School of Mathematical and Computational Sciences

Indian Association for the Cultivation of Science

Compiler Construction: COM 5202

Tutorial VII (26 February, 2025)

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

Exercise 1. *Consider the following grammar and construct the canonical collection of $LR(0)$ items after augmenting it with a special start symbol. If the grammar is SLR, construct the corresponding parsing table.*

$$\begin{aligned} S &\rightarrow (\ L \ ) \\ S &\rightarrow a \\ L &\rightarrow L \ , \ S \\ L &\rightarrow S \end{aligned}$$

Exercise 2. *Consider the following ambiguous grammar of expressions. The associativity and precedence of operators are as usual.*

0	S	$\rightarrow$	E
1	E	$\rightarrow$	$E - E$
2	E	$\rightarrow$	E/E
3	E	$\rightarrow$	(E)
4	E	$\rightarrow$	$-E$
5	E	$\rightarrow$	ic

(a) *Build an $LR(1)$ DFA of viable prefixes and show that the grammar is not $LR(1)$.*

(b) *Consider states with shift-reduce conflicts. Can this be resolved using the associativity and precedence of operators? Show the parsing table entries with conflicts. Also show the modified table entries if it can be resolved using the associativity and precedence of the operators.*

(c) *Show instances of input and their parsing that take the parser to a conflicting states.*

Exercise 3. *Consider the grammar given in* **Tutorial 5, Ex. 6**.
Terminals: { `zah, flt, print, input`, *id, ic, fc,* {, }, :, =, +, −, ∗, /), (}.
Non-terminals: $\{P, DL, D, TY, VL, CL, C, PC, RC, AC, E, T, F\}$.
Start Symbol: P.

The production rules are,

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

You have already transformed it to an equivalent $LL(1)$ grammar, computed the First() of production rules and the Follows() of non-terminals.
*You also have written a scanner using flex (***Tutorial 5, Ex. 7***).*

Write a recursive descent predictive parser for the language of the grammar. Use the header file provided in **Tutorial 5, Ex. 7**.

```
// 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
```

Use the scanner of **Tutorial 5, Ex. 7**. *You may need to modify it a little bit. Use the following Makefile.*

```
objfiles = recursiveDescent.o lex.yy.o
```

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

recursiveDescent.o: recursiveDescent.c y.tab.h
                    cc -c -Wall recursiveDescent.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>.7.tar y.tab.h lex.l <file name>.c Makefile
```

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

Input

```
// This is the first comment
{
     zah a0 b0 c0
     flt x0 y0
     :
         input a0
         x0 = 10.2e-2
         input c0
         c0 = 12*c0/(a0+5.6)
         print c0
// This is the second comment
}
```

Output

```
$ ./a.out < d1
Accept
```

Input

```
// This is the first comment
{
     a0 b0 c0
     :
         input a0
}
```

Output

```
$ ./a.out < d2
'flt'/'zah' missing:3
```

```
T() error
D() error
DL() error
Reject
```

Input

```
     zah a0 b0 c0
     :
         input a0
}
```

Output

```
$ ./a.out < d3
'{' missing:1
Reject
```

A Sample Function

```
int VLdash(){
    getToken();
    if(token == ZAH || token == FLT || token == ':'){
       return 1;
    }
    if(token==ID){
       tokenPresent=0;
       if(VLdash()){
          return 1;
       }
       else {
             printf("VLdash() error\n");
             return 0;
       }
    }
    else {
         printf("'ID' missing:%d\n", line_count);
         return 0;
    }
}
```