Answer here
}
```

Notes

P3 (3 points) Complete the missing portions in the scala definition corresponding to the grammar for regular expressions given below. Please keep the names of terminals and non-terminals unchanged.

$$\begin{array}{lll} \textbf{Regex} & \Rightarrow & \mathit{Match}(\textbf{String}) \\ & | & \mathit{Empty}() \\ & | & \mathit{Or}(\textbf{Regex}, \textbf{Regex}) \\ & | & \mathit{Seq}(\textbf{Regex}, \textbf{Regex}) \\ & | & \mathit{Star}(\textbf{Regex}) \end{array}$$

Note: **String** refers to the in-built scala type.

```
sealed trait Regex // Your answer below

case class Match(s: String) extends Regex

case class Empty() extends Regex

case class Or(r1: Regex, r2: Regex) extends Regex

case class Seq(r1: Regex, r2: Regex) extends Regex

case class Star(r: Regex) extends Regex
```

Notes

P4 (7 points) The evaluation for `let` bindings we discussed in class/notes is called "eager" evaluation because we *eagerly* evaluate `let` bindings whether they are needed or not. For example, the following `let` expression never uses the value of x, but eager evaluation nonetheless computes `fact(50)`:

```
let x = fact(50) in
200
```

An alternative to eager evaluation is "lazy" evaluation where we evaluate a `let` binding if and only if it is necessary. A simple way to carry out lazy evaluation is by changing the type of the environment *env* from `Map[String,Value]` to `Map[String,Expr]`. In other words, a `let` variable is now bound to its defining expression e (instead of the result of evaluating e) in the environment *env*. We define the following semantic rule for the `let` expression under our version of lazy evaluation:

$$\frac{}{\mathsf{eval}(\mathsf{Let}(id, e_1, e_2), env) = \mathsf{eval}(e_2, env \circ \{id \to e_1\})}\text{(lazy-let-ok)}$$

Indeed, we need to change the variable/identifier evaluation rule to make the lazy evaluation work:

$$\frac{id \in dom(env)}{\mathsf{eval}(\mathsf{Ident}(id), env) = \mathsf{eval}(env(id), env)}\text{(lazy-var-ok)}$$

The error rule for variables remains the same as discussed in class notes: if $id \notin dom(env)$ then $\mathsf{eval}(\mathsf{Ident}(id), env)$ evaluates to error. The other rules remain the same as well (refer to class notes since this is just a sample exam).

(a, 3 points) Consider the following program:

```
let x = 10 in
let y = x + 5 in
let x = 22 in
 y + 10
```

$\emptyset$

$(x \to 10)$

$(x \to 10, y \to x+5)$

$[x \to 10, y \to x+5, \boxed{x \to 22}]$

$\boxed{x}+5+10$

22

$= 37$

What value does it evaluate to under the lazy semantics

Answer

37

(a, 4 points) Consider four lettuce programs as shown below. Which among the 4 evaluate differently under the new rules compared to the standard old rules?

[✓ **(A)**]

```
let x = 10 in
let y = x + 5 in
let x = 22 in
 y + 10
```

old: 25
New: 37

(B)

```
let y = let x = 10 in
          x + 5 in
let x = 22 in
y + 10
```

[✓ **(C)**]

```
let z = let x = 0 in
          200/x in
let y = z+1 in
300
```

(old: Error, New: 300)

(D)

```
let w = let x = 0 in
          200/x in
w+10
```

P5 (4 points) Fill in the missing parts of the following two rules for `and` expression such that it evaluates as expected.

(a, 2 points) Complete the "and-ok-long" rule for the non-error case when evaluating e_2 is necessary:

$$\frac{\mathsf{eval}(e_1, env) = v_1,\ v_1 \neq \mathsf{error},\ v_1 = true}{\mathsf{eval}(\mathtt{And}(e_1, e_2), env) = eval(e_2, env)}\text{(and-ok-long)}$$

(a, 2 points) Complete the "and-ok-short" rule for the non-error case when evaluating e_2 is not necessary (short circuiting):

$$\frac{\mathsf{eval}(e_1, env) = v_1,\ v_1 \neq \mathsf{error},\ v_1 = false}{\mathsf{eval}(\mathtt{And}(e_1, e_2), env) = \mathtt{false}}\text{(and-ok-short)}$$

P6 (4 points) For each of the following problems requires solution in Scala using a combination of map, filter, foldLeft, foldRight, and/or zip. No other API functions should be used.

(a, 2 points) Given `lst` of type `List[Int]`, compute sum of square of each element in the list. **Example:** `lst = List(1, 2, 3, 5)`, output: 39

Answer

lst. foldLeft (0)((acc, x) => acc + x*x)

(a, 2 points) Given two lists of equal size, `lst1` and `lst2`, both of type `List[Int]`, compute the list of pairs (x_i, y_i) such that x_i and y_i are the i'th elements of `lst1` and `lst2` (respectively), and $x_i > y_i$. Order of elements in the output list corresponds to that in the input list.
Example: `lst1 = List(1, 5, 2, 3, 8, 6)`, `lst2 = List(3, 4, 4, 7, 2, 1)`, output: `List((5,4), (8,2), (6,1))`

Answer

lst. zip .filter (p => p._1 > p._2)