Primary Exam for CSE 450 (2016)

Answer the questions in the spaces provided on the page. If you run out of room for an answer, continue on the back of the page.

- DO NOT OPEN THE EXAM UNTIL TOLD TO DO SO
- You only need to answer 5 of the 6 questions.
- On one of the questions make a large slash through it, which indicates that it should not be graded.
- On every page (including the first and last page), write your first and last name, before answering the question. Unnamed pages may be lost.
- If you start to answer a question and then change your mind, please cross out the attempt and write *DO NOT GRADE* across it.

https://xkcd.com/303/
Question 1: Regular Expression Matching.............................................4 points

For each of the following regular expressions, fill in the bubble of the strings which can be generated?

(a) (2 points) \[^aeiou\][aeiou]+[^aeiou]?

- √ dog
- √ cat
- √ Jo
-steam
- √ bean
- √ Doe
-Trump
-Hillary

Grade Breakdown:
- 1 point if only one wrong answer
- 1/2 point if two wrong answers
- 0 points if more than two wrong answers

(b) (2 points) ((ab+)*c)+

- √ abc
- √ c
- √ ababc
-abcd
- √ abcabbabc
- √ abccc
- √ ccababc
-ccabcabb

Grade Breakdown:
- 1 point if only one wrong answer
- 1/2 point if two wrong answers
- 0 points if more than two wrong answers

Points earned: __________ out of a possible 4 points
Question 2: Tube Intermediate Code.........................................................4 points
Given the source code, fill in the blanks in the Tube Intermediate Code generated by a compiler.

```plaintext
val x = 4;
if (x > 1) {
    while (x) {
        val y = 3; x = y;
        break;
    }
} else print(x);
```

<table>
<thead>
<tr>
<th>command</th>
<th>arg1</th>
<th>arg2</th>
<th>arg3</th>
</tr>
</thead>
<tbody>
<tr>
<td>val_copy</td>
<td>4</td>
<td>s3</td>
<td></td>
</tr>
<tr>
<td>val_copy</td>
<td>s3</td>
<td>s1</td>
<td></td>
</tr>
<tr>
<td>val_copy</td>
<td>1</td>
<td>s4</td>
<td></td>
</tr>
<tr>
<td>test_gtr</td>
<td>s1</td>
<td>s4</td>
<td>s5</td>
</tr>
<tr>
<td>jump_if_0</td>
<td>s5</td>
<td>if_else_0</td>
<td></td>
</tr>
<tr>
<td>while_start_2:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jump_if_0</td>
<td>s1</td>
<td>while_end_3</td>
<td></td>
</tr>
<tr>
<td>val_copy</td>
<td>3</td>
<td>s6</td>
<td></td>
</tr>
<tr>
<td>val_copy</td>
<td>s6</td>
<td>s2</td>
<td></td>
</tr>
<tr>
<td>val_copy</td>
<td>s2</td>
<td>s1</td>
<td></td>
</tr>
<tr>
<td>jump</td>
<td>while_end_3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>jump</td>
<td>while_start_2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>while_end_3:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>jump</td>
<td>if_end_1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if_else_0:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>out_val</td>
<td>s1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>out_char</td>
<td>'n'</td>
<td></td>
<td></td>
</tr>
<tr>
<td>if_end_1:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 1 point for declaration and assignment
- 1 point for testing and jumping
- 1 point for break
- 1 point for printing

Points earned: __________ out of a possible 4 points
Question 3: Arrays and Memory ................................................................. 4 points

Note: Both parts (a) and (b) read and write to the same memory at the bottom of the page.

(a) (1 point) For the memory representation below, write in the row "New Value" any values that would change from executing the Tube Intermediate Code instruction. Memory location 0 stores the start of the free memory.

```
ar_set_size a3 s1
ar_get_idx a3 s2 s4
add s1 s4 s1
```

• 1/2 point for entirely correct ar_set_size
• 1/2 point for the get index and add memory changes

(b) (2 points) For the same memory representation below, write in the row "New Value" any values that would change from executing the series of Tube Code Assembly instructions.

```
load 2 regA
load 5 regB
add 1 regA regA
val_copy regA regD
store regB regA
load 6 regC
mem_copy regC 6
test_less regA regC regC
```

