Computer Science and Engineering
IIIT Kalyani, West Bengal

Compilers Design Laboratory (Spring: 2017 - 2018)

3rd Year CSE: 6th Semester

Assignment - 5 *Marks:* 20
Assignment Out: 9^{th} *February, 2018* Report on or before: 9^{th} *March, 2018*

1. Consider the following *context-free grammar* G: the *non-terminals* are $N = \{$ *AE, BE, D, DL, E, F, ES, IOS, IS, NE, P, PE, RE, S, SL, T, TY, VL, WS* $\}$,
   the *terminals* are
   $\Sigma = \{ + - * / = < > ($ **( ) { } := ; and else end ic id if int do fc float not or print prog scan str then while** $\}$,
   the *start symbol* is P and the production rules are,

P	$\rightarrow$	**prog** DL SL **end**
DL	$\rightarrow$	D DL | ε
D	$\rightarrow$	TY VL **;**
TY	$\rightarrow$	**int** | **float**
VL	$\rightarrow$	**id** VL | **id**
SL	$\rightarrow$	S SL | ε
S	$\rightarrow$	ES | IS | WS | IOS
ES	$\rightarrow$	**id :=** E **;**
IS	$\rightarrow$	**if** BE **then** SL **end** |
		if BE **then** SL **else** SL **end**
WS	$\rightarrow$	**while** BE **do** SL **end**
IOS	$\rightarrow$	**print** PE | **scan id**
PE	$\rightarrow$	E | **str**
BE	$\rightarrow$	BE **or** AE | AE
AE	$\rightarrow$	AE **and** NE | NE
NE	$\rightarrow$	**not** NE | **{** BE **}** | RE
RE	$\rightarrow$	$E = E$ | $E < E$ | $E > E$
E	$\rightarrow$	$E + T$ | $E - T$ | T
T	$\rightarrow$	$T * F$ | T / F | F
F	$\rightarrow$	**(** E **)** | **id** | **ic** | **fc**

   Most of the terminals are self-explanatory. **id** is an (identifier), **ic** is an integer constant, **fc** is a floating-point constant, **str** is a string of characters, **:=** is an assignment etc.

2. Transform the grammar G to an equivalent $LL(1)$ grammar G_1 by removing *left recursion* and *left factoring*. Compute *First*$(A \rightarrow \alpha)$ for every production rule and *Follow(A)* of every non-terminal producing an ε.

   In the report clearly present this computation to establish that the grammar is $LL(1)$.

3. You already have the scanner written for these non-terminals in the assignment 3. The function `yylex()` of the scanner returns tokens. Attributes are available in global variables.

4. Write a C program to implement a *recursive descent predictive parser* for the modified grammar G_1. The parser calls `int yylex()` for token and uses the global variables if required.

   Given an input string x over the alphabet Σ, the parser either *accepts* or *rejects* the string depending on whether $x \in L(G) = L(G_1)$ or not. If there is an error, it may also print the *line number* and *character position* where the error has occurred.

5. Take the scanner files `myLex.c++` and `myLex.h` from the assignment 3. The name of the parser file should be `myRDPparser.c++`.

6. Prepare a `Makefile` to compile both the files. Finally prepare a `.tar` file (`$ tar cvf <rollNo>.tar Makefile myLex.c myRDPparser.c myLex.h`) and return the `.tar` file.