Exam 1

Graded

Student

Adam Fenjiro

Total Points

63.5 / 70 pts

Question 1

Q1 **4** / 4 pts

1.1 **firstSprime** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect

1.2 **firstS** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect

1.3 **firstX** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect

1.4 **firstY** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect

1.5 **followSprime** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect

1.6 **followS** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect

1.7 **followX** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect

1.8 **followY** **0.5** / 0.5 pts

✔ **– 0 pts** Correct

– 0.5 pts Incorrect

Question 2

Q2 **4** / 4 pts

✔ **– 0 pts** Correct

– 0.5 pts Minor error

– 1 pt S'() design incorrect/missing

– 1.5 pts S() design incorrect/missing

– 1.5 pts T() design incorrect/missing

– 4 pts Incorrect

Question 3

Q3 **8** / 8 pts

3.1 a **3** / 3 pts

✔ **– 0 pts** Correct

– 0.5 pts One set incorrect

– 1 pt Two sets incorrect

– 1.5 pts Three sets incorrect

– 2 pts Four sets incorrect

– 3 pts More than four sets incorrect

3.2 b **4** / 4 pts

✔ **– 0 pts** Correct

– 0.5 pts One set incorrect

– 1 pt Two sets incorrect

– 1.5 pts Three sets incorrect

– 2 pts Four sets incorrect

– 2.5 pts Five sets incorrect

– 3 pts Six sets incorrect

– 4 pts More than six sets incorrect

3.3 c **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Incorrect

Question 4

Q4 **8** / 8 pts

✔ **– 0 pts** Correct

– 7 pts Only one parse step correct

– 6 pts Two parse steps correct

– 5 pts Three parse steps correct

– 4 pts Four parse steps correct

– 3 pts Five parse steps correct

– 2 pts Six parse steps correct

– 1 pt Seven parse steps correct

– 8 pts Incorrect

Question 5

Q5 **7.5** / 8 pts

– 0 pts Correct

– 7 pts Two correct states with transitions

– 6 pts Three correct states with transitions

– 5 pts Four correct states with transitions

– 4 pts Five correct states with transitions

– 3 pts Six correct states with transitions

– 2 pts Seven correct states with transitions

– 1 pt Eight correct states with transitions

✔ **– 0.5 pts** More than 8 states correct with transitions

– 8 pts Incorrect

Question 6

Q6 **5** / 5 pts

✔ **– 0 pts** Correct

– 2 pts A rule not updated correctly

– 3 pts A' (or equivalent) rule not created correctly

– 5 pts Incorrect

Question 7

Q7 **6** / 6 pts

✔ **– 0 pts** Correct

– 2 pts S rule not updated correctly

– 2 pts A rule not updated correctly

– 2 pts B rule not updated correctly

– 6 pts Incorrect

Question 8

Q8 **4** / 4 pts

✔ **– 0 pts** Correct

– 1 pt S' row incorrect

– 1 pt S row incorrect

– 1 pt A row incorrect

– 1 pt B row incorrect

– 4 pts Incorrect

Question 9

Q9 **6** / 6 pts

✔ **– 0 pts** Correct

– 6 pts Only one parse step correct

– 5 pts Two parse steps correct

– 4 pts Three parse steps correct

– 3 pts Four parse steps correct

– 2 pts Five parse steps correct

– 1 pt Six parse steps correct

– 0.5 pts Seven parse steps correct

– 8 pts Incorrect

Question 10

Q10 **7** / 7 pts

10.1 **a** **6** / 6 pts

✔ **– 0 pts** Correct

– 1 pt S_0 row incorrect

– 1 pt S_1 row incorrect

– 1 pt S_2 row incorrect

– 1 pt S_3 row incorrect

– 1 pt S_4 row incorrect

– 1 pt S_5 row incorrect

– 1 pt S_6 row incorrect

– 1 pt S_7 row incorrect

– 6 pts Incorrect

10.2 **b** **1** / 1 pt

✔ **– 0 pts** Correct

– 1 pt Incorrect

Question 11

Q11 **4** / 10 pts

– 0 pts Correct

– 1 pt Various minor errors

– 2 pts Various errors

– 1 pt Integer constant isn't passed from scanner

– 2 pts Lists aren't printed in the appropriate rule

– 2 pts No list (or equivalent) is created in the appropriate rule

– 2 pts Missing some Flex rules

– 4 pts Incorrect Flex rules

✔ **– 2 pts** Incorrect Bison definitions

– 2 pts Some Bison actions are missing or incorrect

✔ **– 4 pts** Incorrect Bison actions

– 10 pts Incorrect or not attempted

CS4121 Exam #1

Feb. 19, Spring 2025 (70 Points Total)

