# Final Assessment Test – July 2023

Course: **BCSE307L - Compiler Design**

Class NBR(s): **0890/0892/0894/0896/0898/0900/0902/ 0904/0906/0908/0910/0912/0914/0916/0920/0922/ 0924/0926/0928/0930/0932/0934/0935/0938/0940/ 1365**

Slot: **C1+TC1**

Time: **Three Hours**

Max. Marks: **100**

**KEEPING MOBILE PHONE/SMART WATCH, EVEN IN 'OFF' POSITION IS TREATED AS EXAM MALPRACTICE**

**Answer ALL Questions**

**(10 X 10 = 100 Marks)**

1. a) A regular expression is ambiguous when there exists a string which can be constructed in two different ways from the regular expression. Which of the following regular expressions are ambiguous? **[5]**

   (i) **a((ab)*cd*U a(ababcb*)*a***

   (ii) **aab*(ab)*U ab*Ua*bba***

   (iii) **aaba*U aaabaU aabba*U a**

   b) Draw a NFA that accepts the language corresponding to the regular expression **((ab)*U(bc)*)ab** by using Thomson construction method. **[5]**

2. Explain the various phases of the compiler and show the output of each phase of the compilation for the given expression.

   a= (b*c) + (c-d) /f

3. Suppose that we want to describe Javastyle class declarations like these using a grammar:

   **class Car extends Vehicle**

   **public class JavaIsCrazy implements Factory, Builder, Listener**

   **public final class President extends Person implements Official**

   One such grammar for this is

   G={ (1) C → P F **class identifier** X Y

   (2) P → **public**

   (3) P → ε

   (4) F → **final**

   (5) F → ε

   (6) X → **extends identifier**

   (7) X → ε

   (8) Y → **implements** I

   (9) Y → ε

   (10) I → **identifier** J

   (11) J → **,** I

   (12) J → ε }

Your job is to construct an LL (1) parser table for this grammar. For reference, the terminals in this grammar are **public final class identifier extends implements**, here **$** is the end of input marker, and the nonterminals are C P F X Y I J.

a) Compute the FIRST sets for each of the non-terminals in this grammar. Show your result.

b) Compute the FOLLOW sets for each of the non-terminals in this grammar. Show your result.

c) Using your results from (i) and (ii), construct the predictive parsing table for this grammar. When indicating which productions should be performed, please use our numbering system from above. Show your result.

4. a) Consider the following grammar [5]

$S \rightarrow id\ [E] := E$

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid id$

Is the behavior of the ***SLR(1)*** parser and ***LR(1)*** parser identical for all strings?

b) Consider the following (already augmented) grammars for the language $a^*$. [5]

$S \rightarrow A$

$A \rightarrow Aa \mid \varepsilon$

and

$S \rightarrow A$

$A \rightarrow aA \mid \varepsilon$

Both of these grammars are ***SLR(1)***. However, the ***SLR(1)*** parser for one of these grammars will use ***O(n)*** space in its parsing stack when run on the string $a^n$, while the other parser will only use ***O(1)*** stack space.

Identify which grammar's parser uses ***O(n)*** stack space and which grammar's parser uses ***O(1)*** stack space. Justify your answer by making specific references to how an SLR(1) parser for each grammar parses strings of the form $a^n$. In other words, even if you can figure out why one parser uses ***O(n)*** stack space, you should still explain why the other parser uses only ***O(1)*** stack space.

5. Consider the following grammar for a very small csh-like language.

P → S; P | S;

S → A | F

A → id := E

E → E + id | id | int

F → **foreach** id **in** ( L ) **do** { P }

L → L int | int

A program consists of a sequence of statements. A statement can be an assignment statement (A) or a foreach loop statement (F). The foreach loop has a loop list L. L is a list of integers. In each iteration, an object from the loop list is assigned to the loop identifier (id) and P in the loop is executed. The following is a sample program for this csh-like language.

```
total := 0;
foreach num in (11 15 302 287 19 8) do
{ total := total + num; }
```

a) This language has a semantic rule: The loop identifier should not be modified in the loop body. Use an attributed grammar to check the semantic rule given above. In other words, you should add attribute rules to the given grammar to check and report error when there is semantic rule violation. Note that you can assume that the name of the id can be retrieved using id.name (which is a character string). An attribute can be an item or a list of items. When writing your answer, try to use synthesized S-attributed rules and/or L-attributed rules. You are not allowed to use global attributes.

b) If your attributed grammar contains L-inherited attributes, it is possible to use stack based technique to evaluate it during parsing. If so, then discuss how to do it.

