

Massachusetts Institute of Technology
6.005: Elements of Software Construction
Fall 2011
Quiz 1
October 14, 2011

Name: _____

Athena* User Name: _____

Instructions

This quiz is 50 minutes long. It contains 1 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!

Question Name	Page	Maximum Points	Points Given
Regexes & grammars	2	12	
State machines & testing	3	15	
Specifications	4	18	
Abstract data types (a)	5	5	
Abstract data types (b)	6	18	
Recursive data types (a)	7	15	
Recursive data types (b)	8	15	

*Athena is MIT's UNIX-based computing environment. OCW does not provide access to it.

Name: _____

Regular Expressions and Grammars

Consider the following grammar:

$F ::= B? E N^* M$

$B ::= >$

$E ::= : | ; | 8$

$N ::= - | ^$

$M ::= D | O | P$

(a) [8 pts] Which of the following strings could be legally and entirely recognized by the grammar? (**circle all that apply**)

☒ ;P

☐ :-^[

☒ >8O

☐ :^OD

☐ B:^D

☐ <3

☐ O^:

☒ :---D

(b) [4 pts] Write a regular expression for this grammar. You can use any operators from common regular expression syntax. Quoting or escaping is unnecessary if your meaning is clear.



Name: _____

State Machines and Testing

Before going to bed every night, Ben Bitdiddle turns on his alarm clock. It rings in the morning to wake him up, and he turns it off. Sometimes – not often – he wakes up early and turns the alarm off before it has a chance to ring.

(a) [12 pts] Draw a state machine for the alarm clock below. Label the states and the transitions, **using only the labels shown at the right**. Some labels may be used more than once, and some may be unused.

bed
clock
early
off
on
ring
ringing
turnoff
turnon

(b) [3 pts] Devise one or more test cases that together achieve **all-transitions** coverage for this state machine. Write each test case below, **not as Java code but as an event trace -- a sequence of event labels from the state machine above**.



Name: _____

Specifications

Write good specifications for the following methods. Do not change the parameter types or return type of the method, but you may change other parts of the method signature if you feel it's necessary to write a good spec.

/** Compute the square root of a number.



*/
public static int squareRoot(int x);

// IntSet represents a set of integers.
public class IntSet {
 ... // other fields and methods here
 /** Find the smallest element in the set.



*/
public int smallest();
}

/** Double every number.



*/
public static List<Integer> doubleAll(List<Integer> lst);

Name: _____

Abstract Data Types

Consider the following code.

```
1 /** Text is an immutable data type representing English text. */
2 public class Text {
3     private final String text;
4     private final String[] words;
5
6     // Rep invariant:
7     //   text != null; words != null;
8     //   concatenation of words (words[0]+words[1]+...+words[words.length-1])
9     //   is the same as text with spaces and punctuation removed
10    // Abstraction function:
11    //   represents the English text in the string variable text
12
13    /**
14     * Make a Text object.
15     * @param sentence a sentence in English. Requires sentence != null.
16     */
17    public Text(String sentence) {
18        this.text = sentence;
19        this.words = sentence.split(" ");
20    }
21
22    /** @return the words in the sentence */
23    public String[] getWords() {
24        return words;
25    }
26
27    /** @return the sentence as a string */
28    public String toString() {
29        return text;
30    }
31
32    /** concatenates this Text to that Text. Requires that != null. */
33    public Text add(Text that) {
34        return new Text(this.text + that.text);
35    }
36
37    /** @return true if and only if the word w is in the sentence.
38     * Requires w != null */
39    public boolean contains(String w) {
40        for (String v : words) { if (w.equals(v)) { return true; } }
41        return false;
42    }
43 }
```



(a) [5 pts] For each constructor and method above, write in the box next to it:

C for creator
P for producer
O for observer
M for mutator

(b) [18 pts] The code above was code-reviewed, producing the comments below. Circle AGREE or DISAGREE depending on whether the comment is correct or incorrect, and add your own **one-sentence comment** explaining your answer. The right explanation is worth more than the right circle.

line 23: Rep exposure threatens the rep invariant.




AGREE


DISAGREE


Name: _____

line 23 reply: No it doesn't, words is a final variable.  AGREE **DISAGREE**

line 18: Constructor doesn't establish the rep invariant.  **AGREE** DISAGREE

line 17: Rep exposure! Need to make a copy of text before storing it in your rep.  **AGREE** DISAGREE

line 33: add() changes this.text, you shouldn't do that in an immutable type.  AGREE **DISAGREE**

line 39: the compiler's static checking will throw an exception here if `w == null`  AGREE **DISAGREE**

Name: _____

Recursive Data Types

In this problem you will implement a recursive data type representing sets of words and intersections of those sets. The datatype definition is:

$\text{WordSet} = \text{Base}(t:\text{Text}) + \text{Intersect}(\text{left}:\text{WordSet}, \text{right}:\text{WordSet})$

(a) [15pts] Write Java code below that implements this datatype. **Include the reps (fields) and creators (constructors), but no other methods.** **You don't need to write specs for your methods in this problem.** Note that you will need to use the Text datatype defined in the previous problem; assume that all its implementation bugs have been fixed so that it behaves according to its spec.

Name: _____

(b) [15pts] Define a function over your datatype:

```
contains: WordSet ws, String w => boolean
// requires: w is a word, with no spaces or punctuation
// returns: true if and only if w is an element of the
//          set of words represented by ws
```

Implement the function using the **Interpreter pattern**. Write your Java code below. Again, you don't need to write specs for this problem. You also don't need to repeat the code you wrote above, but if you need to insert methods into classes you wrote above, just put a brief outline around it with the name of the class, e.g.

```
ClassName {
    public void myNewMethod() {
        ...
    }
}
```

END OF QUIZ

MIT OpenCourseWare
<http://ocw.mit.edu>

6.005 Elements of Software Construction
Fall 2011

For information about citing these materials or our Terms of Use, visit: <http://ocw.mit.edu/terms>.