University of Chittagong
Department of Computer Science and Engineering
7^{th} Semester B.Sc. Engineering Examination 2020
CSE 711: Compilers
Marks: 52.5 Time: 4:00 Hours

[The figures in the right hand margin indicate full marks. Answer any **Three** questions from **Section-A** and any **Three** questions from **Section-B**]

Section - A

1. (a) What tasks are performed in the *front-end* and *back-end* of a compiler, and why? How are the *phases* related to the *passes*? (4)

   (b) How can immediate left-recursion be eliminated? Illustrate how left-recursion involving derivation of multiple steps can be eliminated using the following grammar: (3)

$$S \to Pa \mid q$$
$$P \to Pc \mid Sd \mid \epsilon$$

   (c) Introduce the error recovery strategies in syntax analysis. (1.75)

2. Consider the following grammar:

$$S \to a\,S\,b\,S \mid b\,S\,a\,S \mid \epsilon$$

   (a) Compute FIRST and FOLLOW. (3)

   (b) Construct a predictive parsing table. (4)

   (c) Is the grammar LL(1)? (1.75)

3. (a) Consider the following grammar:

$$S \to (\,L\,) \mid id$$
$$L \to L\,,\,S \mid S$$

   i. Construct the LR(0) items. (1.75)

   ii. Compute the LR(0) states. (5)

   (b) Differentiate between synthesized and inherited attributes. (2)

4. (a) Write a Yacc/Bison program that takes boolean expressions as input using following grammar and produces the truth value of the expressions. (5)

```
bexpr -> bexpr or bterm | bterm
bterm -> bterm and bfactor | bfactor
bfactor -> not bfactor | ( bexpr ) | true | false
```

   (b) What is the role of symbol table in compilers? Remark briefly on the key technical aspects in creating and managing the *symbol table*. (3.75)

Please go on to the next page...

Section - B

5. (a) When can a compiler's implementation of a language be called *strongly typed*? What is *widening* and *narrowing* in type conversion? (2)

   (b) Given an LR(k) item $[A \rightarrow \alpha \,.\, \gamma, \delta]$ from a production $A \rightarrow \beta$, describe what A, α, γ and δ signify. (3)

   (c) Employing the following Action/Goto table (3.75)

State	Action		Goto	
	c	\$	S	T
0	Shift 1	Reduce $T \rightarrow \epsilon$	2	3
1	Shift 3	Reduce $S \rightarrow c$	0	
2		Accept	3	0
3	Shift 1	Accept		1

   demonstrate the operation of a shift-reduce parser with the string "**c**". Show the stack contents, input and action at each step (assuming state 0 initially).

6. (a) Consider the following program in 3-address intermediate code

(1) $a = 2$	(1) $t_8 = j - 1$
(2) $b = 3$	(2) $t_9 = 4 * t_8$
(3) $t = a$	(3) $tmp = A[t_9]$
(4) $r_1 = 0$	(4) $t_{10} = j + 1$
(5) if $(t \leq 0)$ goto (9)	(5) $t_{11} = t_{10} - 1$
(6) $t_1 = 2 * a$	(6) $t_{12} = 4 * t_{11}$
(7) $r_1 = t_1 + r_1$	(7) $t_{13} = A[t_{12}]$
(8) goto (5)	(8) $t_{14} = j - 1$
(9) $t = b$	(9) $t_{15} = 4 * t_{14}$
(10) $r_2 = 0$	(10) $A[t_{15}] = t_{13}$
(11) if $(t \leq 0)$ goto (15)	(11) $t_{16} = j + 1$
(12) $t_2 = 7 * b$	(12) $t_{17} = t_{16} - 1$
(13) $r_2 = t_2 + r_2$	(13) $t_{18} = 4 * t_{17}$
(14) goto (11)	(14) $A[t_{18}] = tmp$
(15) $t_3 = r_1 + r_2$	
(16) $r = 8 * t_3$	

   i. Indicate where new basic blocks start. For each basic block, give the line number such that the instruction in the line is the first one of that block. (3)

   ii. Give names $B_1, B_2, \ldots$ for the program's basic blocks in the order the blocks appear in the given listing. Draw the control flow graph making use of those names. (3.75)

   (b) Construct the DAG for the basic block (2)

   (1) $d = b * c$
   (2) $e = a + b$
   (3) $b = b * c$
   (4) $a = e - d$

7. (a) What is the difference between a parse tree and an abstract syntax tree (AST)? (2)

   (b) Introduce the idea of semantics preserving (local) optimizations in code optimization. (3)

   (c) Illustrate how *triples*, *quadruples* and *static single assignment* (**SSA**) form are different in representing intermediate codes. (3.75)

8. Consider the following merge-sort program.

```
void mergesort(int a[], int i, int j)
{
    int mid;
    if (i < j)
    {
    mid = (i+j)/2;
    mergesort(a, i, mid);     // left recursion
    mergesort(a, mid+1, j);   // right recursion
    merge(a, i, mid, j);      // merging of two sorted sub-arrays
    }
}
void merge(int a[], int p, int q, int r)
{
        /* Combine the elements back in a[p . . r] by merging

        the two sorted subarrays a[p . . q] and

        a[q+1 . . r] into a sorted sequence. */


        . . .
}

main()
{
        int a[11] = { 10, 14, 19, 26, 27, 31, 33, 35, 42, 44, 0 };
        mergesort(a, 0, 10);
}
```

   (a) Define following terms: activation tree and activation record. (2)

   (b) Show the complete activation tree for the given *merge-sort* program. (3)

   (c) Draw the activation records for the given *merge-sort* program. (3.75)