Name: Adam FENJIRO User ID: afenjiro

(User ID is your Michigan Tech email ID. For example, put in *joe* if your email address is joe@mtu.edu.)

(Note: For the grammars in Questions 1-10, an upper-case letter denotes a non-terminal and a lower-case letter denotes a terminal.)

1. (4 pts) Calculate the First, Follow sets for the grammar below.

$S' \rightarrow S$
$S \rightarrow XY$
$X \rightarrow a$
$\quad | \; b$
$Y \rightarrow c$
$\quad | \; d$

Non – terminal	*First Set*	*Follow Set*
S'	{a, b}	{$}
S	{a, b}	{$}
X	{a, b}	{c, d}
Y	{c, d}	{$}

2. (4 pts.) Construct pseudo-code for a top-down recursive descent parser for the following grammar. You may assume the existence of a routine called `match` that takes a character (token) parameter and if that parameter matches the next input symbol, it advances the input pointer and returns true; otherwise it just returns false.

$S' \rightarrow S$
$S \rightarrow \mathbf{p}\, S\, T$
$\quad | \; \mathbf{m}\, S\, T$
$\quad | \; T$
$T \rightarrow \mathbf{n}\, S$
$\quad | \; \mathbf{d}$

```
S(){
  if(match(p) || match(m)){
    if(S()){
      if(T()){
        return true;
      }
    }
  }
  else if(T()){return true;}
  else {return false;}
}
```

```
T(){
  if(match(n)){
    if(S()){
      return true;
    }
  }
  else if(match(d)){
    return true;
  }
  else {return false;}
}
```

3. (8 pts) Calculate the First, Follow and Predict sets for the grammar below. Is this grammar LL(1)? Why?

$$
\begin{array}{lcl}
S' & \rightarrow & S \\
S & \rightarrow & ACB \\
A & \rightarrow & Ba \\
 & | & \epsilon \\
B & \rightarrow & bC \\
 & | & \epsilon \\
C & \rightarrow & cBd \\
 & | & f
\end{array}
$$

(a) First and Follow sets

Non − terminal	*First Set*	*Follow Set*
S'	{b, a, c, f}	{$}
S	{b, a, c, f}	{$}
A	{b, a, ε}	{c, f}
B	{b, ε}	{$, a, d}
C	{c, f}	{b, $, a, d}

(b) Predict sets

Rule	*Predict Set*
$S' \rightarrow S$	{b, a, c, f}
$S \rightarrow ACB$	{b, a, c, f}
$A \rightarrow Ba$	{b, a}
$A \rightarrow \epsilon$	{c, f}
$B \rightarrow bC$	{b}
$B \rightarrow \epsilon$	{$, a, d}
$C \rightarrow cBd$	{c}
$C \rightarrow f$	{f}

