CS 453 Introduction to Compilers **Midterm Examination**
Spring 2010 **March 11, 2010**

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 14 pages, including this cover page and a blank page at the end. 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	15	
2	15	
3	10	
4	5	
5	5	
6	15	
7	15	
8	20	

Name: ______________________________

Email: ______________________________

DO NOT TURN TO NEXT PAGE TILL YOU GET PERMISSION

1. [15 points] Predictive Parse Table.

(a) Assume that you have to implement syntactic analysis for the following language:

```
(1) infile   -> songlist EOF
(2) songlist -> songlist song
(3)          |  epsilon
(4) song     -> ES  NAME
(5)          |  OTHER  NAME
```

Assuming that ES, NAME, OTHER, and EOF are all tokens, show the nullable property and 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. You will run into some errors that make it impossible to create a predictive parser. Show the whole table including the errors.

2. [15 points] Top Down Parsing.
(a) Using one to two sentences to describe why the errors occur when constructing the predictive parse table for the grammar in problem #1. Also describe how you can modify the grammar to fix the errors. Show the fixed grammar. Show the modified predictive parsing table.

(b) For the fixed grammar, write a recursive-descent, predictive parser with panic-mode error handling assuming 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.

problem #2 more space if needed ...

3. [10 points] Non-deterministic Finite-state Automata. Draw an NFA that can recognize all of the tokens from the language in problem #1. Mark the accept states and indicate which token each accept state recognizes. Note that:

   - ES is a token with the lexeme "80s",
   - OTHER is a token with the lexeme "ot", and
   - NAME is any string of letters with no spaces.

   You can use LETTER and DIGIT to label edges in the NFA, where LETTER = [A-Za-z] and DIGIT = [0-9].

4. [5 points] Deterministic Finite Automata. Convert the NFA you drew for question #3 to a DFA and draw the DFA.

5. [5 points] Expressions in the MiniJava Compiler.
(a) Write the actions for constructing an AST for the parts of the MiniJava expression grammar shown below. You should recall how to do this from PA3. AST node names can be approximate.

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

        :}

    |   exp:a MINUS exp:b
        {:

         :}

    |   LPAREN exp:e RPAREN
        {:

         :}
```

6. [15 points] Parse Tree and AST.
   Given the following expression:

```
(1-3) * 4 + 5 * 7
```

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

   (b) draw the AST that would be generated in your MiniJava expression compiler.

7. [15 points] Expression Evaluation in MIPS.
Write MIPS code that uses the run-time stack to evaluate the following expression:

```
    x - 4
```

Indicate at which nodes in an AST each set of instructions will be generated from. Assume that the lvalue of the variable `x` is `$fp-4`. You can ONLY use the registers `$fp, $sp, $t0, and $t1`.

Here are some example MIPS instructions to show the syntax you will need:

```
li $t0, 42
lw $t0, 4($t1)
sw $t0, 0($t1)
addi $t1, $t1, 4
sub $t1, $t2, $t3        # $t1 = $t2 - $t3
```

8. [20 points] Visitor Design Pattern.
Here are some AST nodes, an abstract visitor class, and a concrete visitor class called MysteryVisitor. See parts (a), (b), and (c) of the problem below the code and on the next two pages.

```
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 {
    void visitAdd(AddNode n) {
        n.left.accept( this );  n.left.accept( this ); System.out.print( " + " ); }
    void visitNum(NumNode n) {
        System.out.print( " " + n.val + " " ); }
}
public static void main( String[] args ) {
    ...
    Node ast = parser.parse();
    ast_root.accept( new PrintPostfixVisitor() );
}
```

(a) Use one sentence to describe what the call `ast_root.accept( new MysteryVisitor() )` will do given the AST for the expression (3 + 4).

(b) Write down the sequence of visitor-related method calls that occur starting with the call to `ast.accept( new MysteryVisitor() )` on the AST for the expression (3 + 4). Indicate the specific method being called at a virtual call site by using the notation Class::method. Let `v` be a reference to the visitor, `ast_root` be a reference to the AddNode, `p3` be a reference to the NumNode for 3, and `p4` be a reference to the NumNode for 4.

(c) Explain and show code for how we could change the Visitor base class and MysteryVisitor class so that the visit methods in the concrete visitor MysteryVisitor do not have to traverse the data structure. Only show the methods that need to change to handle the AddNode.

(empty page)