**COP3402 Spring 2011 Final Exam Sample Problems**

1. Write a sequence of lex-style regular expressions for each of the following sets

A = { w | w is over the alphabet {a,b,1,2} and with every numerical character preceded by an alphabetic one }

* *Could be any number of alphabetic characters with no constraints e.g. aab, aabbabab …*
* *Each numerical character is preceded by an alphabetic one*



B = { x | x is a binary string of odd parity } *= string has odd number of 1s*

* *All strings with even number of 1s : *
* *Add one more 1 (makes their number odd) and arbitrary number of 0s :*



1. Write an unambiguous grammar that leads to correct parse trees for the language consisting of expressions involving the operand ID and the operators described below.

OPERATOR ASSOCIATIVITY PRECEDENCE BINARY/UNARY

-, + right to left High (3) Unary

^ right to left Medium (2) Binary

&, | left to right Low (1) Binary

Parentheses are also allowed, with their usual interpretation.



Present a parse tree, using your grammar, for the string  
  
- + ID & ID ^ - ID ^ ID

*Here is the leftmost derivation*:

****

*And here is the parse tree*:



1. Consider the repeat statement, which has the following description:  
     
   **repeat\_stmt → REPEAT POSITIVE IDENT = expression SEMICOLON**

Wan expression involves **IDENTs**, positive numbers and the binary operators, **+**, and **\***, with normal precedence and associativity. Expressions can also involve parentheses for subexpressions. Write the procedures, **repeat\_stmt**( ) and **expression**( ), needed to do a recursive descent parse of a REPEAT statement. Assume token are returned by a procedure token( ) which sets a global variable SY. Assume **SY =** **REPEAT** at start. Assume **SY = IDENT** on an identifier, **SY = POSITIVE** on a positive number, ASSIGN on “=”, **SEMICOLON** on ";", **LPAREN** on a “(“, **RPAREN** on a “)”, **PLUS** on a “+” and **TIMES** on a “\*”

1. Redo #3 but with a Bison grammar. Recall how precedences are set.
2. What are the triples that might be generated as intermediate code for?

**REPEAT 7 a = (b+a) \* e \* 2 + d;**

1. Consider the following bison grammar for binary real numbers. Add actions at each rule so the printf will print the real version of the input:

**%union{ int ival;**

**float rval; }**

**%type <rval> R F**

**%type <ival> N B D**

**%%**

**N: R { printf("value = %f\n", $1); };**

**R: B '.' F { $$ = $1 + $3 / 2.0f; } |**

**B { $$ = $1; }**

**;**

**B: B D { $$ = 2 \* $1 + $2; } |**

**D { $$ = $1; }**

**;**

**F: D F { $$ = $2 / 2.0f + $1} |**

**D { $$ = $1; }**

**;**

**D: '0' { $$ = 0; } |**

**'1' { $$ = 1; }**

**;%%**

**#include "lex.yy.c"**

**where the Flex is created by**

**%%**

**. { return(yytext[0]); }**

EXAMPLE: Input = 101.101. Output is 5.625

1. Consider the following context free grammar

**S : E ';' | 'if' E 's' | B 's' | 'if' B ';' (1) to (4)**

**E : 'exp' (5)**

**B : 'exp' (6)**

(a) Produce an LR(1) parser. There are fewer than 15 states.

(b) Show what states would be merged to create an LALR(1) parser.

Point out any conflicts that arise.

**state 0**

**$accept : \_S , $end**

**S : \_E ; , $end**

**S : \_if E s , $end**

**S : \_B s , $end**

**S : \_if B ; , $end**

**E : \_exp , ;**

**B : \_exp , s**

**if shift 3**

**exp shift 5**

**S goto 1**

**E goto 2**

**B goto 4**

**state 1**

**$accept : S\_ , $end**

**$end accept**

**state 2**

**S : E\_; , $end**

**; shift 6**