# 2014 Midterm Exam for Compiler (Totally 4 pages)

## (by Prof. Jenq-Kuen Lee)

1. (10%) Manually execute the following program

```
program parameter-passing;
   var i: integer;
         a: array [1..3] of integer;
   procedure    mess(v : integer);
      var i: integer;
    begin                            t= 1
            i:= 1;                   a[i]=a[i]+1
            v := v + 1;              a(t)=12
            a[i] := 12;              t= 3
            i := 3;                  a[i]=a[i]+1;
            v := v +1;
      end;
 begin
    for i:= 1 to 3 do a[i] := 9;     a[1]=9
    a[2] := 5;                       a[2]=5
    i := 2;                          a[3]=9
    mess(a[i]);
    . . .            <--------    Observation Point 1
 end.
```

(a) If by assuming Call-by-Text, what's the value in the array a and the variable i in the observation point 1 of the program?
(a[1]=?,a[2]=?,a[3]=?,i=?)

(b) If by assuming Call-by-Name, what's the value in the array a and the variable i in the observation point 1 of the program?
(a[1]=?,a[2]=?,a[3]=?,i=?)

2. (20%) Explain the following concepts?

(a) Why is a left-recursion grammar not in LL(1)?

(b) Explain how to decide if a grammar is in LL(k) ?

(c) Discuss the difference among LL(0), LL(1), and LL(2).

(d) Explain the techniques of left factoring in LL(1) grammar writing.

3. (20%) If we use BNF form to write a grammar for an arithmetic expression includes ``*" (multiplication), ``#" (exponential operators), ``+" (addition), "%" (mod), and parenthesis. We get a grammar below:

```
E ->  E  *  E
E ->  E  #  E
E ->  E  +  E
E ->  E  %  E
E ->  ( E )
E ->  Number
```

Assume the precedence order from the highest to the lowest is parenthesis, ``\#", "%", ``*", ``+". The exponential operation is right associate, and all other operators are left associate.

(a) Re-Write the above grammar into an un-ambiguous grammar following the given precedences and associativity.

(b) Is the grammar generated in (a) a LL(1) grammar? If it's not a LL(1) grammar, try to convert it into a LL(1) grammar?

(c) To use the concept of selection set to explain why the grammar you generated in (b) is a LL(1) grammar.

(d) Write a C program for the top-down recursive parser of the LL(1) grammar generated in (b).

$(aa|bb)^* b (aa)^*$ $(aa)|b$

4. (20%)

(a) To write a Lex-style regular expression to represent the syntax of the "Variable Name" in C language.

(b) Write a Lex Program that copies a C program, replacing all instance of int by float. In addition, please print out those replacements happen in which lines.

$(aa)^* b (aa)^*$

5. (10%) Write the regular expression for the following language.

All strings of a's and b's with even number of a's and odd number of b's.

0算偶數

6. (20%) (a) Describe the language denoted by the following regular expressions.

$$a^*|ab$$

(b) Construct nondeterministic finite automata for the regular expression above.

(c) Construct the DFA (deterministic finite state automata) for the machine generated in (b)?

(d) Construct the minimum-state DFA for the machine generated in (c).

7. (10\%) (a) Is the following grammar a LL(1) grammar?

S, E, and F are nonterminals, and ``(``, ``)", ``+", and ``a" are terminals in the grammar below.

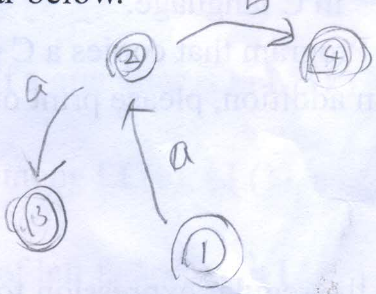
S -> E

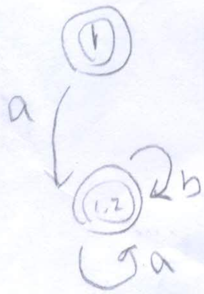
E -> ( a + a )

| F

F -> ( a )

(b) Convert the grammar in (a) into LL(1) if it's not a LL(1) yet.





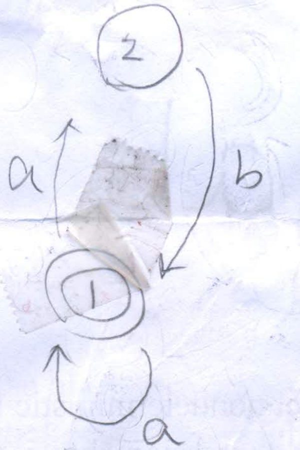
E→ E+T | T

T→ T*P | P

P→ F#P | F

F→ (E) | num

| | aε* | bε* |
|---|---|---|
| 1 | 1,2 | — |
| 1,2 | 1,2 | 1,2 |



E→ TE'

TE'→ +TE' | ε

T→ PT'

PT'→ *PT' | ε

P→ F#P | F

F→ (E) | num