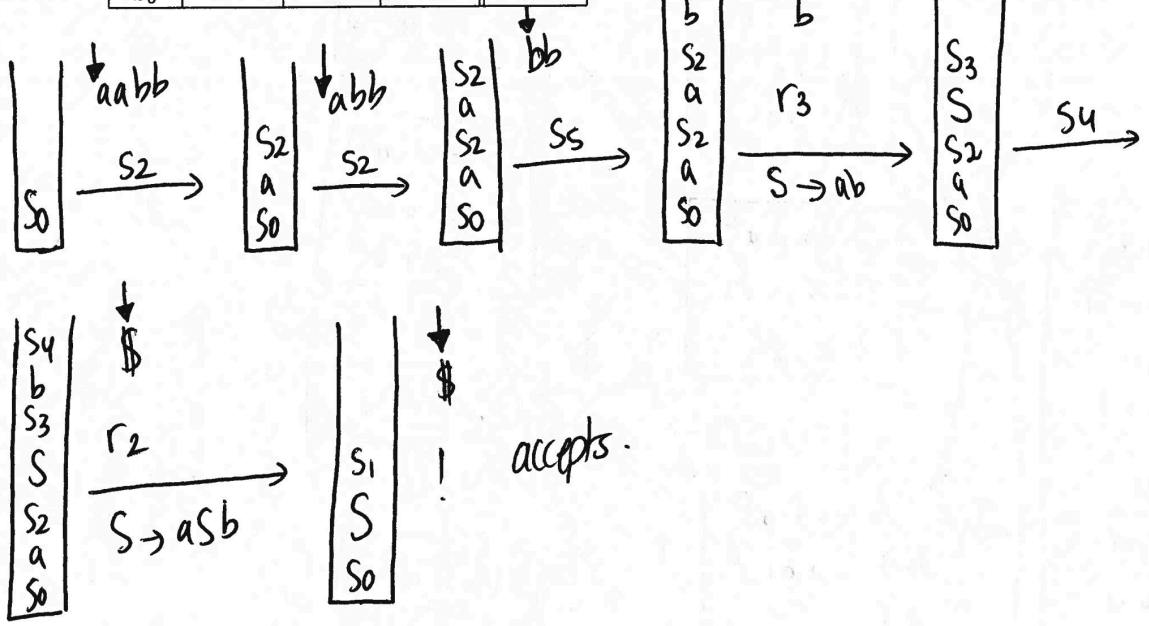
(c) Is this grammar LL(1)? Why?

Yes, this grammar is LL(1).
This is because there are no conflicts.

4. (8 pts.) Show the parsing process of *aabb* based on the following grammar and its SLR parse table.

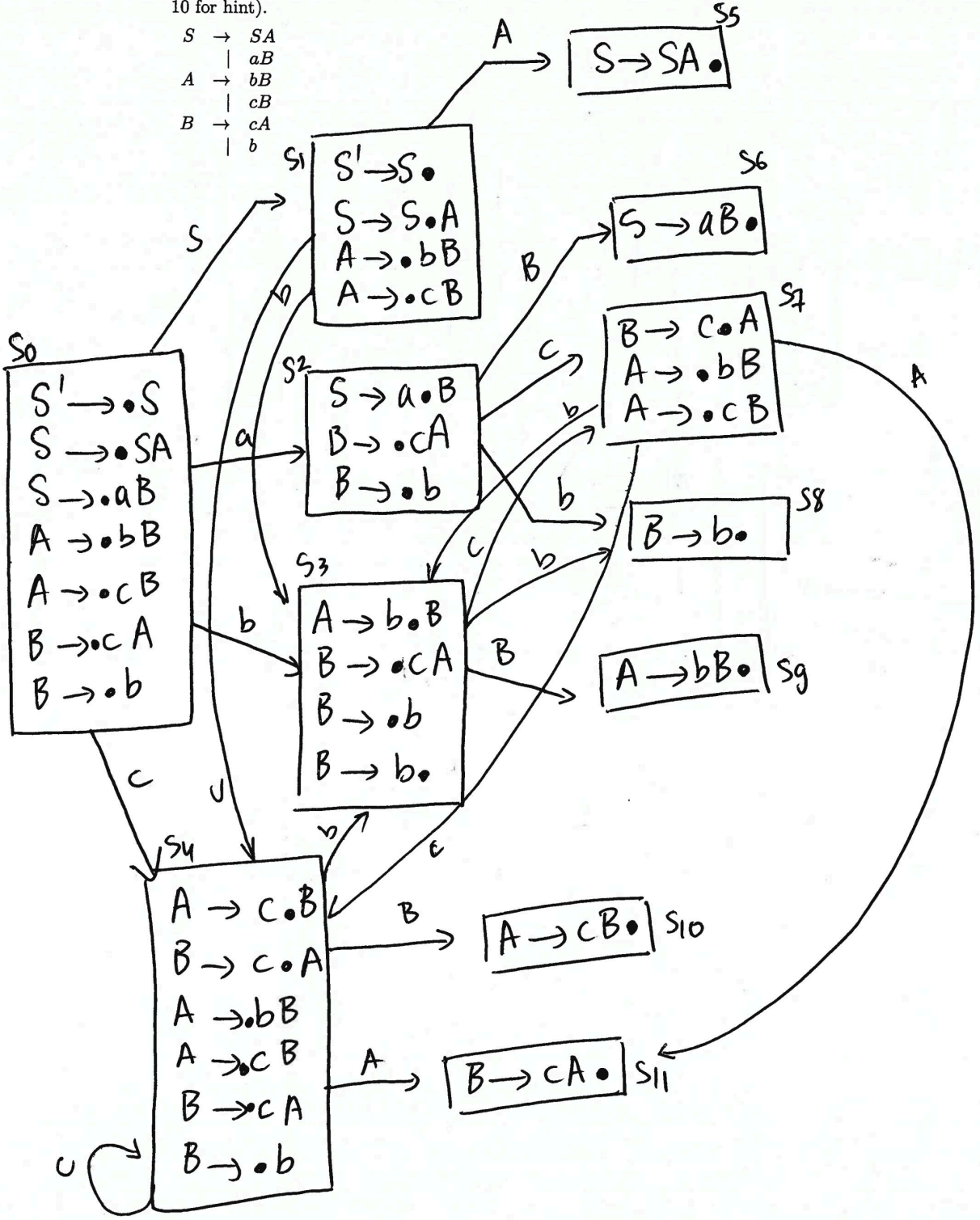
1. $S' \rightarrow S$
2. $S \rightarrow aSb$
3. $\quad\quad | \; ab$

