CSE 413 Final Exam

December 13, 2012

Name _____

The exam is closed book, closed notes, no electronic devices, signal flags, tin-can telephones, or other signaling or communications apparatus.

Style and indenting matter, within limits. We're not overly picky about details, but we do need to be able to follow your code and understand it.

Please wait to turn the page until everyone has their exam and you have been told to begin. If you have questions during the exam, raise your hand and someone will come to you. ***Don't*** leave your seat.

Advice: The solutions to many of the problems are short. Don't be alarmed if there is a lot more room on the page than you actually need for your answer.

More gratuitous advice: Be sure to get to all the questions. If you find you are spending a lot of time on a question, move on and try other ones, then come back to the question that was taking the time.

1	/ 10
2	/ 12
3	/ 6
4	/ 8
5	/ 12
6	/ 10
7	/ 14
8	/ 18
9	/ 6
10	/ 4
Total	/ 100

Question 1. (10 points) Regular expressions I. Describe the set of strings generated by each of the following regular expressions. For full credit, give a description of the sets like "all sets of strings made up of a's, b's, and c's with 4 a's and at least as many b's as a's". Don't just transcribe the expressions from regular expression notation into English.

(a) (ab)*(bc)*

(b) ((a|b)+ c)*

Question 2. (12 points) Regular expressions II. Write a regular expression or set of regular expressions that generate the following sets of strings.

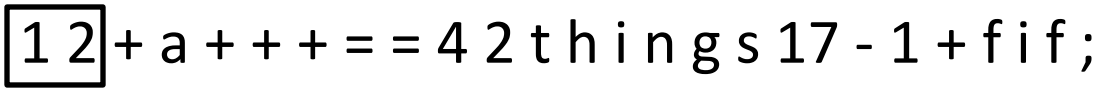
Fine print: You may use basic regular expressions (sequences rs, choice r|s, and repetition r* and parentheses for subexpressions). You may also use + (one or more) and ? (zero or one), and character classes like [ax-z] and [^abc], but you may not use additional regular expression operators that might be found in various programming languages and software tools. You also may use named abbreviations like "vowels = [aeiou]" if these help.

(a) All non-empty strings of 0's and 1's such that the number of 1's is even if there are any 1's in the string. (There are no other restrictions on the positions of the 1's in the string other than the total number of 1's needs to be even.).

(b) Ruby identifiers. A Ruby identifier is a sequence of letters, digits, and underscores. An ordinary identifier may not begin with a digit. Identifiers may be preceded by `$`, `@`, or `@@` to indicate global, instance, or class variables. The character following `$`, `@`, or `@@` may not be a digit. An identifier may also end with one of the characters `?`, `!`, or `=`.

Question 3. (6 points) The C family of languages (including Java) has prefix and binary + and − operators, as well as increment and decrement operators ++ and −− that can appear either before or after an expression or sub-expression. The languages also include the other usual kinds of tokens, including identifiers, reserved words, and numbers.

How would a scanner for a language like Java divide the following input into tokens? Draw a box around each set of characters that make up a token. You should assume that these characters are adjacent to each other. We have added some space to make it easier to draw boxes, and the first box is drawn for you. (Remember that we're only asking about how the scanner would divide this sequence of characters into tokens, not whether they make any sense as part of a C or Java program.)



Question 4. (8 points) Give a context-free grammar that generates all strings of a's and b's that are not empty and are *palindromes* – that is, the string reads the same backwards as forwards. A few strings in this set are a, b, aa, bb, aaa, bbb, aba, bab, aaaa, abba, etc.

Question 5. (12 points) Consider the following gammar

expr ::= `a` | `a` *subs*
subs ::= `[` *expr* `]` | `[` *expr* `]` *subs*

(a) (4 points) What are the terminals and non-terminals of this grammar?

Terminals:

Non-terminals:

(b) (5 points) Draw the parse tree for `a[a][a]`

(c) (3 points) Describe in English the set of strings generated by this grammar.

Question 6. (10 points) In most of the languages descended from C, the syntax for function calls uses parentheses around the argument list, while array subscripts are indicated with brackets around each individual subscript. In some other languages like Fortran parentheses are used for both function calls and array subscripts, and subscripts for multiple-dimension arrays are given in a single list of expressions inside a single pair of parentheses. For instance, the C statement `ans=b[i][j]*sin(theta)` would be written as `ans=b(i,j)*sin(theta)` in Fortran.

Here is part of an expression grammar for Fortran that includes function calls and array references:

exp ::= *term* | *exp* + *term* | *exp* – *term*
term ::= *factor* | *term* * *factor* | *term* / *factor*
factor ::= *id* | *number* | (*exp*) | *funcall* | *arrayref*
funcall ::= *id* (*explist*) | *id* ( )
arrayref ::= *id* (*explist*)
explist ::= *exp* | *explist* , *exp*

(a) (7 points) Show that this grammar is ambiguous.

(b) (3 points) A compiler for Fortran has to be able to analyze and understand programs in spite of this ambiguity. Suggest one way that this might be done (and the solution might involve other parts of the compiler, not just the parser).

