Instructions

This quiz is 50 minutes long. It contains 12 pages (including this page) for a total of 100 points. The quiz is closed-book, closed-notes.

Please check your copy to make sure that it is complete before you start. Turn in all pages, together, when you finish. Write your name on the top of every page. Please write neatly. No credit will be given if we cannot read what you write. Good luck!

<table>
<thead>
<tr>
<th>Question Name</th>
<th>Page</th>
<th>Maximum Points</th>
<th>Points Given</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing</td>
<td>2</td>
<td>9</td>
<td></td>
</tr>
<tr>
<td>State Machines</td>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Regular Expressions and Grammars</td>
<td>4</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>Immutable Data Types</td>
<td>5</td>
<td>21</td>
<td></td>
</tr>
<tr>
<td>Visitors and Interpreters</td>
<td>7</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Specifications and Rep Invariants</td>
<td>9</td>
<td>20</td>
<td></td>
</tr>
</tbody>
</table>
Statistics

By Question:

Testing

State Machines

Reg Expressions

Immutable Data Types

Visitors and Interpreters

Specs and Rep Invariants
Statistics

Testing (9 Pts)

1) Which of the following best defines regression testing: 3 pts
   a) Changes should be tested against all inputs that elicited bugs in earlier versions of the code.
   b) Every component in your code should have an associated set of tests that exercises all the corner cases in its specification.
   c) Tests should be written before you write the code as a way of checking your understanding of the specification.
   d) When a new test exposes a bug, you should run it on all previous versions of the code until you find the version where the bug was introduced.

2) Match each role with the correct piece of the testing framework: 3 pts

   Driver
   Replaces system components outside the scope of the test
   Stub
   Runs the test
   Oracle
   Determines if the test passed or failed

3) Consider the following specification. 3 pts

   **Precondition:** The method takes an integer greater than 0 and less than 100
   **returns:** the biggest prime that is less than or equal than that number or -1 if there is no such prime.
   On incorrect inputs, an IncorrectInput exception is thrown.

   Following the methodology we studied in class, which 5 inputs would you test first:

   0, 100, 1, some prime, some non-prime

   Also accept 99 in place of 100
A fancy new car has an electronic ignition system that uses voice identification for security and a push-button to actually start and stop the engine. In order to start the car, I need to verbally identify myself to the car (a VOICEID event); after I do that, I can push the button to turn the engine on, and push the button again to push the engine off. If I push the button without identifying myself, nothing should happen. Also, if I don’t push the button within five minutes of VOICEID, a timeout event will reset the car to its initial state so I will have to verbally identify myself again to start the car; the same will happen if I open the door after identifying myself but before starting the engine. Finally, if the door is opened while the car is on, I need to verbally identify myself within five minutes, or another timeout event will turn off the car.

The system can be modeled with the following events: PUSH, VOICEID, TIMEOUT, OPENDOOR. The state machine below shows all the states that the car could be in. Label each of the transition edges with their appropriate events.
Regular Expressions and Grammars (18 pt)

Consider the following regular grammar:

\[
\begin{align*}
E & := C^* \ t \\
C & := x (A \mid B)^* x \\
A & := \text{NAME} \ \text{NUM} \ # \\
B & := \text{NUM} \ \text{NAME} \ # \\
\text{NAME} & := n (a \mid b \mid c \mid d \mid e)^* \\
\text{NUM} & := (1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \mid 0)^*
\end{align*}
\]

1) What are the terminals in this grammar? _____________________________ 3 pts

2) What are the non-terminals? ________________________________________ 3 pts

3) In the list below, circle those strings that are in the grammar: 6 pts

a) x x x x x x t  
   b) x naba 125# x x need 123# x t  
   c) x 1234# x x nace 33# x t  
   d) x nebe # x x nebe 334# x t  
   e) x x x nabe 55# 77 need# 23 naaaae# x t  
   f) x x x nabe # 77# naaaae# x t  

4) Which of the following pairs of regular expressions are equivalent: 6 pts

a) \((c \ b^* \ c)^*\) vs. \(c \ (b^* \ c)^*\)  
   b) \(ax(y^*by^*)^*\) vs. \(ax(y^*)^*(by^*)^*\)  
   c) \((a\mid b\mid c^*)(d?)\) vs. \((a\mid b\mid c^*)(c^* \mid d)\)  

name cannot be empty
Immutable Data Types (21 pts)

Consider the following two data-types.

Tree: Node(left:Tree, right:Tree, val:int) + EmptyTree
OList: Cons(val:int, next:OList) + EmptyList

The tree has the rep invariant that all values in the left subtree are smaller or equal to \( \text{val} \), and all values in the right subtree are larger or equal to \( \text{val} \). The ordered list OList has the rep invariant that all values in the next sublist are bigger than or equal to \( \text{val} \).

1) Write a function that given a Tree satisfying the rep invariant produces an OList satisfying the rep invariant. 15 pts

Sol 1: 
\[
\text{toList}(\text{Node}(\text{left}, \text{right}, \text{val})) = \\
\text{toList}(\text{left}) = \text{Cons} (\text{val}, \text{toList}(\text{right})) \\
\text{return} \ \text{Concat} (\text{toList}(\text{left}), \text{toList}(\text{right})) \\
\text{toList}(\text{EmptyTree}) = \text{EmptyList} \\
\text{toList}(\text{EmptyList}, \text{rest}) = \text{rest}
\]

Sol 2: 
\[
\text{toListB}(\text{Node}(\text{left}, \text{right}, \text{val}), \text{rest}) = \\
\text{toListB}(\text{right}, \text{rest}) \\
\text{return} \ \text{toListB}(\text{left}, \text{Cons}(\text{val}, \text{toListB}(\text{right}, \text{rest}))) \\
\text{toListB}(\text{EmptyTree}, \text{rest}) = \text{rest}
\]

\[
\text{toList}(\text{tree}) = \text{toListB}(\text{tree}, \text{EmptyList})
\]

Acceptable solutions would be written either in pseudocode as above or in terms of the interpreter or visitor patterns.
2: Below are three diagrams describing data-structures built from immutable data-types. For each of them, state whether or not it is actually possible to represent the given structure with the immutable data-type shown. 