State	a	b	\$	S
S_0	$s2$			1
S_1			!	
S_2	$s2$	$s5$		3
S_3		$s4$		
S_4		$r2$	$r2$	
S_5		$r3$	$r3$	



5. (8 pts.) Construct the LR(0) items sets (CFSM) for the following grammar (You do NOT need to calculate First and Follow sets and do NOT need to show parse table for this question. See Question 10 for hint).

$$
\begin{aligned}
S &\rightarrow SA \\
&\mid aB \\
A &\rightarrow bB \\
&\mid cB \\
B &\rightarrow cA \\
&\mid b
\end{aligned}
$$



6. (5 pts.) Eliminate left recursion in the following grammar.

$$
\begin{array}{rcl}
A & \rightarrow & Aaa \\
 & | & Abb \\
 & | & cc \\
 & | & dd
\end{array}
$$

$A \rightarrow ccA'$
$A \rightarrow ddA'$
$A' \rightarrow aaA'$
$A' \rightarrow bbA'$
$A' \rightarrow \varepsilon$

7. (6 pts.) Eliminate left recursion in the following grammar.

$$
\begin{array}{rcl}
S & \rightarrow & Sa \\
 & | & Ab \\
A & \rightarrow & Bc \\
 & | & Sd \\
B & \rightarrow & Ae \\
 & | & fg
\end{array}
$$

$S \rightarrow Sa \mid Ab \Rightarrow S \rightarrow AbS' \quad S' \rightarrow aS' \mid \varepsilon$

$A \rightarrow Bc \mid Sd \Rightarrow A \rightarrow Bc \mid AbS'd \Rightarrow A \rightarrow BcA' \quad A' \rightarrow bS'dA' \mid \varepsilon$

$B \rightarrow Ae \mid fg \Rightarrow B \rightarrow BcA'e \mid fg \Rightarrow B \rightarrow fgB' \quad B' \rightarrow cA'eB' \mid \varepsilon$

Thus:

$S \rightarrow AbS'$
$S' \rightarrow aS'$
$S' \rightarrow \varepsilon$
$A \rightarrow BcA'$
$A' \rightarrow bS'dA'$
$A' \rightarrow \varepsilon$
$B \rightarrow fgB'$
$B' \rightarrow cA'eB'$
$B' \rightarrow \varepsilon$

8. (4 pts.) Given the following grammar (Note that each rule is associated with a rule number) and the predict sets, fill in the LL(1) parse table.

