CSE 401/M501 24au Midterm Exam 11/1/24

Name ______________________________________ UW netid ____________@uw.edu

There are 7 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, the taking of photographs, and time travel are 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.

After the blank page with extra space for answers is a copy of the MiniJava grammar. You should remove that page from the exam and use it for reference as needed.

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

Score _________________

1 _______ / 18

2 _______ / 12

3 _______ / 10

4 _______ / 36

5 _______ / 8

6 _______ / 14

7 _______ / 2

Question 1. (18 points) Regular expressions and DFAs. A *decimal fixed-point number* is similar to a floating-point number in that it has a decimal integer part and an optional decimal fraction. Unlike a floating-point number, it does not have an exponent. For this problem, decimal fixed-point numbers are strings *n.f* consisting of an integer *n* and an optional fractional part *.f* with the following properties:

- The integer part *n* is required and may not have any leading zeros, unless it is the number 0 itself.
- The fractional part is optional. If it is present (*.f*) it may not have any trailing 0's.
- If a decimal point is present, meaning the number has both an integer and a fractional part, then there must be at least one digit before the decimal point and one digit after it.

Examples of strings that are legal decimal fixed-point numbers according to these rules: 0, 0.1 , 17 , 1.23 , 101010 , 101010.010101 , 9080.706 , 3.14159 , 401.501 .

Examples of strings that are not legal decimal fixed-point numbers according to these rules: 17.0 (trailing 0 in fractional part), 0.0 (trailing 0 in fractional part), 401.5010 (trailing 0 in fractional part), 01.23 (leading 0 in integer part), .1 (no integer part), 17. (no digits after the decimal point .).

(continued

on

next

page)

⇩

Question 1 (cont.) (a) (8 points) Give a regular expression that generates all strings that are legal decimal fixed-point numbers according to the rules on the previous page.

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 decimal fixed-point numbers described above and generated by the regular expression from part (a).

Question 2. (12 points) Scanners. A scanner for MiniJava should be able to process any text input, even if it contains something other than a legal MiniJava program. It should report errors if it encounters input characters that do not form proper MiniJava tokens.

What happens if we use a scanner for MiniJava to process the following input?

```
public void f(string s) {
  /* int i;
  /* int[] */ arr; */
   return arr[0xffffdc04];
}
```

Below, list in order the tokens that would be returned by a scanner for MiniJava as it reads this input. If there is a *lexical* error in the input, indicate where that error is encountered by writing a short explanation of the error in between the valid tokens that appear before and after the error(s) (something brief like "illegal character %" would be fine). The token list should include any tokens found after any error(s) in the input, i.e., scanning should continue after discovering an error. You may use any reasonable token names (e.g., LPAREN, ID(x), etc.) as long as your meaning is clear.

A copy of the MiniJava grammar is attached as the last page of the exam. You should remove it from the exam and use it for reference while you answer this question. You should assume that the scanner processes MiniJava syntax as defined in that grammar, with no extensions or changes to the language. Also recall that the MiniJava project defines an <IDENTIFIER> as a sequence of letters, digits, and underscores, starting with a letter; uppercase letters are distinguished (different) from lowercase; and an <INTEGER_LITERAL> is a sequence of decimal digits not starting with 0, or the number 0 by itself, denoting a decimal integer value.

Question 3. (10 points) Ambiguity. Consider the following grammar for arithmetic expressions involving addition (+) and exponentiation (`**`):

E ::= *E* + *E* | *E* `**` *E* | `x`

(a) (4 points) Show that this grammar is ambiguous by constructing two different parse trees for a single string generated by the grammar, with the following restriction: your string must include at least one occurrence of the `**` exponentiation operator.

(b) (6 points) Give an unambiguous grammar that generates the same strings and has the correct precedence and associativity for expressions involving + and `**` using the following rules: Exponentiation has higher precedence than addition, so `x**x+x` means `(x**x)+x`. Addition is left-associative, so `x+x+x` means `(x+x)+x`. Exponentiation is right-associative, so `x**x**x` means `x**(x**x)` (i.e., $x^{(x^x)}$).

