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 should answer only 5 of the 6 questions.
- On one of the questions make a large slash through it, which indicates that it should not be graded.
- If you start to answer a question and then change your mind, please cross out the attempt and write *DO NOT GRADE* across it.
- Have your ID ready when you turn in your exam.
Question 1: Regular Expression Matching ................................................. 40 points

For each of the following regular expressions, indicate which of the strings can be generated?

(a) (20 points) \(1^*01^*0\)
   - 0
   - 00
   - 1010
   - 000010
   - 10
   - 01110
   - 1100
   - 1

(b) (20 points) \((ab^*)+c?\)
   - ac
   - ababab
   - aaabbb
   - c
   - abcd
   - abbaabbcc
   - bbc
   - ab+c
Question 2: Construct Regular Expression .................................................. 40 points

(a) (20 points) Design a regular expression that will identify all literal integers where
the digits are in numerical order. Allow the empty string.
For example, 01457889 is allowed but 8953 is not.

(b) (20 points) Design a regular expression that will identify an integer value between
0 and 255. Your expression should identify all of these values with no extra leading
zeros and no additional characters.
Good examples:
- 200
- 36
- 2
- 0
- 255

Bad examples:
- -1
- 3.14
- 450
- 007
- 256
Question 3: Arrays and Memory ........................................ 40 points
Note: Both parts (a) and (b) read and write to the same memory at the bottom of the page.

(a) (20 points) For the memory representation below, write in the row “New Value” any values that would change from executing the **LMAOcode** instruction. Memory location 0 stores the start of the free memory.

```
AR_COPY a3 a5
AR_SET_SIZE a5 s1
AR_SET_IDX a5 s4 7
```

(b) (15 points) For the same memory representation below, write in the row “New Value” any values that would change from executing the series of **ROFLcode** instructions.

```
LOAD 1 regD
LOAD regD regC
MEM_COPY regC regD
VAL_COPY 8 regA
SUB regA regC regB
```

(c) (5 points) Final value for registers after the above **ROFLcode** instructions:

- regA: __________
- regB: __________
- regC: __________
- regD: __________

<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>116</td>
<td>2</td>
<td>114</td>
<td>113</td>
<td>1</td>
<td>107</td>
<td>'c'</td>
<td>-9.5</td>
<td>3</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>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location</th>
<th>112</th>
<th>113</th>
<th>114</th>
<th>115</th>
<th>116</th>
<th>117</th>
<th>118</th>
<th>119</th>
<th>120</th>
<th>121</th>
</tr>
</thead>
<tbody>
<tr>
<td>Old Value</td>
<td>7</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</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>
Question 4: Construct Context Free Grammars. 
Design a context free grammar for the following problems. Denote non-terminals with single quotes.

Example CFG rule: \( S \rightarrow 'a' \ B \ 'c' \) The \( S \) nonterminal consists of a terminal \( a \), followed by a non-terminal \( B \), followed by a terminal \( c \).

(a) (10 points) All possible sets of matched parentheses.

(b) (1 point) All strings consisting of a series of \( a \)'s followed by a series of \( b \)'s, where there are always more \( a \)'s than \( b \)'s.

(c) (2 points) All possible palindromes (strings that read the same forward or backward) over the alphabet \( \{a, b, c\} \).
Question 5: Compile Good To Bad .................................................. 40 points
Given the source code, fill in the blanks in the Bad Code generated by a compiler.

HAI 1.450
IM IN YR LOOP
  I HAS A array ITZ A YARN AN THAR IZ 10
  array R "I’m"
  IN array’Z QUOSHUNT OF WHATEVR 20 PUT GIMMEH
  GTFO
  VISIBLE array’Z 2
NOW IM OUTTA YR LOOP
KTHXBYE

<table>
<thead>
<tr>
<th>command</th>
<th>arg1</th>
<th>arg2</th>
<th>arg3</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_start_1:</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR_SET_SIZE</td>
<td>a1</td>
<td>s2</td>
<td></td>
</tr>
<tr>
<td>AR_SET_SIZE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AR_SET_IDX</td>
<td>a3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>AR_SET_IDX</td>
<td>a3</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>AR_SET_IDX</td>
<td>a3</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td></td>
<td>a3</td>
<td>a1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>s4</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>s5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAL_COPY</td>
<td>20</td>
<td>s6</td>
<td></td>
</tr>
<tr>
<td>DIV</td>
<td>s5</td>
<td>s6</td>
<td>s7</td>
</tr>
<tr>
<td>AR_GET_IDX</td>
<td>a1</td>
<td>s7</td>
<td>s8</td>
</tr>
<tr>
<td>AR_SET_IDX</td>
<td>a1</td>
<td>s7</td>
<td></td>
</tr>
<tr>
<td>VAL_COPY</td>
<td>s4</td>
<td>s8</td>
<td></td>
</tr>
<tr>
<td>JUMP</td>
<td>loop_end_2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>VAL_COPY</td>
<td>2</td>
<td>s9</td>
<td></td>
</tr>
<tr>
<td>OUT_CHAR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JUMP</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Question 6: Deterministic Finite Automata

(a) (20 points) For the following Deterministic Finite Automata (DFA), write a Regular Expression that accepts the same language.

(b) (5 points) 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>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(c) (15 points) Which strings does the DFA accept?