6. Consider the following grammar for specifying binary trees (in linearized form):

***BinTree*** → ***(num BinTree₁ BinTree₂)*** | ∈

Extend the above grammar by defining a translation scheme such that a depth-first left-to-right traversal of a parse tree (denoting a binary tree) annotated with semantic actions will entail checking if the binary tree is ordered. A binary tree is ordered if the values of the numbers of the first subtree (***BinTree₁***) are

less than the value of the current number and the values of the numbers of the second subtree ($BinTree_2$) are greater than the value of the current number. For instance, (2 (1 nil nil) (3 nil nil)) is ordered but (1 (2 nil nil) (3 nil nil)) is not. Note that you can use any language constructs (such assignments, if-then-else, expressions, boolean expressions etc.) for writing the semantic rules.

7. Generate a 3AC for the given program (Assume that there is no syntax errors). Apply short circuiting of the code for the logical – if expression in Case 2. The dimension of the a-array is a[40][40] and the width is 4.

```
int i=0,j=0;

switch(a+b)

{
  Case 1: {x=x+1;  y=y+2; break;}

  Case 2: {

        if ( (x ! = y) and (x >=y ) )

          {
             a=a-2; b=b-2, c=c-2;
          }
        else
          {
             a=a+2; b=b+2, c=c+2;
          }
           break;

Case  3: {
          for (i=0; i<=40; i++)
           {
             for(j=0; j<=40; j++)
               {
                  a[i] [j] = 0;
               }
           }
          break;
         }
 default:
        {
          a=a+b; break
        }
}
```

8. Consider the code depicted below for a function with integer local variables i, a, b, c, d and an integer input parameters p and n.

int i,a,b,c,d;

For this code fragment determine:

i. The Data Flow Graph (DFG).

ii. The dominator tree and the loops (identify the back edges).

iii. Use constant propagation to the statements in the body of the loop identified in (ii).

Are the statements in line 8 and 16 loop invariants? Explain and describe where can they be moved to if they can in fact be moved outside any of the loops.

| | | |
|---|---|---|
| 1: | | i=0; |
| 2: | | a=1; |
| 3: | | b=i*4; |
| 4: | | c=a+b; |
| 5: | L1: | If( i>n)goto L2 |
| 6: | | c=a+1; |
| 7: | | i=i+1; |
| 8: | | b=i*4; |
| 9: | | If( c<=p) goto L3 |
| 10: | | e=1; |
| 11: | | c=e-b; |
| 12: | | a=e; |
| 13: | | goto L4 |
| 14: | L3: | d=2; |
| 15: | | c=d+b; |
| 16: | | a=d; |
| 17: | L4: | goto L1 |
| 18: | L2: | return; |

9. a) Consider the following three address code in a basic block. **[5]**

(1) read (a, b);

(2) c := a + b;

(3) d := a – b;

(4) c := c + d;

(5) a := a – b;

(6) e := a * c;

(7) print (e);

(a) Generate machine code based on the register assignment.

(b) Build a DAG for the basic block.

b) Consider the following three address code in a basic block. **[5]**

(1) t1 = j – 1
(2) t2 = 4 * t1
(3) temp = A[t2]
(4) t3 = j
(5) t4 = j + 1
(6) t5 = 4 * t3
(7) t6 = A[t5]
(8) t7 = j – 1
(9) t8 = 4 * t7
(10) A[t8] = t6
(11) t9 = j
(12) t10 = j + 1
(13) t11 = 4 * t9
(14) A[t11] = temp

Perform the following optimization techniques on the above code

(a) Copy propagation

(b) Dead code elimination

10. With a suitable example, write a short notes on the following:

(a) Instruction Scheduling

(b) Software Pipelining

⇔⇔⇔