22 April 2019 **CS335: END SEMESTER EXAM** Total Marks: 125

NOTE:

- Presenting your answers properly is your responsibility. You lose credit if you can not present your ideas clearly, and in proper form. Please DO NOT come back for re-evaluation saying, "What I actually meant was ....".
- Be precise and write clearly. Remember that somebody has to read it to evaluate!
- If required, make suitable assumptions and write them along with your answer.

1. **(Parsing)** ..... **(Marks: 15)**
Consider the grammar below with terminals = $\{a, b, c, d, (,), [,]\}$ and nonterminals = $\{S, A, B, C, D, X\}$. Is the grammar SLR(1)? Is it LALR(1)? Is it LR(1)? Justify your answer[1].

$$
\begin{aligned}
S &\rightarrow (\ X\ A\ a\ ) \\
&\mid [\ A\ C\ ) \\
&\mid (\ X\ B\ b\ ] \\
&\mid [\ B\ D\ ] \\
\\
A &\rightarrow c \\
\\
B &\rightarrow c\ B \\
&\mid c \\
\\
C &\rightarrow d \\
\\
D &\rightarrow d\ D \\
&\mid d \\
\\
X &\rightarrow \epsilon
\end{aligned}
$$

[1] One way you can do it is by showing a state having conflicting items if your answer is NO, and by constructing a parse table if the answer is YES.

2. **(Run-time environment)** ..... **(Marks: 25)**

Consider the following program in a C++-like language, where parameters passed by reference use & (e.g. `int& a`):

```
void main (){
    int p=0, q=5;

    void S(int& a, int* b){
        int z=0,x=1;

        void Q(){
            int k=0,v=0;

            int P(int y, int z){
               int i,j;
               //...
            };/*End of P*/

            //...
            if (k) S(a,&a); else v = P(*b,a);
            //...
        } /*End of Q*/

        Q();
    }; /*End of S*/

    S(p, &q);
} /*End of main*/
```

Assume that the activation record has the space for the following fields (not necessarily in the same order): (i) parameters, (ii) local variables, (iii) (old) stack pointer (SP), (iv) return address (RA), (v) static link (SL), (vi) return value (RV).

For this program

(a) Show the activation records of `main`, `S`, `Q` and `P` when the execution is at the call chain

$$\texttt{main} \rightarrow \texttt{S} \rightarrow \texttt{Q} \rightarrow \texttt{P}$$

(b) Write the compilation of the boxed `if ...` statement in pseudo assembly language[2]. Comment the code to help correction, for example which fragment of code is setting which field of the activation record.

[2]You can use either x86-like or MIPS-like assembly notations, but mention which one are you using.

3. **(Aho-Johnson Algorithm)** ..... **(Marks: 25)**

Consider the following declaration in global scope of a program in C-like language:

```
struct {
    float f;
    int a[5];
    int *p;
} s;
int j, k;
```

The program contains an expression `s.a[j] + *(s.p) + k`. We plan to use Aho-Johnson algorithm to generate code for the expression on a machine having two general purpose registers, r_1 and r_2. The instructions of the machine, with their respective costs, are as follows:

Sl#	instruction	cost
1.	$r \leftarrow$ constant	1
2.	$r \leftarrow mem$	1
3.	$mem \leftarrow r$	1
4.	$r_i \leftarrow r_i \; op \; r_j$	2
5.	$r_i \leftarrow r_i \; op \; mem$	3
6.	$r_i \leftarrow ind \; r_i$	1
7.	$r_i \leftarrow ind(r_i + mem)$	4

Assume each of float, int and addresses use 4 bytes each.

(a) Treating + as left-associative, draw the expression tree for the given expression.

(b) Label each node with the array of costs. Note that the instructions have different costs. Circle the cost at each node that gives the minimum cost for the root[3].

(c) Show regset and memset for each node for the choice that gives the minimum cost for the root.

(d) Generate optimum code for the expression using Aho-Johnson algorithm.

[3] if more that one choices give the same minimum cost, chose any one of them.

4. **(Sethi-Ullman Algorithm)** ..... **(Marks: 35)**

Consider writing compiler for a Target machine having

- a load instruction, $r \leftarrow m$
- a store instruction, $m \leftarrow r$
- Binary operation involving two registers, $r_i \leftarrow r_j \ op \ r_i$
- Binary operation involving a memory and a register, $r_i \leftarrow m \ op \ r_i$
- Conditional Jump `if` $(r == 0)$ `goto L`
- Unconditional jump: `goto L`

Here `L` is a label of some instruction.
Note that the memory argument is the FIRST (left) argument for *op*.
The source language includes an **if-then-plus** expression, having an operator called *itp*:

$$itp \ \ e_1 \ \ e_2 \ \ e_3$$

The semantics of *itp* are as follows:

(a) First e_1 is evaluated, say its value is v_1.
(b) If the value v_1 is 0, e_3 is evaluated and its value is the value of the expression.
(c) Otherwise ($v_1 \neq 0$), e_2 is evaluated (say value v_2) and the value of the expression is $v_1 + v_2$.

Design a variation of the Sethi-Ullman algorithm for the desired compiler for the given target machine. In particular:

(a) Describe the changes in the labeling algorithm.
(b) Describe the changes in the function **gencode**.
(c) For expression tree shown next, show the labels at different nodes. The leaves in the tree are two character names (ma, mb, mc, ...) as memory locations.
(d) Show the code generated for the expression tree assuming that there are 3 registers R_0, R_1 and R_2. Add comments indicating which code fragments correspond to which sub-expressions, to help correction.