**COP3402 Spring 2011 Final Exam Name Cheat Sheet**

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

**A** =

**B** =

 **2.** Consider the following grammar that leads to correct parse trees for the language consisting of expressions involving the operand **v**, parentheses and the operators described below.

**Blah, blah**

Fill in the following table with associativity (left-to-right or right-to-left or non-associative), precedence, and whether or not the associated operator is binary or unary. Assume that 1 indicates the lowest precedence operators, 2 the next higher, etc.

|  |  |  |  |
| --- | --- | --- | --- |
| **OPERATOR** | **ASSOCIATIVITY** | **PRECEDENCE** | **BINARY/UNARY** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Present a parse tree, using this grammar, for the string

**Blah blah blah**

**3.** Consider the ***blah*** statement, which has the following description:

***blah* → lots of blahs**

You may assume the existence of functions **int whoKnows ( )** etc.These return the number of the triple associated with their intermediate code generation. Write the function, **int** **blah( )**, needed to do a recursive descent parse of a **blah** statement. Assume tokens are returned by procedure **token( )** which sets a global variable **SY**. Assume **SY =** **…**. Your routine should produce triples that represent intermediate code associated with the ***blah*** statement. Use the triples we did in class, including the …

**4.** Consider the following bison grammar for …. Add actions following each rule so the printf will print …:

**%%**

**…**

**%%**

**…**

EXAMPLE: Input = …; Output is …

 **5.** Present the **CKY** recognition matrix for the string **a – a + a - a** assuming the Chomsky Normal Form grammar specified by the rules

**E → E F | M E | P E | a**

**F → M F | P F | M E | P E**

**P → +**

**M → −**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **a** | **−** | **a** | **+** | **a** | **−** | **a** |
| **1** |  |  |  |  |  |  |  |
| **2** |  |  |  |  |  |  |
| **3** |  |  |  |  |  |
| **4** |  |  |  |  |
| **5** |  |  |  |
| **6** |  |  |
| **7** |  |

Be sure to note whether or not the string is accepted.

**6.** Consider the grammar
G = (…),
where P is: …

Convert this to a right recursive grammar.

Compute the FIRST and FOLLOW sets for this grammar's non-terminals.

Produce the LL(1) parsing table based on these sets. Fill in production numbers, not productions. If there are any conflicts, circle those entries

**7.** Consider the following context free grammar
**…**

 (a) Produce the description of an SLR(1) parser. There are no more than x states. I already did y of them. Point out any and all conflicts that arise.

# State Item(s) Action Goto

**0**

Once again, consider the above context free grammar

 (b) Produce the description of an LR(1) parser. There are no more than w states. I already did z of them. Point out any and all conflicts that might arise.

# State Item(s) Action Goto

**0**

**8.** Consider the following context free grammar

**…**

**The LR(1) parser is specified by the following table:**

# State Item(s) Action Goto

**0**

Indicate what states are merged to create the corresponding LALR(1) parser, and show the combined States, Items, Actions and Gotos.

# State Item(s) Action Goto

**9.** Recall the … dataflow analysis problem. This computes for each basic block the values …, where

XX[B] = { }

YY[B] = { }

Etc. – like the sample I already gave you.

**10.**

Indicate which of the following are true (T) and which are false (F).

|  |  |
| --- | --- |
| **Statement** | **True (T) or False (F)** |
| Recursive descent parsers are easy to create from left recursive grammars | F |
| … | ? |
| Remo is a way cool GTA | T |

**11.** Some questions about things I didn’t cover in the rest of exam.