Question 7. (14 points) The local library would like your help to write a Ruby program to print information about overdue books. The input to the program is a simple text file. The first line is the current date in the format *yyyymmdd*, where *yyyy* is the year, *mm* is the month, and *dd* is the date. For example, today's date, Dec. 13, 2012, would be written 20121213. The rest of the file contains information about checked-out books. There are three lines in the file for each book giving the book title, the borrower's name, and the due date. Here's an example with three books, two of which are overdue (due before the date given at the beginning of the input):

```
20121213
50 Shades of Ruby
A. Turing
19361112
Naughty and Nice
S. Clause
20121225
Bored of the Rings
P. Jackson
20121128
```

Write a Ruby program that reads data from standard input in this format and prints the following:

- For each overdue book, print the due date, borrower's name, and book title on a single line, in that order with a comma between the name and title.
- After reading all of the input, print a list of names of the borrowers who have one or more overdue books. The list may be in any order. The name of each borrower in the list should be printed only once even if they have several overdue books, and each name should be on a separate line.

Here is the output for the above sample input (the order of the last two lines could also be reversed):

```
19361112 A. Turing, 50 Shades of Ruby
20121128 P. Jackson, Bored of the Rings
A. Turing
P. Jackson
```

For full credit you should use Ruby iterators like `each` to process the contents of any containers like arrays or hashes. Recall that if `h` is a hash, you can iterate through its key/value pairs with

```
h.each {|key, value| ... }
```

Hint: The date format was chosen so that dates could be compared as strings without having to convert them to numbers. But you do not have to do it that way if you don't want.

Write your code on the next page. If you find it helpful, you can remove this page from the exam for reference while you are working.

Question 7. (cont.) Write your code here:

Question 8. (18 points) The programming languages we've seen this quarter write arithmetic expressions in two quite different ways. Racket uses *prefix* notation, where each operator appears first followed by its operands. Ruby uses the more traditional *infix* notation, where binary operators are written between their operands. For example, the infix expression `(2+3)*(9-6)` is written in prefix notation as `(* (+ 2 3) (- 9 6))`.

There is another notation for arithmetic expressions called *postfix* notation, where each operator follows its operands, i.e., the infix expression *exp1 op exp2* is written in postfix as *exp1 exp2 op*. For example, the above expression would be written in postfix as `2 3 + 9 6 - *`. Interestingly enough, postfix expressions do not require parentheses to indicate either associativity or precedence. Expressions are evaluated from left to right, and when an operator (+, *, etc.) is encountered, the two operands immediately to its left are combined using the operator, and the result replaces all three.

For this problem write a Ruby parser to translate an infix expression to postfix. The parser should read input tokens from a scanner, as in our project, and *print* the postfix version of the expression – it should ***not*** evaluate it. Here is the input expression grammar.

exp ::= *term* | *exp* + *term* | *exp* – *term*
term ::= *factor* | *term* * *factor* | *term* / *factor*
factor ::= *id* | *number* | (*exp*)

Here are some more examples of infix to postfix translations. (Remember that the rule is that *exp1 op exp2* is translated to *exp1 exp2 op*, even when *exp1* or *exp2* are themselves complex expressions.)

Infix	Postfix	Comment
`2+3+4`	`2 3 + 4 +`	Same as `(2+3)+4`
`2+(3+4)`	`2 3 4 + +`	
`(a+b)*c`	`a b + c *`	
`a*(b+c)`	`a b c + *`	
`(5-y)*(x+3)`	`5 y - x 3 + *`	

You should assume the following as you write your solution:

- There is a method `next_token` that returns a new `Token` object with the next input token each time it is called. There is a global variable `current_token` that contains the first `Token` in the expression when your parser procedure for `exp` is first called.
- If `t` is a `Token` object, `t.kind` returns the token kind, which is either "ID", "Number", or one of the terminal symbols `"+"`, `"-"`, `"*"`, `"/"`, `"("`, or `")"`. If `t.kind` returns "ID" or "Number", then `t.value` returns the actual identifier or number.
- There are no end-of-file or end-of-line tokens. The parser should stop once it has processed a complete expression (i.e., printed the postfix version of a single expression).

Hint: All that is required is to print out the tokens in the correct final order.

Write your answer on the next page(s). You may detach this page for reference if you wish.

Question 8. (cont.) Write your Ruby code for the infix-to-postfix parser/translator here.

(more room on the next page if you need it)

Question 8. (cont.) (Additional space for your answer if you need it.)

A couple of short questions to wrap up. Please keep your answers brief and to the point (and legible!!). Your readers thank you.

Question 9. (6 points) Java has interfaces as well as classes, and a class can implement multiple interfaces. Neither Ruby nor C++ have interfaces, but for quite different reasons.

(a) Why are there no Java-style interfaces in Ruby? Give a technical reason why interfaces would not be appropriate and/or needed in this language.

(b) Why are there no Java-style interfaces in C++? Again, give a technical reason why interfaces would not be appropriate and/or needed in this language.

Question 10. (4 points) One of the classic strategies for automatic memory management is *reference counting*. Yet reference counting is not used as the normal strategy for reclaiming memory in languages like Java, Ruby, and Racket, all of which use automatic garbage collectors. What is the main technical reason that reference counting cannot be used if we want to reclaim all unused memory?

Best wishes for the holidays and the New Year!!