Translator Design **CRN 24414** **Test 3 Revision 1**
CMSC 4173 **Spring 2012**

Name:
Student ID:
E-Mail:
Course **CMSC 4173, Translator Design**
CRN: **24414, Spring, 2012**
Date: **Thursday, May 3, 2012 at 5:30 p.m.**

Instructions:

1. Type your name in the space provided.
2. Type your student identifier in the space provided.
3. Type your e-mail address in the space provided.
4. Questions requiring written answers must be answered using standard American English.
5. Answers must be coded legibly. Neatly drawn diagrams are acceptable. Text must be typewritten. Answers that cannot be read by your instructor will be given no credit.
6. You must do your own work.
7. Submit this test with your answers on this document to your instructor electronically on or before **5:30 p.m. on Thursday, May 3, 2012.** Create an electronic note.
   7.1. Send the note to trturner@uco.edu.
   7.2. Make the subject line your CRN, hyphen, your-last-name, hyphen, your-first-name, hyphen, **t03**. For example, if your name is Alan Turing and you are enrolled in **CRN 24414**, the subject line of your note is **24414-Turing-Alan-t03**.
   7.3. Attach this document completed and renamed to your-CRN, hyphen, your-last-name, hyphen, your-first-name, hyphen, **t03.doc**. For example, if your name is Alan Turing and you are enrolled in **CRN 24414**, the subject line of your note is **24414-Turing-Alan-t03.docx**.

Scoring:

1. The table below lists the number of points available for each problem and the total number of points that can be earned on this test.
2. You can earn up to 360 points on the test. You need only earn 200 points to have a perfect score on this examination. Any additional points beyond 200 will be added to your course score.

Problem	Available	Earned
1	20	
2	20	
3.1	20	
3.2	20	
3.3	20	
4.1	20	
4.2	20	
4.3	20	
4.4	20	
4.5	20	
4.6	20	
5.1	20	
5.2	20	
6	20	
7.1	20	
7.2	20	
7.3	20	
7.4	20	
Subtotal	**360**	
Total		

1. (20 points) Diagram and briefly describe the phases of a compiler.
2. (20 points) Construct a nondeterministic finite automata for the regular expression $(a \mid b)^* abb(a \mid b)^*$.
3. Construct minimum-state DFA's for the regular expression $(a \mid b)^* a(a \mid b)$.
   3.1. (20 points) Construct a NFA for the regular expression $(a \mid b)^* a(a \mid b)$.
   3.2. (20 points) Convert the NFA found in 3.1. to a DFA.
   3.3. (20 points) Minimize the DFA found in 3.2.
4. Create an SLR grammar for logical expressions.
   4.1. (20 points) Construct a grammar for logical expressions. Logical expressions have operators **and**, **or**, and **not** as shown in the Table 4. Parenthesized expressions are permitted. Operands consist of **0** or **1**, where **0** signifies false and **1** signifies true. Operators are ranked by precedence where the operator having highest precedence is the grouping operator, **( )**.

Operator	Meaning	Operands	Precedence	Associativity
( )	grouping operator	NA	1	left-to-right
~	not	unary	2	right-to-left
*****	and	binary	3	left-to-right
+	or	binary	4	left-to-right

Table 4. Logic operators

   4.2. (20 points) Construct the first set for all the non-terminals in your grammar.
   4.3. (20 points) Construct the follow set for all the non-terminals in your grammar.
   4.4. (20 points) Construct the canonical LR(0) collection for your augmented grammar.
   4.5. (20 points) Construct the action-goto SLR parsing table for your augmented grammar
   4.6. (20 points) Construct a table that records the moves of an LR parser similar to the table shown in Figure 4.38 on page 253 of Compilers: principles, techniques, and tools 2nd Ed by Aho, Sethi, and Ullman or Figure 4.32 on page 220 of Compilers, principles, techniques, and tools by Aho, Sethi, and Ullman for the expression $a + b * c$, where $a, b,$ and c are logic variables.
5. Consider the grammar in Table 5. Parse trees are discussed in section 4.2.4 of your text, Compilers: principles, techniques, and tools / Aho, A. V. et al. 2nd ed. Terminal symbols are printed in bold. Terminal symbols include **+**, **-**, *****, **/**, **(**, **)**, **id**, **intlit**, and **realit**. An identifier, **id**, begins with a letter and can have zero or more subsequent letters or digits. An integer literal, **intlit**, consists of one or more digits. A real literal, **realit**, is an integer literal and an exponent or a decimal value followed by an optional exponent. A real literal has no embedded blank characters. A decimal value is a decimal point with at least one digit on either side of the decimal point. An exponent is the letter **e** or **E**, followed by an optional sign, followed by an integer literal.

1	*expression*	$\to$	*term*
2	*expression*	$\to$	*expression***+***term*
3	*expression*	$\to$	*expression***-***term*
4	*term*	$\to$	*factor*
5	*term*	$\to$	*term*******factor*
6	*term*	$\to$	*term***/***factor*
7	*factor*	$\to$	**(***expression***)**
8	*factor*	$\to$	**id**
9	*factor*	$\to$	**intlit**
10	*factor*	$\to$	**realit**

Table 5. Expression grammar

5.1. (20 points) Construct a parse tree of the expression **(*f*-32)*5/9**.
5.2. (20 points) Construct an expression tree for expression **(*f*-32)*5/9**.

6. (20 points) Draw diagrams that represent symbol table entries for the declaration given in Figure 6.

```
var a:array[-5..4] of char;
```

Figure 6. Declaration for question 6.

7. Translate program **p01** in figure 7 to P-Code. Put your answer in Table 7.

```
program p01;
    var a: integer;
    function celcius(f:real):real;
    begin
        celcius:=(f-32)*5/9
    end{celcius};
begin{p01}
    WriteInteger(celcius(77))
end{p01}.
```

Figure 7. Program for question 7.

7.1. (20 points) Complete the Label Table in Figure 7.1.

Label	Value

Table 7.1 Label Table

7.2. (20 points) Complete the Constant Tables in Figures 7.2. and 7.3.

Index	Integer constant

Table 7.2 Integer constants

Index	Real constant

Table 7.3 Real constants

7.3. (20 points) Complete the PCode Instruction Table in Figure 7.4..

Index	Label	opcode	operand 1	operand 2

Table 7.4 P-Code Instructions