School of Mathematical and Computational Sciences
Indian Association for the Cultivation of Science

Compiler Construction: COM 5202

Tutorial V (12 February, 2025)

M. Sc Semester IV: 2024-2025 *Instructor:* Goutam Biswas

Exercise 1. *Variable declarations in a programming language are as follows:*

```
a, b[10], c[10,5] : int ;
x[5,10,15], y[20], z : float;
p[100], q[200,10] : char
```

An array may have any number of dimensions. Different tokens are: integer constants (ic), identifiers (id), type names (T) and {, [ ] : ;}
Give descriptions of non-empty sequence of variable declarations in the following forms.

(a) *in Syntax diagram,*

(b) *in Context Free Grammar, and*

(c) *in Backus-Naur Form.*

Exercise 2. *Consider the following CFG G with E as the start symbol.*

$$\begin{aligned} E &\rightarrow E + T \mid T \\ T &\rightarrow T\ F \mid F \\ F &\rightarrow F * \mid B \\ B &\rightarrow 0 \mid 1 \end{aligned}$$

(a) *Draw the parse tree corresponding to the string* $0{+}1^*01$ *using the grammar G.*

(b) *Remove left-recursion from the grammar G to get an equivalent grammar G_1.*

(c) *Draw the parse tree corresponding to the string* $0{+}1^*01$ *using the grammar G_1.*

(d) *Can you think of a method to construct the Thompson NFA of* $0 + 1^*01$ *using the parse tree of ((a))?*

Exercise 3. *Give a CFG for the language*

$$L = \{0^a 1^b 2^c : \ a, b, c \in \{0\} \cup \mathbb{N} \text{ and } a = b \text{ or } b = c\}.$$

Is your grammar unambiguous?

Exercise 4. *Consider the following grammar and answer the questions:*

$$\begin{aligned} S &\rightarrow A\ a \mid b \\ A &\rightarrow A\ c \mid S\ d \mid \varepsilon \end{aligned}$$

(a) *Remove the ε-production. Note that ε does not belong to the language.*

(b) *Substitute the right-hand sides of the production rules of S in the rules of A.*

(c) *Remove left-recursion from the rules of A.*

Exercise 5. *Give a context-free grammar for Boolean expressions where the terminals are { true, false, and, or, not,), (}. Both and and or operators are left-associative. The precedence relation is or < and < not. true and false are Boolean constants. The parenthesis are used to overrule precedence.*

Exercise 6. *Consider the following context-free grammar: the start symbol is P, the non-terminals are $\{P, DL, D, TY, VL, CL, C, PC, RC, AC, E, T, F\}$, the terminals are* { `zah, flt, print, input`, *id, ic, fc,* {, }, :, =, +, −, ∗, /), (}, *and the production rules are as follows.*

$$
\begin{array}{rcl}
P & \rightarrow & \{\ DL\ :\ CL\ \} \\
DL & \rightarrow & D\ DL \mid \varepsilon \\
D & \rightarrow & TY\ VL \\
TY & \rightarrow & \texttt{zah} \mid \texttt{flt} \\
VL & \rightarrow & id\ VL \mid id \\
CL & \rightarrow & C\ CL \mid C \\
C & \rightarrow & PC \mid RC \mid AC \\
PC & \rightarrow & \texttt{print}\ E \\
RC & \rightarrow & \texttt{input}\ id \\
AC & \rightarrow & id\ =\ E \\
E & \rightarrow & E\ +\ T \mid E\ -\ T \mid T \\
T & \rightarrow & T\ *\ F \mid T\ /\ F \mid F \\
F & \rightarrow & ic \mid fc \mid id \mid (\ E\ )
\end{array}
$$

Transform the grammar to an equivalent $LL(1)$ grammar (if possible). Compute the First() of every production rule and Follow() of nullable non-terminals to justify that the transformed grammar is indeed $LL(1)$.

Exercise 7. *Consider the grammar of* **Ex.6**. **[10]**

The regular expression for id is `[a-zA-Z][a-zA-Z0-9]*`, *except the key words. For integer constants (ic) it is* `0|[1-9][0-9]*`, *and for the floating-point constants (fc) it is* `[0-9]+\.[0-9]+((e|E)[+-]?[1-9][0-9]?)?`.

Write flex specification for the set of terminals of the grammar. The name of the flex file should be `lex.l`. *The header file for it is* `y.tab.h`. *The header file defines some of the token values and the type of* `yylType`.
We take the ASCII codes as token values for all single character terminals.

A comment starts with `//` *and ends at the new-line character* `'\n'`. *The scanner ignores a comment, but notes the line number.*

```
// y.tab.h
#ifndef _Y_TAB_H
#define _Y_TAB_H

#define ID     300
#define IC     301
```

```
#define FC     302
#define FLT    305
#define ZAH    306
#define INPUT  309
#define PRINT  310
#define NOTOK  400

int yylex(void);
typedef union {
        int integer ;
        double real;
char *string;
} yylType;

#endif
```

The scanner function `yylex()` *will be called from* `main()` *in* `myLex.c` *(include* `y.tab.h` *in* `myLex.c`*). The* `main()` *function calls* `yylex()` *and prints the stream of tokens (shown in Input/Output). You may use the following* `Makefile` *for the process of compilation.*

The input is from the `stdin`*. You may write the input in a file and redirect it:*

```
$ ./a.out < data
```

Makefile:

```
objfiles = myLex.o lex.yy.o

a.out: $(objfiles)
cc $(objfiles)

myLex.o: myLex.c y.tab.h
cc -c -Wall myLex.c

lex.yy.o: lex.yy.c
cc -c lex.yy.c

lex.yy.c: lex.l y.tab.h
flex lex.l

clean:
rm a.out lex.yy.c $(objfiles)
```

Finally prepare a `.tar` *file using the following command.*

```
$ tar cvf <roll no>.5.tar y.tab.h lex.l myLex.c Makefile
```

Send the `.tar` *file to* `goutamamartya@gmail.com`*.*

Input

```
// This is the first comment
{
     zah a0 b10 c010
     flt x0a y01b
     :
          input a0
          x0a = 10.2e-2
          input c010
          c010 = 12*c010/(a0+5.6)
          print c010
// This is the second comment
}
```

Output

```
<{>
<zah> <ID, a0> <ID, b10> <ID, c010>
<flt> <ID, x0a> <ID, y01b>
<:>
<input> <ID, a0>
<ID, x0a> <=> <FC, 0.102000>
<input> <ID, c010>
<ID, c010> <=> <IC, 12> <*> <ID, c010> </> <(> <ID, a0> <+> <FC, 5.600000> <)>
<print> <ID, c010>
<}>
```