CS164 Spring 1997 Midterm 1

Prof. S. L. Graham

Problem 1 (10 points)

Following are some small Flex examples For each one, state what Flex will print for the given input.

a.
```
%%
ab      { printf("1"); }
ab*     { printf("2"); }
%%
```

input: `abbbab`

b.
```
%x STATE2
%%
ab      { printf("x"); }
ab*     { printf("y"); }
a       { BEGIN(STATE2); count=1 }
<STATE2>{
a       { count++; printf("%d",count); }
b       { count--; if(count==0) { BEGIN(INITIAL); }; }
}
%%
```

input: `aababbababb`

c.
```
%%
a       { printf("a"); }
a+      { printf("b"); }
a*b     { printf("c"); }
b*      { printf("d"); }
.       { printf("%s", yytext); }
%%
```

input: `a+aa+b+ab+abb+abbbb`

Problem 2 (20 points)

Indicate whether each statement is true or false by circling either T or F. For each statement, give a brief explanation (one sentence or an example)justifying your answer.

T **F** Every language can be specified by a context-free grammar.

T **F** If a grammar G is the smallest grammar that generates a given language, then G is reduced.

T **F** Some reduce/reduce conflicts can be eliminated by introduction of Bison-style precedence and/or associativity rules.

T **F** Regular expressions can derive the empty string.

T **F** It is always the scanner's job to recognize keywords.

T **F** For an unambiguous grammar and a string X generated by the grammar, the leftmost nad rightmost derivations for X will always produce parse trees of the same shape.

T **F** If a grammar has a reduce/reduce conflict, then it is ambiguous.

T **F** Instead of using Flex to write a scanner, you could write it with Bison

T **F** Instead of using Bison to write a parser, you could write it with Flex.

Problem 3 (20 Points)

Following is a context-free grammar. This grammar describes a small subset of the English language. An example of a sentence of the language is "mice which eat fish eat mice". All sentences in the language contain a subject (S) and a predicate (P). Subjects are nouns (N) optionally followed by a list of subordinate clauses (Cs). Each subordinate clause is the keyword *which* followed by a verb (V) and a predicate. The only verb in the language is *eat*. A predicate consists of a verb followed by a subject.

1. `Sentence -> S P`
2. `S -> N Cs`
3. `N -> fish`
4. `N -> mice`
5. `Cs -> /* empty */`
6. `Cs -> Cs C`
7. `C -> which P`
8. `P -> V S`
9. `V -> eat`

a. Give a leftmost derivation of the sentence "mice eat fish"

Use the parse table below to answer parts (b) - (d)

State	*fish*	*mice*	*eat*	*which*	*$*	*Sent*	*S*	*N*	*Cs*	*C*	*P*	*V*
0	s1	s2				13	3	4				
1			r3	r3	r3							
2			r4	r4	r4							
3			**?**								6	7
4			r5	r5	r5				8			
5	r9	r9										
6					r1							
7	s1	s2					9	4				
8			r2	**?**	r2					11		
9			r8	r8	r8							
10			s5								12	7
11			r6	r6	r6							
12			r7	r7	r7							
13					acc							

b. Fill in the two missing stack configurations (indicated by the blank boxes):

0	fish which eat mice eat fish $
0 fish 1	which eat mice eat fish $
0 N 4	which eat mice eat fish $
.	.
.	.
.	.
0 S 3 V 7	fish $

c. What is the missing entry for state 3? (Hint: step through the LR parser execution using a very simple sample input such as "mice eat fish". When you reach the unknown action, first decide whether a shift or a reduce is called for. If you decide to reduce, find an applicable rule to use. If you decide to shift, you can immediately ignore all states which have an error given the current lookahead token. Beyond that, you're on your own.)

d. Consider the missing action in state 8 on the lookahead token *which*. This parse table entry determines whether *which* is left-or right-associative. For example, consider the subject "mice which eat fish which eat mice". There are two possible ways to parse this sentence. One would be parenthesized as "mice which eat (fish which eat mice)" - the mice are getting revenge on those nasty fish. Theis corresponds to the right associativity. The other possibility is "(mice which eat fish) which eat mice". Here, the left-associativity of *which* makes mice into cannibals, devouring both mice and fish.

   What whould the action be in order for *which* to be left-associative? What should it be if *which* is right associative? (Your answers should be of the form "shift" or "reduce by rule n". You do not have to say which state to transition to on the shift action.)

Problem 4 (15 Points)

For each grammar, indicate whether it is LR(0) by circling the answer. If the grammar is not LR(0), breifly explain why not. (Hint: in most cases, it is **not** necessary to construct the states in order to answer the question.)

LR(0)	**Not LR(0)**	`S -> bool == S | bool`
LR(0)	**Not LR(0)**	`S -> not S | true`
LR(0)	**Not LR(0)**	`S -> S + S | num`
LR(0)	**Not LR(0)**	`S -> -S | real * S | int`
LR(0)	**Not LR(0)**	`S -> abS | aS | b`

Problem 5 (15 Points)

Following are six regular expressions

1. `(b|a+b+|c+b+)*(a+|c+)?`
2. `[ab]*[bc]*`

3. `([bc]*(aba?|ba))*[cb]*`
4. `[ac]*(b(a[ac]*)?)?`
5. `[bc]*a([bc]*a[bc]*a[bc]*)*`
6. `a*[bc][abc]*`

For each description below, identify each regular expression that it describes, or none of the above

a. All strings of `a`'s, `b`'s, and `c`'s in which all `a`'s occur before any `c`'s.
b. All strings of `a`'s, `b`'s, and `c`'s without `bc` in them.
c. All strings of `a`'s, `b`'s, and `c`'s that begin or end with `bc`.
d. All strings of `a`'s, `b`'s, and `c`'s in which `a` and `c` are never adjacent
e. All strings of `a`'s, `b`'s, and `c`'s with an odd number of `a`'s

Problem 5 (15 Points)