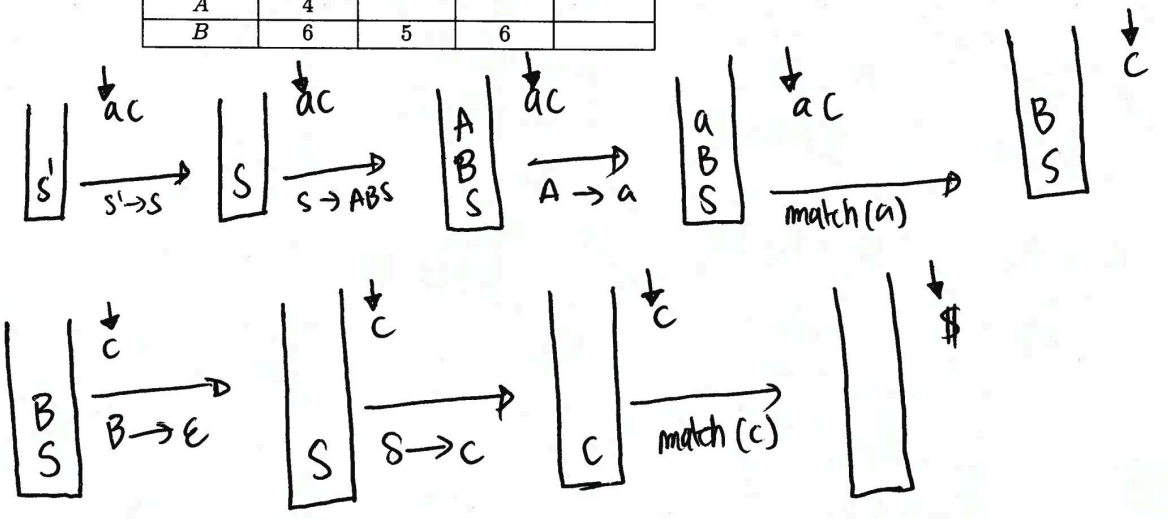
1. $S' \rightarrow S \quad \{a, c\}$
2. $S \rightarrow ASB \quad \{a\}$
3. $\quad | \; c \quad \{c\}$
4. $A \rightarrow a \quad \{a\}$
5. $B \rightarrow b \quad \{b\}$
6. $\quad | \; \epsilon \quad \{\$\}$

Stack top	a	b	c	\$
S'	1		1	
S	2		3	
A	4			
B		5		6

9. (6 pts) Given the following grammar (Note that each rule is associated with a rule number) and its LL(1) parse table, show the LL(1) parsing process of ac.

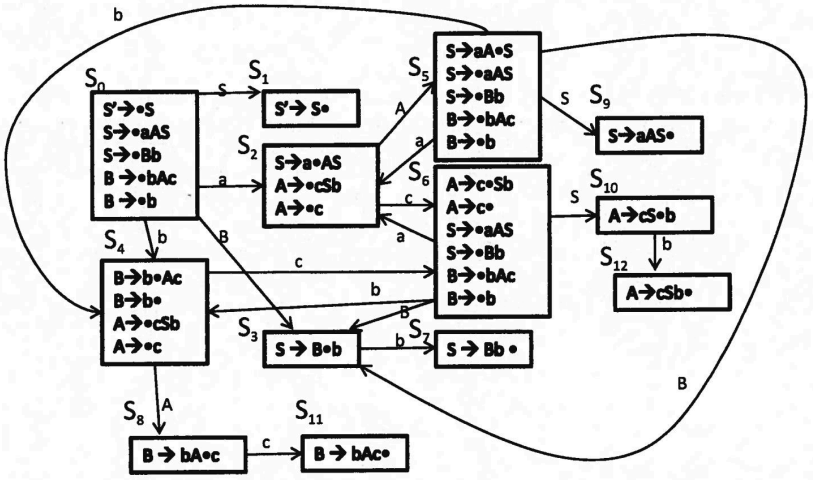
1. $S' \rightarrow S$
2. $S \rightarrow ABS$
3. $\quad | \; c$
4. $A \rightarrow a$
5. $B \rightarrow b$
6. $\quad | \; \epsilon$

Stack top	a	b	c	\$
S'	1		1	
S	2		3	
A	4			
B	6	5	6	



10. (7 pts.) Given the following grammar (Note that each rule is associated with a rule number) and the follow sets, answer the following questions (**Don't forget Question (b)**):

1.	S'	$\rightarrow$	S	$Follow(S') = \{\$\}$
2.	S	$\rightarrow$	aAS	$Follow(S) = \{\$, b\}$
3.		$\mid$	Bb	
4.	A	$\rightarrow$	cSb	$Follow(A) = \{a, b, c\}$
5.		$\mid$	c	
6.	B	$\rightarrow$	bAc	$Follow(B) = \{b\}$
7.		$\mid$	b	



(a) Fill in the SLR parse table for states S_0 through S_7 based on the given CFSM.