- bb
- babbcc
- bbbbcc
- babbabbabbcc
- babbabbabbbb
- babbabbb
- babbbccc
- babbccc
- babcc
You want to implement a new pair literal in LOLcode that generates an pair of values (called left and right) contained within less-than and greater-than operators and separated by a comma. The types of left and right can be NUMBR literals, LETTR literals, or other pair literals. However, the type of left and right must **NOT** match. Examples of some legal pair literals are:

- `<5, 'c'>` Nice and simple pair literal!
- `<6, '<d', 0>>` Pairs are allowed in pairs!
- `<'y', <<'o', '<8', 8>>, 9>>` You can nest as deep as you like!

Examples of ILLEGAL pair literals are:

- `<>` Pairs can’t be empty!
- `<7>` Pairs must have a left and right value!
- `<'c', 'y'>` Left and right can’t have the same type!
- `<1, 'K>', '<G', 7>>` Can’t match types (both sides are pairs)!

Write a set of Context-Free Grammar rules that capture the grammar for pair_literal. The only terminal symbols you may use are: LETTR_LITERAL, NUMBR_LITERAL, LESS_THAN, GREATER_THAN and COMMA.
Question 8: Context Free Grammar Matching ................................. 40 points

For each of the following Context Free Grammars (CFGs), indicate which of the strings can be generated? Note: terminals are lowercase letters and non-terminals are uppercase letters.

(a) (20 points)

S : A B
A : b | c | t
B : C A | B B
C : a | e | i | o | u
   - CAB
   - cat
   - toot
   - cucuc
   - tubui
   - betot
   - bewit
   - cattab
   - tettoob
   - tebicot

(b) (20 points)

S : A S | B
A : h | k
B : A | w | C C
C : p | d
   - w
   - k
   - pd
   - kwh
   - kkdd
   - kkhpd
   - hhkkhw
Question 9: Construct Regular Expression ........................................... 40 points

(a) (20 points) Design a regular expression that will match all positive even integers (no leading zeros) and nothing more.
For example, 15048 and 9860 are matched but 04 and 123407 are not.

(b) (20 points) Design a regular expression that will identify strings with at least one consecutive repeated letter. For simplicity, the language is restricted to the letters a, b, and c.
Matching examples:
  • aa
  • abcbabba
  • aabbccacbbab
  • abb
Non-matching examples:
  • a
  • ababa
  • ababa
  • abcbbabca
Question 10: LMAOcode to ROFLcode .................................................. 40 points
For each LMAOcode instruction below, convert to ROFLcode. For this page only, you can not use the MEM_COPY command.

(a) (10 points) VAL_COPY s10 s40

(b) (10 points) RANDOM s9

(c) (10 points) AR_GET_SIZE a8 s99

(d) (10 points) AR_SET_IDX a19 2 'n'}
Question 11: New Context Free Grammar .................................................. 40 points

You want to implement a new array literal in LMAOcode that generates an array based on a comma-separated series of one or more LETTR literals or one or more NUMBR literals contained within "[" and "]". Examples of some legal arrays are:

```
[[ 1, 2, 3, 4, 5 ]]
[[1,2,3,4,5,100]]
[[ 'G', 'o', 'o', 'd', '!' ]]
[ [ 42 ] ]
```

No spaces are required!
Character arrays are okay!
Minimum one entry; spaces okay!

Examples if ILLEGAL array are:

```
[[ ]]
[[ 1, 2, '3', 4]]
[[ 1, 2, , 4]]
[[ 1, 2, 4, ]]
[[ 1 + 2, 3]]
```

Must have at least one entry!
Cannot mix types!
Cannot have an extra comma!
Cannot have a trailing (or leading) comma!
Cannot have expressions!

Write a set of CFG syntactic rules that capture the grammar for array_literal. These are the only tokens you may use:

LETTR_LITERAL, NUMBR_LITERAL, OPEN_BRACKET, CLOSE_BRACKET and COMMA.
Given the context free grammar:

\[
\begin{align*}
S & : A \ 'b' \\
A & : 'a' \ A \ 'a' \\
A & : 'b'
\end{align*}
\]

Draw the parse tree (using the CFG above) associated for the string: \text{aaabaaab}
Question 13: NFA .................................................. 40 points
Create a Non-deterministic Finite State Machine for the following regular expression:

\(((ab)+(0*1))\)+
Question 14: DFA .................................................. 40 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>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
</tr>
<tr>
<td>D</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

(b) Which strings does the DFA accept?
- 0
- 000
- 1010
- 00010
- 10000000
- 010101000100
- 000100
- 0001000
- 0001000
LMAOcode 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`]  
Apply the given math operation on `num1` and `num2`, and place the answer in `result`.

**SUB**  
[VAL: `num1`]  [VAL: `num2`]  [SCL: `result`]  

**MULT**  
[VAL: `num1`]  [VAL: `num2`]  [SCL: `result`]  

**DIV**  
[VAL: `num1`]  [VAL: `num2`]  [SCL: `result`]  

**VAL_COPY**  
[VAL: `from`]  [SCL: `to`]  
Copy the value `from` into the scalar variable `to`.

**OUT_NUM**  
[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

**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_NO**  
[VAL: `test`]  [VAL: `line`]  
Move the instruction pointer to `line` if `test` is NOT equal to zero.

**RANDOM**  
[VAR: `result`]  
Generate random number 0 to 100 and store in `result`.

**IN_CHAR**  
[VAR: `result`]  
Retrieve a character from STDIN 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: `size`]  
Create an `array` with `new_size` space for elements

**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.

ROFLcode Overview

ROFLcode (assembly) is very similar to LMAOcode (intermediate code), with a handful of changes
- The AR_* 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`.