• 1 point for each of the 2 memory locations that should change
• 1/2 point deducted for any unrelated memory changes

(c) (1 point) Final value for registers after the above TCA instructions:

```
regA: 2
regB: 'J'
regC: 1
regD: 2
```

• 1/2 point if 3/4 registers were correct
• 1 point if 4/4 registers were correct

Points earned: ___________ out of a possible 4 points
<table>
<thead>
<tr>
<th>Location</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old Value</td>
<td>106</td>
<td>3</td>
<td>1</td>
<td>102</td>
<td>'X'</td>
<td>'J'</td>
<td>7</td>
<td>11</td>
<td>'\n'</td>
</tr>
<tr>
<td>New Value</td>
<td>110</td>
<td>7</td>
<td>'J'</td>
<td>106</td>
<td>4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location</th>
<th>100</th>
<th>101</th>
<th>102</th>
<th>103</th>
<th>104</th>
<th>105</th>
<th>106</th>
<th>107</th>
<th>108</th>
<th>109</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old Value</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>New Value</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Points earned: __________ out of a possible 0 points
Question 4: Tube Code Assembly

For the Tube Intermediate Code below, convert each line to Tube Code Assembly.

(a) (1 point) add 5 s3 s4

Solution:
val_copy 5 regA
load 3 regB
add regA regB regC
store regC 4

(b) (1 point) jump_if_0 s6 if_end_1

Solution:
load 6 regA
jump_if_0 regA if_end_1

out_char 'J'

Solution:
out_char 'J'

(c) (1 point) ar_get_size a10 s11

Solution:
load 10 regA
mem_copy regA 11

(d) (1 point) ar_get_idx a12 s13 s14

Solution:
load 12 regA
load 13 regB
add 1 regA regA
add regA regB regA
mem_copy regA 14

Points earned: __________ out of a possible 4 points
Question 5: Abstract Syntax Tree. 4 points
Below is an Abstract Syntax Tree of a Tubular expression statement.

(a) (2 points) What is the Tubular program that could have generated such a tree?

Solution:
val x = 5;
if (x) {
    val y = x + 3;
    print(y);
} else {
    val y = 7;
    print(y);
}

Accepted valid Python code (provided correct indentation). One point was removed for missing if-else construct, or wrong parentheses. No points were taken off for missing semicolons.

(b) (1 point) What is the output from the program? (You can ignore the newline character.)

8

(c) (1 point) Which of the variables are global variables (variables of scope level 0)?

x

Points earned: __________ out of a possible 4 points
Question 6: New Context Free Grammar................................................. 4 points

You want to implement a new dictionary literal in Tubular that generates a dictionary based on a comma-separated entries (key-value pairs). The keys must be string literals. And the values (separated from the key by a colon) may be either a dictionary literal, string literal, val literal, or char literal. Dictionaries are contained within "{" and "}".
Examples of some legal dictionaries are:

```
{"josh": 7, "doug": 'j'}
{}
{"class": "compilers"}
{"pets": {"racetrack": "ferret", "crashdown": '!'}}
```

Empty dictionaries are okay!
One entry is okay
Dictionary literals can be values

Examples of ILLEGAL dictionaries are:

```
{"c": "hi"}
{"j": "o", "s": "h",}
```

Key must be a string
Cannot have a trailing (or leading) comma!

Write a set of YACC-style syntactic rules that capture the grammar for `dict_literal`.
These are the only tokens you may use: `STRING_LITERAL, CHAR_LITERAL, VAL_LITERAL,'{', '}', ':', ':' and ','.

**Solution:**

```
dict_literal : '{' body '}'
body :
   body : entries
entries : entry
   entries : entry ',' entries
entry : STRING_LITERAL ':' value
value : dict_literal
value : STRING_LITERAL
value : CHAR_LITERAL
value : VAL_LITERAL
```

Full credit required all the above rules. One point removed for each missing rule.

Points earned: __________ out of a possible 4 points
Question 7: Extra Credit: DFA .............................................. 1 points

(a) Convert the above DFA to a table. Use a slash to denote a halt on failure.

