First Midterm Exam
CS 314, Spring '23
February 24
MIDTERM C SAMPLE SOLUTION

DO NOT OPEN THE EXAM UNTIL YOU ARE TOLD TO DO SO

Name: _____

Rutgers ID number: _____

Section: _____

WRITE YOUR NAME ON EACH PAGE IN THE UPPER RIGHT CORNER.

Instructions

We have tried to provide enough information to allow you to answer each of the questions. If you need additional information, make a *reasonable* assumption, write down the assumption with your answer, and answer the question. There are **4** problems, and the exam has **7** pages. Make sure that you have all pages. The exam is worth **300** points. You have **70 minutes** to answer the questions. Good luck!

This table is for grading purposes only

1	/ 60
2	/ 80
3	/ 95
4	/ 65
total	/ 300

Problem 1 - Regular Expressions and FSAs (60 pts)

Assume the following regular expression definitions:
lower = (a | b | c | d | e),
upper = (A | B | C | D | E),
digit = (0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9),
special = (# | % | $)

Assume that the following regular expression describes the set of valid identifiers in your programming language:

$$(upper|lower)^* special^+ digit (upper|lower)^+$$

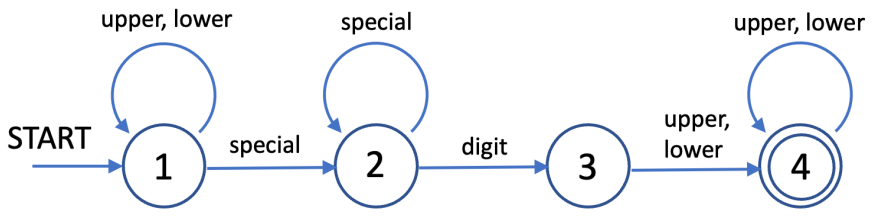
1. Give an example of a string of **length 7** that

   is a valid identifier in this language: abAb#7b

   is not a valid identifier in this language: 5bAb#7b

   (10 pts each)

2. Build a DFA (Deterministic Finite Automaton) for the regular expression of valid identifiers as defined above. The start state is **state 1**, and the final (accepting) state is **state 4**. You are only allowed to add edges with their appropriate labels, i.e., valid labels are *lower*, *upper*, *digit*, and *special*. Note that an edge may have more than one label. (40 pts)



NAME: _____

Problem 2 – Context Free Grammars and Regular Expressions (80 pts)

A *context-free language* is a language that can be specified using a context-free grammar. A *regular* language is a language that can be specified using a regular expression.

For the three languages given below, if the language is context-free, give a compact context-free grammar in Backus-Naur-Form (BNF). If the language is regular, give a compact regular expression using the regular expression syntax introduced in class. If a language is context-free and regular, give both specifications, a BNF and a regular expression. You do not have to justify why you believe a language is not context-free or not regular.

1. { $a^n b^{4n}$ | n $\geq$ 0 }, with alphabet Σ = {a, b}

   S ::= a S bbbb |ϵ

2. { $a^{2n} b^{2m}$ | n > 0, m >0 }, with alphabet Σ = {a, b}

   S ::= A B
   A ::= aa A | aa
   B ::= bb B | bb

   $(aa)^+(bb)^+$

3. { w | w has more than 2 symbols}, with alphabet Σ={a, b}

   S ::= A A A X
   A ::= a | b
   X ::= a X | b X | ϵ

   (a | b) (a | b) (a | b)$^+$

NAME: _____

Problem 3 – Context Free Grammars (95 pts)

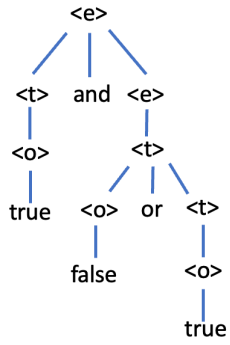
Assume the following context-free logical expression grammar **in BNF** over the alphabet (set of tokens) Σ = { **true**, **false**, **or**, **and** } with the **start symbol** <e>.

```
1: <e> ::= <t> and <e>
2: <e> ::= <t>
3: <t> ::= <o> or <t>
4: <t> ::= <o>
5: <o> ::= true
6: <o> ::= false
```

1. Give a left-most derivation ($\Rightarrow_L$) for the sentence `true and false or true`.(20 pts)

   <e> $\Rightarrow_L$ <t> and <e> $\Rightarrow_L$ <o> and <e> $\Rightarrow_L$ true and <e> $\Rightarrow_L$ true and <t> $\Rightarrow_L$ true and <o> or <t> $\Rightarrow_L$ true and false or <t> $\Rightarrow_L$ true and false or <o> $\Rightarrow_L$ true and false or true

2. Show the corresponding parse tree for your left-most deriviation.(20 pts)



3. Is the grammar LL(1)? Justify your answer by showing the parse table. Entries in the parse table are rules numbered 1 through 6 as listed above.(40 pts)

	or	**and**	**true**	**false**
<e>			1, 2	1, 2
<t>			3, 4	3, 4
<o>			5	6

   Since there is more than one rule for a non-terminal symbol and a single input (more than one entry in a slot), the grammar **is not LL(1)**.

