## CS-3323 Principles of Programming Languages Midterm Exam

## $March\ 4,\ 2020$

Name:	ID#	
	tal of 20pts. The maximum time to complete it f notes. However, no digital device can be used	
Problem 1 (5pts)		
	ne MINUS binary operator (-), left-associativity the how the evaluation steps and the final result of e	
(1pt) Consider the regular expression: (	(a b)*c + (bc dc)*	
Write two strings that would be rejected strings must be at least 4 symbols long.	l and two string that would be accepted by the a	bove regular expression. All
(0.5pt) What's the total number of toke	ens that a C scanner would produce for the follow	wing code snippet?
while ( a + b < 10 ) b = b + 2 ;		

(1pt) For the below statements, mark either True or False:

Statement	
Left/right operator associativity is necessary for the + operator	
Operator associativity deals with the order of evaluation of a sequence of operations	
of the exact same type	
LL parsing works in a bottom-up fashion	
LL parsers are perfectly suitable to handle left-recursion	

(0.5pt) Write a regular expression that would accept the following 4 strings:

- a b
- a a b b c c
- abbbcccc
- b b b b b c c c c c

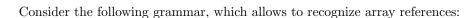
(1.0pt) Write a small grammar that permits to produce lists of function arguments for a C-like language. The tokens to use in this grammar are: ID, INT, FLOAT, LEFTPAR '(', RIGHTPAR ')' and COMMA ', '.  $\epsilon$  is the empty string.

You can use the pipe character '|' to separate multiple right-hand-sides for the same non-terminal. The grammar can have either left-recursion or right-recursion, but not both types.

Below are 3 examples, one per line, of the strings that could be accepted:

- (int a, int b)
- ( int a, float b, int c)
- (int a)
- ()

## Problem 2 (10pts)



- 1. varref  $\rightarrow$  T\_ID dimlist
- 2. dimlist  $\rightarrow$  dimlist '[' fact ']'
- 3. dimlist  $\rightarrow \epsilon$
- 4. fact  $\rightarrow$  varref
- 5. fact  $\rightarrow$  T\_NUM
- 6. fact  $\rightarrow$  '-' fact

(2pts) Assuming a right-most derivation, write all the sentential forms for the string: T\_ID [ T\_NUM ] [ - T\_ID ]

(3pts) For the same string, and assuming again a right-most derivation, draw the corresponding parse tree:

(5pts) For the same string and assumptions, use the below LR parsing table to accept or reject it.

		State /
State 0		6 fact: '-' . fact
0 \$accept: . varref \$end		
		T_ID shift, and go to state 1
T_ID shift, and go to state 1		T_NUM shift, and go to state 6
1 1D shift, and go to state 1	Ct-t- 4	
	State 4	'-' shift, and go to state 7
varref go to state 2		
	0 \$accept: varref \$end .	varref go to state 8
State 1		fact go to state 10
1 varref: T_ID . dimlist	\$default accept	State 8
	State 5	
\$default reduce using rule 3 (dimlist)	2 dimlist: dimlist '[' . fact ']'	4 fact: varref.
decidate reduce using rule 5 (diffinite)	2 diffilist. diffilist [ . idet ]	Tracer variety
dimeliat me to ototo 2	TID skift and make state 1	Edefault made as using mule 4 (fact)
dimlist go to state 3	T_ID shift, and go to state 1	\$default reduce using rule 4 (fact)
	T_NUM shift, and go to state 6	State 9
State 2	'-' shift, and go to state 7	
0 \$accept: varref . \$end		2 dimlist: dimlist '[' fact . ']'
_	varref go to state 8	
\$end shift, and go to state 4	fact go to state 9	']' shift, and go to state 11
4 ,	State 6	State 10
Ctata 2	5 fact: T_NUM .	6 fact: '-' fact.
State 3	5 fact: 1_NOW .	o fact: - fact.
1 varref: T_ID dimlist .		
2 dimlist: dimlist . '[' fact ']'	\$default reduce using rule 5 (fact)	\$default reduce using rule 6 (fact)
		State 11
'[' shift, and go to state 5		
		2 dimlist: dimlist '[' fact ']'.
\$default reduce using rule 1 (varref)		
vacidate reduce asing rule r (varier)		\$default reduce using rule 2 (dimlist)
		\$default reduce using rule 2 (dimlist)
T , C, •		• •

Input String	Stack Contents	Action
		Taken
\$ T_ID [ - T_NUM ] [ T_ID ]	\$ (0)	

Additional space for Problem 2.

Input String	Stack Contents	Action Taken
		Taken

## Problem 3 (5pts)

Design an LL grammar that is capable of recognizing C++-style declarations of containers. The containers that your grammar **must** support are: **vector**, **map**, **set**, and **pair**. The basic data-types to consider are: **int**, **float**, **string**.

(1pt) List all the tokens you will use in your grammar. You can use the examples in the 3rd, 4th and 5th part of this problem to identify the tokens.

(1pt) Define the rules for two non-terminals, datatype and container\_name. These non-terminals will produce the majority of tokens defined above, but will not be the only ones.

(1pt) Create all the remaining rules for the grammar. You must use the above non-terminals. You should define a non-terminal *container* that permits to derive C++ declarations of template strings. Your grammar must meet all the requirements of an LL grammar. If necessary, you can use additional non-terminals, but please keep it to the minimum.

The following is an incomplete list of strings that should be accepted by your grammar:

- vector < int > grades
- $\bullet$  vector < vector < int >> matrix
- set < int > numbers
- $\bullet \ map < set < string >, pair < float, float >> \ points$

(1pt) For the grammar you proposed in the previous step, show all the ser $set < int >> mytable$	ntential forms for the string: $vector <$
(1pt) Enhance your proposed grammar with a rule (or rules) to produce the * (STAR) token operator. The grammar should allow to define single- (e int ***).  Examples of acceptable strings are:  • $vector < int > * ii$ • $vector < int > * * ii$ • $set < string > * aa$ • $set < string > * aa$ • $map < int *, int > xy$	e pointers to containers. You must use e.g. int *) and multi-level pointers (e.g.

This page can be used as scratch area.