6 pts

a) Tree with a list of the leaves:

\[
\text{LT} := \text{TNode(left: LTree, val: int, right: LTree)} + \text{LNode(val: int, next: LTree)} + \text{Empty}
\]

b) Cyclic list

\[
\text{CList} : \text{Cons(val: int, next: CList)} + \text{Empty}
\]

No; it has cycles

c) Two parent Directed Acyclic Graph

\[
\text{Dag} : \text{Node(p1: Dag, val: int, P2: Dag)} + \text{Empty}
\]
Statistics

Visitors and Interpreters: (12 pts)

1) Circle True or False 6 pts

T / F    The visitor pattern does not rely on dynamic dispatch
T / F    The interpreter pattern does not rely on dynamic dispatch
T / F    Adding a new function with the visitor pattern requires us to modify the variant classes

Consider the classes shown below to answer the questions that follow.

```java
public abstract class Expression {
    public abstract <T> T accept(ExpVisitor<T> ev);
}

public class NumExpr extends Expression {
    public final int val;
    public NumExpr(double val) {
        this.val = val;
    }
    public<T> T accept(ExpVisitor<T> ev){
        return ev.visit(this);
    }
}

public class PlusExpr extends Expression {
    public final Expression left;
    public final Expression right;
    public PlusExpr(Expression left, Expression right){
        this.left = left; this.right = right;
    }
    public<T> T accept(ExpVisitor<T> ev){
        return ev.visit(this);
    }
}

public abstract class ExpVisitor<T> {
    public abstract T visit(PlusExpr de);
    public abstract T visit(NumExpr de);
}
```
2) Which of the following implementations of expression evaluation is correct, and what is wrong with the other two?  

6 pts

a) ```java
class ExpEval extends ExpVisitor<Integer>{
    public Integer visit(PlusExpr de){
        Integer l = (Integer)de.left;
        Integer r = (Integer)de.right;
        return l + r;
    }
    public Integer visit(NumExpr de){
        return de.val;
    }
}
```

b) ```java
class ExpEval extends ExpVisitor<Integer>{
    public Integer visit(PlusExpr de){
        Integer l = de.left.accept(this);
        Integer r = de.right.accept(this);
        return l + r;
    }
    public Integer visit(NumExpr de){
        return de.val;
    }
}
```

c) ```java
class ExpEval extends ExpVisitor<Integer>{
    public Integer visit(PlusExpr de){
        Integer l = visit(de.left);
        Integer r = visit(de.right);
        return l + r;
    }
    public Integer visit(NumExpr de){
        return de.val;
    }
}
```
Specifications and Rep Invariants:(20 pts)

Consider the following class:

```java
/**
 * A class for clauses in a CNF representation of a logic formula.
 * A clause is an immutable set of literals that does not contain
 * a literal and its negation.
 */

public class Clause implements Iterable<Literal> {
    private final ImList<Literal> literals;
    /*
     * Rep invariant: literals is non-null but may be empty,
     * contains no duplicate literals, contains no literal and
     * its negation, contains no null elements.
     */
}
```

1) For each of the following methods, you should state whether: 12 pts
i) The method can break the invariant (Even when used correctly)
ii) The method can break the invariant when used incorrectly (i.e. when the precondition is violated).
iii) The method can never break the invariant, even when used incorrectly.
Keep in mind that if a method calls another method, it should not assume anything about its implementation other than what is stated in the specification.

a) ```java
/**
 * Requires: The same variable cannot appear in two
 * literals in the literals list.
 * The literals list cannot be null
 */

private Clause(ImList<Literal> literals) {
    this.literals = literals;
    checkRep();
}
```

b) ```java
/**
 * @return a clause containing a single literal or no literal if
 * parameter is null
 */

public Clause(Literal literal) {
    this(literal ==null?
            new Empty<Literal>():new Cons<Literal>(literal));
}
```
c) /**
 * Add a literal to this clause; if already
 * contains the literal's negation,
 * return null. Requires: l is non-null,
 * l is not in the clause.
 *
 * @return the new clause with the literal added, or null
 */
 public Clause add(Literal l) {
    if (l == null) return this;
    if (literals.contains(l))
        return this;
    if (literals.contains(l.getNegation()))
        return null;
    return new Clause(literals.addToFront(l));
}

d) /**
 * Merge this clause with another clause to obtain
 * a single clause with the literals of each. Returns
 * null if a literal appears as positive in one
 * clause and negative in the other. If a literal
 * appears in the same polarity in both
 * clauses, just appears once in the result.
 * Requires: c is non-null
 *
 * @return the merge of this clause and c
 */
 public Clause merge(Clause c) {
    Clause result = this;
    for (Literal l : c) {
        result = result.add(l); //add from c)
        if (result == null)
            return null;
    }
    return result;
}
2) Pick the right choice

If function F refines function G then its precondition must be weaker / stronger.

A post-condition that makes more promises about the return value of a function is weaker / stronger.

A specification that fails to describe all the possible scenarios is said to be incomplete / underdetermined.

Element of a specification that bounds the changes that a function may make on the environment is frame conditions / modifies.

END OF QUIZ