Programming Assignment 1 Due on Feb. 10 (Monday)

Implement an LR(1) parser, also known as *Shift-Reduce* parser, according to the following grammar and its parsing table.

1. $E \longrightarrow E + T$
2. $E \longrightarrow E - T$
3. $E \longrightarrow T$
4. $T \longrightarrow T * F$
5. $T \longrightarrow T \ / \ F$
6. $T \longrightarrow F$
7. $F \longrightarrow (E)$
8. $F \longrightarrow n$ where n is any positive integer between 0 and 32767.

- Let \$ be the symbol to end the arithmetic expression.
- If the entry contains two reduction rules, which one should be used will be determined by the terminal symbol popped from the stack.

	n	+	−	*	/	(	)	\$	E	T	F
0	shift 5					shift 4			1	2	3
1		shift 6	shift 6					accept			
2		$E \to T$	$E \to T$	shift 7	shift 7		$E \to T$	$E \to T$			
3		$T \to F$	$T \to F$	$T \to F$	$T \to F$		$T \to F$	$T \to F$			
4	shift 5					shift 4			8	2	3
5		$F \to$ n	$F \to$ n	$F \to$ n	$F \to$ n		$F \to$ n	$F \to$ n			
6	shift 5					shift 4				9	3
7	shift 5					shift 4					10
8		shift 6	shift 6				shift 11				
9		$E \to E + T$ $E \to E - T$	$E \to E + T$ $E \to E - T$	shift 7	shift 7		$E \to E + T$ $E \to E - T$	$E \to E + T$ $E \to E - T$			
10		$T \to T * F$ $T \to T \ / \ F$	$T \to T * F$ $T \to T \ / \ F$	$T \to T * F$ $T \to T \ / \ F$	$T \to T * F$ $T \to T \ / \ F$		$T \to T * F$ $T \to T \ / \ F$	$T \to T * F$ $T \to T \ / \ F$			
11		$F \to (E)$	$F \to (E)$	$F \to (E)$	$F \to (E)$		$F \to (E)$	$F \to (E)$			

To demonstrate your parser works correctly, your parser should print out the contents of the stack (from bottom of the stack to its top) followed by the contents of a queue for the remaining token string of the arithmetic expression whenever the stack is updated (i.e., after push or pop) according to the following format.

Let `(□:0)` denote the item at the bottom of the stack, where 0 indicates that the initial state is 0. At any moment, the state indicated in the top item of the stack is the current state. For example, if the top item is `(F=17:3)`, it indicates that the current state is 3, and the symbol is `F` with value 17. If the symbol, is an operator, there is no need for the value, and thus it is omitted from the printout, e.g. `(*:7)`.

Consider the following printout of parsing `17+20*15.`

```
java LR1 "17+20*15"

Stack:[(□:0)]    Input Queue:[17 + 20 * 15 $]
Stack:[(□:0) (n=17:5)]    Input Queue:[+ 20 * 15 $]
Stack:[(□:0) (F=17:3)]    Input Queue:[+ 20 * 15 $]
Stack:[(□:0) (T=17:2)]    Input Queue:[+ 20 * 15 $]
Stack:[(□:0) (E=17:1)]    Input Queue:[+ 20 * 15 $]
Stack:[(□:0) (E=17:1) (+:6)]    Input Queue:[20 * 15 $]
Stack:[(□:0) (E=17:1) (+:6) (n=20:5)]    Input Queue:[* 15 $]
Stack:[(□:0) (E=17:1) (+:6) (F=20:3)]    Input Queue:[* 15 $]
Stack:[(□:0) (E=17:1) (+:6) (T=20:9)]    Input Queue:[* 15 $]
Stack:[(□:0) (E=17:1) (+:6) (T=20:9) (*:7)]    Input Queue:[15 $]
Stack:[(□:0) (E=17:1) (+:6) (T=20:9) (*:7) (n=15:5)]    Input Queue:[$]
Stack:[(□:0) (E=17:1) (+:6) (T=20:9) (*:7) (F=15:10)]    Input Queue:[$]
Stack:[(□:0) (E=17:1) (+:6) (T=300:9)]    Input Queue:[$]
Stack:[(□:0) (E=317:1)]    Input Queue:[$]

Valid Expression, value = 317.
```

If the input expression is valid, print out the value of the valid expression as above. Hint: Whenever the parser obtains a nonterminal symbol from some reduction, its value should be computed and stored in the new item. Thus. you may design a certain data structure for every nonterminal symbol to store its value. At the end of the parsing, if correct, the *start symbol* `E` in the stack will have the value of the input arithmetic expression; in our example, 317.

Where to prepare your programs: Read carefully. If you have any confusion, ask.

Make directory ~`/IT327/Asg1` under your home directory and make sure it is not readable to everyone but yourself, where all programs for this assignment should be put under this directory, and don't make sub-directory here. Name your program as `LR1.java` and use the command line input to provide input expression to the parser. See the firt line in the example above. Note, the user doesn't have to provide `$` sign at the end of the input. Your parser should attach `$` sign automatically.

After you've finished your work, select a secret name, say "`peekapoo`" as an example (you should chose your own), and that will be your secret directory, and you should not be given to anyone except the instructor. Run the following bash script program:

```
bash /home/ad.ilstu.edu/cli2/Public/IT327/submit327.sh peekapoo
```

That's all you have to do to copy programs for me to grade. For physical submission, follow the guidelines, i.e., cover page, summary, source code(optional), output, folder, and so on.

Submission guidelines:

- Any form and any degree of plagiarism will receive 0 point.
- Put a few comment lines at the beginning of your program files, in which you should

clearly identify yourself and declare your copyright to the program.

- Any program with **syntax error receives 0 point on the programs part.** An incorrect program may receive as much as 70% of the credit but you have to describe a reasonable self-diagnosis to your program in your summary to get some partial credit.
- Submit all items as described in the following order.

  1. A cover page with the following information: 1) Student name, 2) Unix account's user name **(i.e. ULID)** and 3) secret directory name.

  2. A brief summary about the assignment and your approach to the problem. You may include the difficulties you had faced and how do you resolve the difficulties. This part will carry 10% of your credit.

  3. A hard-copy of a few direct output of your program, if any. You can use Unix's redirecting commands to redirect the output to some text files and print out the files. If the hard-copy of the direct outputs of your program is inconsistent to your program's design, you will receive 0 point.

  4. All items described above must be put in a letter-sized Manila folder with your name on it. Don't put them in an envelop.

- **Every program will be tested on our Unix server.** Don't email me your programs.