Question 4. (36 points) The grammar/parser kitchen sink question! Consider the following grammar. The nonterminal *S* is the start symbol of the grammar, and the extra *S'* ::= *S* $ rule needed to handle end-of-file in an LR parser has been added for you.

0. *S'* ::= *S* `$` (`$` is EOF)
1. *S* ::= `a` *X*
2. *S* ::= `b`
3. *X* ::= *S* `a` *Y*
4. *Y* ::= `b c`
5. *Y* ::= `b` *S*

(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 that you can use the state numbers in later parts of this question. The state numbers should be successive integers starting with 0, 1, 2, 3, ….

(continued on next page)

Question 4. (cont.) Grammar repeated from previous page for reference:

0. $S' ::= S\ \$$ ($ is EOF)
1. $S ::=$ a X
2. $S ::=$ b
3. $X ::= S$ a Y
4. $Y ::=$ b c
5. $Y ::=$ b S

(b) (8 points) Write the LR(0) parser table for the LR parser DFA shown in your answer to part (a). To save time, an empty table is provided below. However, it probably has more rows than you need. Use only as many rows as needed and leave the rest blank.

State #	a	b	c	$	*S*	*X*	*Y*
0							
1							
2							
3							
4							
5							
6							
7							
8							
9							
10							
11							
12							
13							
14							
15							

(continued on next page)

Question 4. (cont.) Grammar repeated from previous pages for reference:

0. $S' ::= S$ `$` (`$` is EOF)
1. $S ::=$ `a` X
2. $S ::=$ `b`
3. $X ::= S$ `a` Y
4. $Y ::=$ `b c`
5. $Y ::=$ `b` S

(c) (3 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 `blah`", but with, of course, correct state numbers and details from your parser). If the grammar is LR(0), you should give a technical explanation why it is (this can be brief).

(d) (6 points) Complete the following table showing the FIRST and FOLLOW sets and nullable for each of the nonterminals in this grammar. You should include $ (the end-of-file marker) in the FOLLOW set for any nonterminal where it is appropriate.

	FIRST	FOLLOW	nullable
S			
X			
Y			

(e) (3 points) Is this grammar SLR? Give a brief technical explanation why or why not.

Question 5. (8 points) Top-down parsing. Take another look at the grammar from the previous problem, but omitting the *S'* ::= *S* $ rule that was added for LR parsing:

1. *S* ::= `a` *X*
2. *S* ::= `b`
3. *X* ::= *S* `a` *Y*
4. *Y* ::= `b c`
5. *Y* ::= `b` *S*

(Recall that *S* is the start symbol for this grammar.)

Is this grammar suitable for constructing a top-down LL(1) predictive parser? If so, explain why. If not, explain why not, and, if possible, construct a different grammar that generates the same language that is suitable for a top-down LL(1) predictive parser, or explain why this can't be done. (You might find the FIRST/FOLLOW/Nullable information from the previous problem useful in answering this question.)

Question 6. (14 points) Semantics. Python includes a convenient operator for checking whether an integer value occurs in an integer array. The expression *e1* `in` *e2* evaluates to true if the `int` value *e1* occurs in array *e2*, and false otherwise. Suppose we add this `in` operator to our MiniJava language without making any other changes to the language, and then we encounter the following statement in our new version of MiniJava:

```
while (i in arr) { i = i + 1; }
```

(a) (7 points) At the bottom of this page, draw an abstract syntax tree (AST) for this loop. You should use appropriate names for the 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 starter code.

(b) (7 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 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.

Question 7. (2 free points) (All reasonable answers receive the points. All answers are reasonable as long as there is an answer. ☺)

(a) (1 point) What question were you expecting to appear on this exam that wasn't included?

(b) (1 points) Should we include that question on the final exam? (circle or fill in)

Yes

No

Heck No!!

$!@$^*% No !!!!!

Yes, yes, it *must* be included!!!

No opinion / don't care

None of the above. My answer is ________________________________.

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.