CS 453 Introduction to Compilers	**Midterm Examination**
Spring 2011	**March 8, 2011**

75 minutes (maximum)
Closed Book

- You may use one side of one sheet (8.5x11) of paper with any notes you like.
- This exam has 12 pages, including this cover page and two mostly empty pages for extra work space. Do all your work on these exam sheets.
- Be specific and clear in your answers. If there is any question about what is being asked, then indicate the assumptions you need to make to answer the question.
- Show all your work if you wish to be considered for partial credit.

Question	Points	Score
1	10	
2	15	
3	10	
4	10	
5	15	
6	5	
7	5	
8	20	
9	10	

Name: _____

Email: _____

DO NOT TURN TO NEXT PAGE TILL YOU GET PERMISSION

1. [10 points] Lexer
   Write the (a) regular expressions, (b) NFA's, and (c) a single DFA for the keyword "to" and identifiers that contain only lower-case letters. Mark each accept state in the DFA with the token that will be returned. The keyword "to" shows up in loop constructs like

```
for i=1 to 5
```

2. [15 points] Predictive Parsing
(a) Assume that you have to implement syntactic analysis for the following language:

```
(1) chestContents     -> itemlist EOF
(2) itemlist          -> item itemlist
(3)                   |  epsilon
(4) item              -> NAME LPAREN ID ingredient_list RPAREN
(5) ingredient_list   -> COMMA ID ingredient_list
(6)                   | epsilon
```

Assume that NAME, LPAREN, RPAREN, COMMA, ID, and EOF are all tokens. Show the FIRST and FOLLOW sets for all of the nonterminals in the above grammar. (DO NOT MODIFY THE GRAMMAR). You do not have to show the nullable property if you are using the book's approach.

(b) Using the FIRST and FOLLOW sets, construct the predictive parsing table for the above grammar.

3. [10 points] Recursive Descent Parser.
   For the grammar in problem [2], write a portion of the recursive-descent, predictive parser with panic-mode error handling. Write the functions that parse the chestContents and ingredient_list nonterminals. Assume the following routines are available for use:

```
match(tok) { if(tok==lookahead) lookahead = scan();
             else throw new SyntaxException(message); }

panic( nonterminal ) {
    print error;
    while ( scan() not in (FOLLOW(nonterminal)) ) {
    }
}
```

   You can leave out the default cases in the switch statements to save room.

problems #2 and #3 more space if needed ...

4. [10 points] Expressions in the MeggyJava Compiler.
   Write the actions for evaluating expressions for the parts of the MeggyJava expression grammar shown below. You should recall how to do this from PA3.

```
exp ::=
        NUMBER:n
        {:




        :}

    |   exp:a MINUS exp:b
        {:



         :}

    |   LPAREN exp:e RPAREN
        {:



         :}
```

5. [15 points] Parse Tree and AST.
   Given the following expression:

```
(byte)5 - (byte)1 + (byte)2
```

   (a) draw the implicit parse tree using the full MeggyJava expression grammar and appropriate precedence and associativity and

   (b) draw the AST that you will be generating for PA4 assuming the following AST nodes PlusExp, MinusExp, ByteCast, MultExp, IntegerExp, and Token.

6. [5 points] AVR code.
   Write the AVR code that implements the following statement:

```
Meggy.setPixel( (byte)0, (byte)3 - (byte)1, Meggy.Color.DARK);
```

7. [5 points] Visitor Design Pattern.
Here are some AST nodes, an abstract visitor class, and a concrete visitor class called MysteryVisitor. Answer the question below the code.

```
public interface Node { void accept(Visitor v); }
public class AddNode implements Node {
    Node left, right;
    AddNode(Node l, Node r) { this.left = l; this.right = r;    }
    void accept(Visitor v) { v.visitAdd(this); } }
public class NumNode implements Node {
    int val;
    NumNode(int val) { this.val = val; }
    void accept(Visitor v) { v.visitNum(this); } }
public interface Visitor {
    void visitAdd(AddNode n);
    void visitNum(NumNode n); }
public MysteryVisitor extends Visitor {
    HashMap<Node,String> mNodeToString;
    Node mRoot;
    void visitAdd(AddNode n) {
        n.left.accept( this );  n.right.accept( this );
        mNodeToString.put(n, "(" + mNodeToString.get(n.left) + " + "
                                          + mNodeToString.get(n.right) + ")" );
        mRoot = n; }
    void visitNum(NumNode n) {
         mNodeToString.put(n, n.val.toString());
         mRoot = n;  }
    String getResult() {
        return mNodeToString.get(mRoot); }
}
public static void main( String[] args ) {
    ...
    Node ast_root = parser.parse();
    MysteryVisitor v = new MysteryVisitor();
    ast_root.accept( v );
    System.out.println( v.getResult() );
}
```

What does the call `System.out.println( v.getResult() );` output given that ast_root references the AST for the expression $3 + 4 + 5 + 6$?

(empty page)

8. [20 points] LR Parsing Table
   Below is the LR parsing table for the following grammar:

$$
\begin{array}{lllll}
(0) & S & \rightarrow & L & \\
(1) & S & \rightarrow & \epsilon & \\
(2) & L & \rightarrow & a \wedge L & \\
(3) & L & \rightarrow & a &
\end{array}
$$

	a	$\wedge$	\$	S	L
0	s2		r(1)		g1
1			accept		
2		s3	r(3)		
3	s2				g4
4			r(2)		

   Show the stack, input, and actions for the shift-reduce parse of the string $a \wedge a$.

9. [10 points] Various Parsers
   Using two to three full sentences describe the relationship between how the terms LL(1), LR(1), right-most derivation, and left-most derivation relate.