<table>
<thead>
<tr>
<th>State</th>
<th>a</th>
<th>b</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>/</td>
<td>E</td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>/</td>
<td>/</td>
</tr>
<tr>
<td>C</td>
<td>D</td>
<td>A</td>
<td>/</td>
</tr>
<tr>
<td>D</td>
<td>E</td>
<td>A</td>
<td>/</td>
</tr>
<tr>
<td>E</td>
<td>A</td>
<td>/</td>
<td>D</td>
</tr>
</tbody>
</table>

(b) Fill in the bubbles of the strings that the DFA accepts.

√ aaaa
√ baab
√ bccaa
○ abca
○ baabac
√ bccaaacca
○ DAEDA
○ baac

Points earned: __________ out of a possible 1 points
If you have finished early, feel free to bring your exam to an instructor.
Or you can draw a picture of your favorite Pokémon.
Or you can write a haiku about your love of Abstract Syntax Trees.

<table>
<thead>
<tr>
<th>Normal Questions:</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regular Expression Matching</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Tube Intermediate Code</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Arrays and Memory</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Tube Code Assembly</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Abstract Syntax Tree</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>New Context Free Grammar</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>20</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Extra Credit Questions:</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extra Credit: DFA</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td><strong>Total:</strong></td>
<td><strong>1</strong></td>
<td></td>
</tr>
</tbody>
</table>

Points earned: __________ out of a possible 0 points
Tube Intermediate Code Overview

**Arguments**
VAL: This argument uses a value; it can be a literal integer, a literal char, a label, or a scalar variable.
SCL: This argument must be a scalar variable (which will be written to)
ARR: This argument must be an array variable

**Instructions**
- `add [VAL: num1] [VAL: num2] [SCL: result]`
- `sub [VAL: num1] [VAL: num2] [SCL: result]`
- `mul [VAL: num1] [VAL: num2] [SCL: result]`
- `div [VAL: num1] [VAL: num2] [SCL: result]`

Apply the given math operation on `num1` and `num2`, and place the answer in `result`.

- `val_copy [VAL: from] [SCL: to]` Copy the value `from` into the scalar variable `to`.
- `out_val [VAL: num]` Output the number represented by the argument.
- `out_char [VAL: char]` Output the character represented by the argument.

**test_** [VAL: num1] [VAL: num2] [SCL: result]
Options are: `test_less, test_gtr, test_equ, test_nequ, test_lte, or test_gte`

Compare `num1` and `num2`. Set `result` argument to 0 or 1 based on if condition is false or true.

- `jump [VAL: line]` Move the instruction pointer to `line`.
- `jump_if_0 [VAL: test] [VAL: line]` Move the instruction pointer to `line` if `test` is equal to zero.
- `jump_if_n0 [VAL: test] [VAL: line]` Move the instruction pointer to `line` if `test` is NOT equal to zero.

- `random [VAL: max] [SCL: result]` Generate random number 0 to `(max - 1)` and store in `result`.

**ar_get_idx [ARR: array] [VAL: index] [SCL: result]**
Look up `index` in `array` and store its value as `result`.

**ar_set_idx [ARR: array] [VAL: index] [VAL: value]**
Loop up `index` in `array` and set its value to `value`.

**ar_get_size [ARR: array] [SCL: result]**
Look up the size of `array` and store it in `result`.

**ar_set_size [ARR: array] [VAL: new_size]**
Resize `array` to `new_size`, copying over those elements in common.

**ar_copy [ARR: array1] [ARR: array2]**
Copy the contents and size of `array1` into `array2`.

**Labels**
A label is a string of alphanumeric characters, beginning with a letter that is used to reference a line number elsewhere in the code. When a label is created, it must be placed at the beginning of a line of code and it must end with a colon (`:`). A label will typically be used to indicate the end point in a jump command.

TubeCode Assembly Overview

TubeCode Assembly is very similar to the intermediate code, with a handful of changes

- The `array_*` instructions are not available.
- Scalar variables are not available, but eight registers (regA through regH) take their place.
- Three new instructions are available that allow you to interact with memory. They are:

  - `load [VAL: from] [REG: to]` Load the value in memory position `from` into register `to`.
  - `store [REG: from] [VAL: to]` Store the value in register `from` into memory position `to`.
  - `mem_copy [VAL: from] [VAL: to]` Copy the value in memory position `from` into memory position `to`.