CSE 401/M501 22au Midterm Exam 11/4/22

Name _______________________________________ ID # ___________________

There are 5 questions worth a total of 100 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed books, closed notes, closed electronics. However, you may have one 5x8 notecard for reference with any hand-written information you wish on both sides. Please turn off all cell phones, personal electronics, alarm watches, and pagers, and return your tray tables and seat backs to their full upright, locked positions. Sound or video recording and the taking of photographs is prohibited.

If you have a question during the exam, please raise your hand and someone will come to help you.

There is an extra blank page at the end of the exam you can use if your answer(s) do not fit in the space provided. Please indicate on the original page(s) if your answer(s) is(are) continued on that last page.

Please wait to turn the page until everyone is told to begin.

Score _________________

1 _______ / 20

2 _______ / 16

3 _______ / 30

4 _______ / 16

5 _______ / 18

Question 1. (20 points) Regular expressions and DFAs.

(a) (10 points) Give a regular expression that generates all strings with any combination of `a`'s, `b`'s, and `c`'s such that the strings contain at least one `a` and at least one `b`. Except for this restriction, the letters in the string may appear in any order – in particular the required `b` may appear before or after the required `a`.

Fine print: You must restrict yourself to the basic regular expression operations covered in class and on homework assignments: *rs* , *r*|*s* , *r** , *r*+ , *r*?, character classes like `[a-cxy]` and `[^aeiou]`, abbreviations *name*=*regexp*, and parenthesized regular expressions. No additional operations that might be found in the "regexp" packages in various Unix programs, scanner generators like JFlex, or programming language libraries are allowed.

(b) (10 points) Draw a DFA that accepts all valid strings of `a`'s, `b`'s, and `c`'s that contain at least one `a` and at least one `b` (i.e., the set described above and generated by the regular expression from part (a)).

Question 2. (16 points) Ambiguity. Here is a small grammar for somewhat silly English sentences.

S ::= `see the` *D T A* | `see the` *D A*
D ::= `orange` | `happy`
T ::= `duck` | `crab`
A ::= `walk` | *T* `walk`

Is this grammar ambiguous? If so, give a proof that it is by showing two distinct parse trees or two different leftmost (or two different rightmost) derivations for some string generated by the grammar. If not, give an informal, but precise, argument why it is not ambiguous.

Notes: whitespace in the grammar is only for readability and is not part of the grammar or the strings generated by it. Each of the words like `see` or `happy` in a grammar production should be treated as a single terminal symbol, not as individual letters.

Question 3. (30 points) The "they *still* can't think of *anything* original!!" LR parsing question. Here is a simple grammar for a language with a `print` and `if` statement and the terminal symbol `x`. The extra *S'* ::= *S* `$` rule needed to handle end-of-file in an LR parser has been added for you. As is usual, whitespace in the grammar is only for readability and is not part of the grammar or the strings generated by it. Terminal symbols like `print` or `if` are single symbols, not strings of letters.

0. *S'* ::= *S* `$` (`$` is end-of-file)
1. *S* ::= `print` *E*
2. *S* ::= `if` *E* *S*
3. *E* ::= `x`

(a) (16 points) Draw the LR(0) state machine for this grammar. When you finish, you should number the states in the final diagram in whatever order you wish so you can use the state numbers in later parts of this question.

Question 3. (cont.) Grammar repeated from previous page for reference if needed:

0. *S′* ::= *S* `$` (`$` is end-of-file)
1. *S* ::= `print` *E*
2. *S* ::= `if` *E S*
3. *E* ::= `x`

(b) (10 points) Write the LR(0) parser tables for the LR parser in your answer to part (a).

(c) (4 points) Is this grammar LR(0)? Explain why or why not. Your answer should describe **all** of the problems that exist if the grammar is not LR(0) by identifying the relevant state number(s) in your answers to parts (a) and (b) and the specific issues in those state(s) (i.e., something like “state 47 has a shift-reduce conflict if the next input is `foo`”, but with, of course, state numbers and correct details from your parser). If the grammar is LR(0) you should explain why (this can be brief).

Question 4. (16 points) LL parsing. Here is a small grammar that generates strings of the letters a, m, n, and o. As usual, whitespace in the grammar rules is only for readability and not part of the generated strings.

1. R ::= a$TC$o | o
2. T ::= m | ε
3. C ::= n | ε

(a) (8 points) Complete the following table showing the FIRST and FOLLOW sets and nullable for each of the non-terminals in this grammar:

	FIRST	FOLLOW	nullable
R			
T			
C			

(b) (8 points) Is this grammar, as written, suitable for constructing a top-down LL(1) predictive parser? If it is, your answer should give a technical explanation why it is. If not, your answer should give a technical explanation describing the problem or problems with this particular grammar that prevent it from being suitable for a LL(1) predictive parser.

Question 5. (18 points) Semantics. Suppose we have the following assignment statement in a MiniJava program:

```
      p = 2*x.f(a[i]);
```

(a) (8 points) Draw an abstract syntax tree (AST) for this statement at the bottom of this page. You should use appropriate names for AST nodes and have an appropriate level of abstraction and structural detail similar to the AST nodes in the MiniJava project AST classes, but don't worry about matching the exact names or details of classes or nodes found in the MiniJava code.

(b) (10 points) Annotate your AST by writing next to the appropriate nodes the checks or tests that should be done in the static semantics/type-checking phase of the compiler to ensure that this assignment statement does not contain errors. If a particular check or test applies to multiple nodes, you can write it once and indicate which nodes it applies to, as long as your meaning is clear and readable. You may assume that `int` is the only numeric type in the language, but remember that MiniJava also contains `boolean` and object (class) types.

Extra space for answers, if needed. Please be sure to label which question(s) are answered here, and be sure to put a note on the question page so the grader will know to look here.