State	a	b	c	\$	S	A	B
S_0	S_2	S_4			1		3
S_1				!			
S_2			S_6			5	
S_3		S_7					
S_4		r_7	S_6		8		
S_5	S_2	S_4			9		3
S_6	S_2 / r_5	S_4 / r_5	r_5		10		3
S_7		r_3		r_3			
...							

(b) Is this grammar SLR? Why?

No, this grammar is not a SLR.
This is because there are some shift-reduce conflicts.

11. (10 pts) `Map/reduce` is a popular programming paradigm for massive data processing. The concept of `map/reduce` can be dated back to the map and reduce functions in functional programming many decades ago. Consider the following grammar for Scheme `map expressions`. (Scheme is a functional programming language we will cover in more detail later this semester.)

$$
\begin{array}{rcl}
S' & \rightarrow & S \\
S & \rightarrow & S\ E \\
 & | & E \\
E & \rightarrow & (\ B\ ) \\
B & \rightarrow & map\ F\ E \\
 & | & L \\
F & \rightarrow & add1 \\
 & | & dbl \\
L & \rightarrow & intconst \\
 & | & L\ intconst
\end{array}
$$

In this grammar, we treat `map`, `add1`, `dbl` as keywords and `intconst` is a terminal that accepts a non-negative integer. Function `add1` increments a value by 1 and `dbl` doubles a value. A Scheme list is a list of integers enclosed by a pair of parentheses. The function `map` applies a function to each element of a list and results in a new list. For example, `(map add1 (1 2 3))` yields (2 3 4) and `(map dbl (2 5 7))` yields (4 10 14). You are asked to write an interpreter for Scheme map expressions using Bison/Flex. You only need to complete the Bison and Flex rules/actions sections and the type definitions of the Bison symbols if your design uses their attributes. A sample input for this grammar is as follows:

```
(1 2 3)
(map add1 (1 2 3))
(map dbl (1 2 3))
(map add1 (map dbl (1 2 3 4 5)))
```

Your interpreter is expected to output:

```
(1 2 3)
(2 3 4)
(2 4 6)
(3 5 7 9 11)
```

You can assume some commonly used data structures, such as queue, stack, and linked list, are already implemented. You can also assume a function named `apply(func, list)` exists, which applies a function to all members of the given list. You can also assume a `printList(list)` function exists that prints a given list. Comment your code. (Hint: Note that tokens in this grammar can only be ints and lists. Declare your types appropriately.)

(a) Complete the Flex rules/actions section below for *MapScanner.l*. Refer to *MapParser.y* in Question (b) for token names.

```
/* insert actions inside the curly brackets as needed */
%%
map       { return MAP;                          }
add1      { return ADD1;                         }
dbl       { return DBL;                          }
\(        { return LP;                           }
\)        { return RP;                           }
[0-9]+    { yytext.ival = atoi(yytext); return INTCONST; }
[ \t\n]   {;} /*no action needed for this regular expression*/
.         {printf("Scanner: lexical error");     }
%%
```

(b) Complete the Bison rules/actions section below for *MapParser.y*.

```
/* define attribute types for the symbols as needed */
```

~~[illegible]~~

union ( )

apply (func , list)

printList (lst)

// Define some used variables in the c code

continue to the next page.

```
/* insert actions inside the curly brackets as needed */
%%
Program : S;
S       : S E
        {
            $$ = $2
        }
        | E
        {
            $$ = $1
        }
E       : LP B RP
        {
            $$ = $1 + $$2 + $$1
        }
B       : MAP F E
        {
            map($$, $1, $2)
        }
        | L
        {
            $$ = $3
        }
F       : ADD1
        {
            $$ = $1 + $S2
        }
        | DBL
        {
            $$ = $$1 $1 + $1
        }
L       : INTCON
        {
            $$ = $2
        }
        | L INTCON
        {
            $$ = $1
        }
%%
```