• Please read all instructions (including these) carefully.

• There are 4 questions on the exam, all with multiple parts. You have 75 minutes to work on the exam.

• The exam is open note - you may use your own notes and laptop/tablet/smartphone, but you may not share notes or electronic devices with others, and you must disable all network access on any device used.

• Please write your answers in the space provided on the exam, and clearly mark your solutions. You may use the backs of the exam pages as scratch paper. Please do not use any additional scratch paper.

• Solutions will be graded on correctness and clarity. Each problem has a relatively simple and straightforward solution. You may get as few as 0 points for a question if your solution is far more complicated than necessary. Partial solutions will be graded for partial credit.

NAME: ____________________________________________

In accordance with both the letter and spirit of the Honor Code, I have neither given nor received assistance on this examination.

SIGNATURE: ______________________________________

<table>
<thead>
<tr>
<th>Problem</th>
<th>Max points</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>TOTAL</td>
<td>75</td>
<td></td>
</tr>
</tbody>
</table>
1. **Lexical Analysis** (15 points)

Consider the following finite automaton for processing a lexical specification:

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
<td>5</td>
<td>T</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>E</td>
<td>4</td>
<td>E</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>E</td>
<td>5</td>
<td>E</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>A</td>
<td>A</td>
<td>T</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>T</td>
</tr>
</tbody>
</table>

The automaton is represented as a table, but note that this table is different from the tables discussed in lecture. The automaton has five states 1-5, each of which has an associated action for each input a-c or end-of-file $. For a given state S and input (or end-of-file) I:

- If the \([S,I]\) entry is a state \(n\), then in state S on input I the automaton transitions to state \(n\) and advances to the next input character.
- If the \([S,I]\) entry is A, then in state S on input I the automaton accepts the prefix of the input up to, but not including, the current input character. This prefix is removed from the input, and the automaton moves to state 1 and continues. Note the current input character is not consumed.
- If the \([S,I]\) entry is E, then in state S on input I the automaton reports an error and halts.
- If the \([S,\$]\) entry is T, then in state S on input $ the automaton terminates successfully.

a) (10 points) Give a lexical specification that tokenizes input strings in the same manner as this automaton. Your lexer should only describe successfully terminating runs of the automaton. For full credit, your specification should be the shortest (in total length of rules) possible.
b) (5 points) Is this form of automaton (a finite state machine extended with actions A, E, and T) powerful enough to implement any lexical specification given by regular expressions? Ignore error recovery, and justify your answer.
2. **LL(1) Parsing** (20 points)

Consider the following grammar:

\[
\begin{align*}
S & \rightarrow \text{ bAb } \mid \text{ bBa} \\
A & \rightarrow \text{ aS } \mid \text{ CB} \\
B & \rightarrow \text{ b } \mid \text{ BC} \\
C & \rightarrow \text{ c } \mid \text{ cC}
\end{align*}
\]

a) (5 points) Give two reasons why this grammar is not LL(1).

b) (5 points) Rewrite the grammar, introducing as few new non-terminals as possible, so that it is LL(1) and recognizes the same strings as the original grammar.
c) (5 points) Construct the first and follow sets for each non-terminal in the rewritten grammar.

d) (5 points) Construct an LL(1) parse table for the rewritten grammar.
3. **SLR(1) Parsing** (20 points)

Consider the language generated by the regular expression $0^*11$.

a) (10 points) Write an SLR(1) grammar for this language that requires $O(n)$ stack space for an input string of length $n$. Prove your grammar is SLR(1) and justify the space bound.

b) (10 points) Write an SLR(1) grammar for this language that requires $O(1)$ stack space for an input string of length $n$. Prove your grammar is SLR(1) and justify the space bound.
4. **Language Design** (20 points)

Pat is writing a compiler for equality tests on numbers, and wants to handle both integers and floating point values. Pat’s initial version looks like this:

```plaintext
[0-9]+ { return(INT); }
[0-9]+"."[0-9]* { return(FLOAT); }
"==" { return(EQ); }
```

This version works fine with Pat’s shift-reduce parser generator, but Pat wants the language to also allow comparisons between integers and floating point values (converting the integer to floating point before comparison), and adds one more production to the grammar, changing the `fltval` rule to:

```plaintext
fltval → FLOAT | INT
```

This augmented grammar is no longer SLR(1). What conflict(s) will there be in the generation of the parsing table? Can they be resolved with precedence declarations on operators? Why or why not?

a) (5 points) This augmented grammar is no longer SLR(1). What conflict(s) will there be in the generation of the parsing table? Can they be resolved with precedence declarations on operators? Why or why not?

b) (5 points) Pat’s friend Chris suggests using a recursive descent parser instead. Will a recursive descent parser (using the augmented grammar as written) generate the parse tree that Pat wants in all cases? Justify your answer.
c) (5 points) Pat’s other friend Francis thinks that the problem can be solved in the lexical analysis. Is this possible? Justify your answer.

d) (5 points) Drew thinks the right answer is to change the language to use a different operator (e.g. `~=`) for floating-point comparisons. Explain how this resolves the conflict(s) in the SLR(1) parsing table, or show what conflict(s) remain.