# CSE 413 Final Exam

## December 10, 2014

**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 | / 16 |
| 2 | / 10 |
| 3 | / 14 |
| 4 | / 14 |
| 5 | / 8 |
| 6 | / 12 |
| 7 | / 14 |
| 8 | / 12 |
| Total | / 100 |

**Question 1.** (16 points) Regular expressions I. For each of the following, (i) describe the set of strings generated by the given regular expression, and (ii) circle the regular expression(s), if any, that generate exactly the same set of strings. For full credit, the descriptions should be things 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. In the lists of possibly equivalent regular expressions, there may be a single expression that generates the same strings, more than one, or none.

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

(i) Description:

(ii) Circle the regular expression or expressions that generate the same sets of strings as the one above, if any:

(a|b)*b(a)+ a*b*a+ (a|b)*aa* (a*b*)*aa+

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

(i) Description:

(ii) Circle the regular expression or expressions that generate the same sets of strings as the one above, if any:

[a-c]* [a]*[bc]* [abc]+ (a|b|c)*(b|ε)

**Question 2.** (10 points) Regular expressions II. Almost all computers have a file system with directories and ordinary files. Every file is contained in a directory, and directories can contain other directories, nested to any level desired.

We are designing a file system where file and directory names are specified as follows. A name can start with an optional leading slash ('/'). Then there is a set of zero or more directory names separated by slashes. Directory names consist of one or more upper- or lower-case letters. Following the directory names, if any, is the file name. File names include an identifier with one or more upper- or lower-case letters, followed by an optional one- to three-letter file extension. If the file extension is present, it is separated from the file name by a period ('.').

Examples:
/long/directory/path/with/a/file.txt
simpleFileNameWithNoExtension
simpleFileNameWithExtension.CC
some/subDirectory/lib.a
/x.yz

Invalid names:

| | |
|---|---|
| a/b//c | -- adjacent '/' characters |
| a/b/ | -- no file name after final '/' |
| foo.docx | -- more than three letters following '.' |
| file1 | -- digit in file name |
| file. | -- no letters following '.' |

Write a regular expression or set of regular expressions that generate this set of names.

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

**Question 3.** (14 points) Consider the following grammar:

*S* ::= ( *L* ) | a
*L* ::= *L* , *S* | *S*

(a) (4 points) What are the terminal and non-terminal symbols in this grammar?

Terminals:

Non-terminals:

(b) (6 points) Draw the parse tree for (a, (a, a))

(c) (4 points) Describe in English the set of strings generated by this grammar.

**Question 4.** (14 points) Ruby Interlude. Suppose we have lines of text stored in an array of strings. Each line of text is a sequence of words, and all punctuation has been removed. For example, if we had the text of Shakespeare's plays in this format, the first part of Hamlet's soliloquy could be stored as:

```
["To be or not to be that is the question",
 "Whether tis nobler in the mind to suffer",
 "The slings and arrows of outrageous fortune",
 "Or to take arms against a sea of troubles"]
```

Write a ruby method `avg_wordlength` that has a single parameter that is an array of strings, as in the example above. The method should return a number giving the average number of characters in the words contained in the strings. For example, if the parameter is the array

```
["hello Hamlet", "how are you Hamlet"]
```

then function `avg_wordlength` should return 4.33333 (which is (5 + 6 + 3 + 3 + 3 + 6) / 6). If a word occurs multiple times, it is counted each time.

For full credit you should use Ruby iterators like `each` to process the contents of any containers like arrays or hashes.

A couple of useful facts about strings:

- If `s` is a string, `s.length` is the number of characters in it.
- The string split method will return an array of the words in a string. Example: `"one two three".split` returns `["one", "two", "three"]`.

Write your code below. There is additional space on the next page if you need it.

**Question 4. (cont.)** Additional space for your `avg_wordlength` method, if needed.

We would like to extend the calculator from the last two assignments to handle Boolean expressions. The next several questions explore these changes.

The basic idea is that we want to add two operators to the language: `&&` and `||`. The `&&` operator is logical "and"; while `||` is logical "or". Variables and constants are Ruby numbers as before. The number 0 (and therefore 0.0) is considered to be false, and all other values are considered to be true (the same convention used in languages like C). When we evaluate `p&&q` or `p||q`, the result is 1 if the formula evaluates to true or 0 if the formula evaluates to false. So, for example, `1&&17` is true and results in 1; `1&&0` is false and results in 0; `0||3` is true and results in 1. Unlike in Java and other languages, both operands of `&&` and `||` are always evaluated.

Boolean operators have their usual precedence: `&&` has higher precedence than `||`, and both have lower precedence than other arithmetic operators. So the expression `3+p&&2||q` means `((3+p)&&2)||q`. Boolean operators are left associative, so that `x&&y&&z` means `(x&&y)&&z`.