4. What is the associativity of the `or` operator as specified in the grammar? (5 pts) right

   What is the associativity of the `and` operator as specified in the grammar? (5 pts) right

   What is the precedence of the `and` and `or` operators as specified in the grammar? (5 pts)
   or binds stronger than and

NAME: _____

Problem 4 – Syntax-Directed Translation (65 pts)

Assume the following partial expression grammar:

```
<expr>   ::=  + <expr> <expr>  |
              * <expr> <expr>  |
              <const>
<const>  ::=  1 | 2
```

instr. format	description	semantics
memory instructions		
`loadI` <const> $\Rightarrow r_x$	load constant value <const> into register r_x	$r_x \leftarrow$ <const>
arithmetic instructions		
`add` r_x , r_y $\Rightarrow r_z$	add contents of registers r_x and r_y , and write result into register r_z	$r_z \leftarrow r_x + r_y$
`mult` r_x , r_y $\Rightarrow r_z$	multiply contents of registers r_x and r_y , and write result into register r_z	$r_z \leftarrow r_x * r_y$

Here is a recursive descent parser that implements a compiler for the above grammar. Here is the important part of the code:

```
int expr() {
        int reg, left_reg, right_reg;
        switch (token) {
        case '+': next_token();
                  left_reg = expr(); right_reg = expr(); reg = next_register();
                  CodeGen(ADD, left_reg, right_reg, reg);
                  return reg;
        case '*': next_token();
                  left_reg = expr(); right_reg = expr(); reg = next_register();
                  CodeGen(MULT, left_reg, right_reg, reg);
                  return reg;
        case '1':
        case '2': return const();
        }
}

int const() {
        int reg;
        switch (token) {
        case '1':  next_token(); reg = next_register();
                     CodeGen(LOADI, 1, reg);
                     return reg;
        case '2': next_token(); reg = next_register();
                     CodeGen(LOADI, 2, reg);
                     return reg;
        }
}
```

NAME: _____

Make the following assumptions:

- The value of variable "token" has been initialized correctly.
- **The first call to function `next_register()` the shown parser returns integer value "1".** In other words, the first register that the generated code will be using is register r_1.
- Your parser "starts" by calling function `expr()` on the entire input.
- The function `CodeGen` formats the instructions and writes them to the output.

  Examples:

  `CodeGen(ADD, 3, 5, 8)` writes ILOC instruction *add* $r3, r5 \Rightarrow r8$ to the output

  `CodeGen(MULT, 3, 5, 8)` writes ILOC instruction *mult* $r3, r5 \Rightarrow r8$ to the output

  `CodeGen(LOADI, 17, 2)` writes ILOC instruction *loadI* $17 \Rightarrow r2$ to the output

1. Show the sequence of ILOC instructions that the recursive descent parser will generate, i.e., print to the output for the input:

   ```
   * + 2 2 1
   ```

   NOTE: THE INSTRUCTION FORMAT AND THE GENERATED INSTRUCTION ORDER HAVE TO BE CORRECT IN ORDER TO GET CREDIT FOR THE ANSWER. This problem is about understanding the provided code fragment and its execution. (30 pts)

loadI 2 ⇒ r1
loadI 2 ⇒ r2
add r1, r2 ⇒ r3
loadI 1 ⇒ r4
mult r3, r4 ⇒ r5

NAME: _____

2. Change the basic recursive-descent parser to implement an interpreter for our example language. You may insert pseudo code in the spaces marked by _____. No error handling is necessary. (35 pts)

```
int expr() {

        int a, b;
        switch (token) {
        case '+': next_token();
                    a = expr(); b = expr();


                    return (a + b);
        case '*': next_token();
                    a = expr();b = expr();


                    return (a * b);
        case '1':
        case '2': return const();
        }
}

int const() {

        switch (token) {
        case '1':  next_token();

                      return 1;

        case '2': next_token();

                      return 2;
        }
}
```