School of Mathematical and Computational Sciences

Indian Association for the Cultivation of Science

Compiler Construction: COM 5202

Tutorial VI (11 February 2026)

M. Sc Semester IV: 2025-2026 *Instructor:* Goutam Biswas

Exercise 1. *Consider the grammar of the assignment of* **Tutorial V**. *There is a small change, the non-terminal* **IC** *is renamed as* **INC** *to avoid ambiguity with the terminal* **ic**. .

$$
\begin{array}{rcl}
P & \rightarrow & \textbf{prog}\ DLO\ \textbf{start}\ CL \\
DLO & \rightarrow & DLO\ DL \mid \varepsilon \\
DL & \rightarrow & TY\ VL \\
TY & \rightarrow & \textbf{int} \mid \textbf{real} \\
VL & \rightarrow & VL\ \textbf{id} \mid \textbf{id} \\
CL & \rightarrow & CL\ C \mid C \\
C & \rightarrow & INC \mid OC \mid AC \\
INC & \rightarrow & \textbf{in id} \\
OC & \rightarrow & \textbf{out}\ E \mid \textbf{out str} \\
AC & \rightarrow & \textbf{id :}\ E \\
E & \rightarrow & E + T \mid E - T \mid T \\
T & \rightarrow & T * F \mid T / F \mid F \\
F & \rightarrow & \textbf{ic} \mid \textbf{rc} \mid \textbf{id} \mid (\ E\ )
\end{array}
$$

(a) *Remove left recursions from the grammar.*

(b) *Compute First() of different production rules and non-terminals, Follow() of different non-terminals for the modified grammar.*

(c) *Is the modified grammar* $LL(1)$*?*

Exercise 2. **[10]**

Consider the modified grammar of Exercise 1(a):
Terminals: { **prog, start, int, real, in, out, ic, rc, id str, :**, +, −, ∗, /, (,) },
Non-terminals: { *P, DLO. DL, TY, VL, VL', CL, CL', C, INC, OC, AC, E, E', T, T', F* }
The start symbol is P.
The regular expressions are,
id: `[A-Z][\_a-zA-Z0-9]*`
ic: `0|[1-9][0-9]*`
rc: `[0-9]+\.[0-9]*((e|E)[+-]?[1-9][0-9]?)?`
str: `(\"[^\"]*\")`
Comment: `(\/\/.*\n)`

You already have the lexical analysis/scanner using flex:
Tutorial 5*:* `lex.l, y.tab.h`

Write a recursive descent predictive parser for the language of the modified grammar (Exercise 1(a)). A few changes in the `lex.l` *and* `y.tab.h` *files may be needed. A* `Makefile` *is as follows.*

```
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>.6.tar y.tab.h lex.l <file name>.c Makefile
```

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

A few input/output of the parser are as follows:

Input: `d0`

```
prog start out "Indian Associatian
for the Cultivation
of Science"
```

Output:

```
$ ./a.out < d0
Accept
```

Input: `d1`

```
prog start out 123
```

Output:

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

Input: `d2`

```
prog start out 123.456
```

Output:

```
$ ./a.out < d2
Accept
```

Input: `d3`

```
prog start out 123 - 4 * 5.2
```

Output:

```
$ ./a.out < d3
Accept
```

Input: d4

```
prog start out (123 - 4) * 5.2
```

Output:

```
$ ./a.out < d4
Accept
```

Input: d5

```
prog start out 123 - 4) * 5.2
```

Output:

```
$ ./a.out < d5
IN/OUT/ID missing: 1
Reject
```

Input: d6

```
prog start out (123 - 4 * 5.2
```

Output:

```
$ ./a.out < d6
')' missing:2
Reject
```

Input: d7

```
prog
  real A1 A2
  int B1 B2 B3
start
  in A1
  in A2
  B1: A1/A2
  out B1
```

Output:

```
$ ./a.out < d7
Accept
```

Input: d8

```
prog
  real A1 A2
  int B1 B2 B3
// start
  in A1
  in A2
  B1: A1/A2
  out B1
```

Output:

```
$ ./a.out < d8
ID missing: 5
Reject
```

Input: d9

```
// prog
  real A1 A2
  int B1 B2 B3
start
  in A1
  in A2
  B1: A1/A2
  out B1
```

Output:

```
$ ./a.out < d9
'prog' missing:2
Reject
```

Input: d10

```
prog
  real A1 A2
  int B1 B2 B3
start
  in A1
  in A2
  B1: A1/A2
  out B1
  B2: B1+2*A1/A2
prog
  real A1 A2
  int B1 B2 B3
start
  in A1
  in A2
  B1: A1/A2
  out B1
  B2: B1+2*A1/A2
```

Output:

```
$ ./a.out < d10
Accept
```

Input: d11

```
prog start  123 - 4 * 5.2
```

Output:

```
$ ./a.out < d11
IN/OUT/ID missing: 1
Reject
```

Exercise 3. *Transform each of the following grammars to an $LL(1)$ grammar (if possible). Show the parsing table of a predictive parser in each case where the transformation is possible.*

(a) $S \rightarrow 0\ S\ 1 \mid 0\ 1$

(b) $S \rightarrow (\ S\ )\ S \mid \varepsilon$

(c) $S \rightarrow S\ (\ S\ ) \mid \varepsilon$

(d) $S \rightarrow S\ +\ S \mid S\ S \mid (\ S\ ) \mid S\ * \mid a$

Exercise 4. *Following grammar is not $LL(1)$.*

$$S \rightarrow S\ S\ + \mid S\ S\ - \mid S\ S\ * \mid S\ S\ / \mid ic$$

where ic is the toke corresponding to an integer constant.

(a) Transform it to an equivalent $LL(1)$ grammar.

(b) Draw the parse tree for the input `7 10 2 / - 4 + 3 * 20 -` *using the transformed grammar.*

(c) Design the parse table for a predictive parser.