The rest of the calculator language remains the same, with integer constants and variables; expressions involving +, -, *, /, **, and parentheses; the `sqrt` function; assignment statements *variable*=*expression*; and the keywords `unset`, `list`, `quit`, and `exit`. Most of this information about the existing calculator language is not needed to answer the following questions.

**Question 5.** (8 points) Suppose that we have added the `&&` and `||` Boolean operators to the implementation and then the scanner reads the following input:

```
xvii = 17

unset a&&b=10***2||3

x+=3+-42 + foolist
```

Show how the calculator scanner would divide those input characters into tokens by drawing a box around each sequence of characters that make up a single token. Boxes on the first line are drawn for you. You do not need to show any "end of line" or "end of file" tokens. (Remember that we're only asking about how the scanner would divide the input characters into tokens, not whether the resulting token sequence makes any sense or is a legal calculator program.)

**Question 6.** (12 points) As part of adding these operations to our calculator, we're experimenting with grammar rules for simplified Boolean expressions involving only integer constant values and Boolean operators. Here is an attempt by one of the new interns:

*bexp* ::= *int* | *bexp* && *bexp* | *bexp* || *bexp*

(Recall that a single '|' identifies choices in a grammar rule – it is not an operator like && or ||.)

(a) (6 points) Show that this grammar is ambiguous.

(b) (6 points) Give a different grammar for this language of Boolean expressions, still involving only *int* values and the && and || operators, that fixes the problem(s). Your grammar should be unambiguous, should give && higher precedence than ||, and should ensure that these operators are left-associative.

**Question 7.** (14 points) While work is going on to fix the grammar we also want to write a prototype implementation of the parser/interpreter method for Boolean `&&` expressions. For this question, complete the parser/interpreter method on the next page so it will parse and evaluate Boolean expressions consisting of a single expression *exp* or multiple expressions connected by `&&` operators. In other words, the method should parse and evaluate expressions of the form

*and_exp* ::= *exp* { `&&` *exp* }

where the braces indicate zero or more additional "`&&` *exp*" strings can follow the initial *exp.*

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 *and_exp* when your parser procedure for *and_exp* is first called and this variable must be updated by your method to contain the first token following the *and_exp* when your method returns its value.

- 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. The token kind for the new logical "and" operator is the string "`&&`".

- There is a method `exp()` that will parse and evaluate the next *exp* in the input and return its value, and will advance the global `$current_token` variable to the first token following the *exp* that it evaluates.

Remember that the `&&` operator returns either the value 0 or 1, but, when it evaluates its operands, it treats 0 as false and any non-zero value as true.

Hint: The answer is (probably) not as complicated as the question.

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

(answer on next page, please)

**Question 7. (cont.)** Write your Ruby code to parse and evaluate logical "and" expressions below. A comment and the first line of the function heading are supplied for you:

```
# parse and evaluate:  exp { && exp } …

def and_exp
```

**Question 8.** (12 points) Memory management (and a little Java!!). Suppose we define the following class to represent nodes in a linked list of integer values:

```
public class Node {
  public int val;
  public Node next;

  // constructor for convenience
  public Node(int n, Node next) {
    this.val = n; this.next = next;
  }
}
```

Now suppose we execute a main method with the following statements:

```
  public static void main(String[] args) {
    Node p = new Node(11,null);
    Node q = new Node(12,null);
    Node r = new Node(13,null);
    Node s = new Node(14,q);
    p.next = q;
    q.next = s;
    s.next = q;
    s = r;
}
```

Assume that we have a Java implementation that uses *reference counting* (*not* garbage collection) to manage memory. On the next page, draw a diagram showing the contents of memory after these statements have executed. The diagram should show each node, the integer value in the node, and an arrow in the "next" field pointing to the appropriate node, if any. Next to each node, write its reference count. Then answer the remaining questions at the bottom of the next page.

(Use the space below to work out the details of your diagram before you make a clean copy on the next page. You can remove this page for reference if you wish.)

**Question 8. (cont.)** (a) (6 points) Draw your diagram of memory showing nodes, their contents, and their reference counts as created by the `main` method on the previous page.

(b) (6 points) Now suppose we add the statements `p=null; q=null;` at the end of the main method. How does this affect the memory diagram and reference counts you drew in part (a), and does the implementation properly free any memory that is no longer reachable? If there are memory leaks or other memory management problems, identify the problem(s) and explain what went wrong and why. Feel free to draw an updated copy of your diagram from part (a) if that is helpful, but this is not required.

*Have a great winter break and best wishes for the new year!*
*The CSE 413 staff*