Programming Assignment 2
Due on Feb. 24, 2021, Wednesday, 11:55 PM

Use Java to 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 "+" and "-" be the operators for addition and substraction, and "*" and "/" be the operators for multiplication and division, respectively. 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, then there is no need for the value, and thus it is omitted from the printout, e.g. `(*:7)`.

I will test your program on Unix using command line to provide the input expression. Name your program as `LR1.java`. 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 1: 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.

Hint 2: You can design a class that can hold all information needed for the stack items. For example, `(T=300:9)` above, the object can tell it is a nonterminal T, its value is 300, the state is 9. Another example, `(*:7)` is a terminal symbol `*` and the state is 7. It is your design, use your technique learned from Java class to simplify your job. I expect to see you discuss your data structures in your report.

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

1. Let ∼ denote your home directory in your Unix account. Make directory ∼`/IT327/Asg2/`. All files related to this assignment should be prepared in this directory.
2. **Final Step on Unix Account.** After you've finished your work, or have decided that what you have is the final version for me to grade, select a secret name, say "`peekapoo`" as an example (you should chose your own), and that will be the secret directory, and should not give to anyone except the instructor. Run the following bash script program:

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

Note that your programs have to be in the required directory ~/IT327/Asg2/ as described in step 1 in order to let this script program correctly copy your programs into the secret directory, where your programs will be tested. Also, unix is case sensitive. Name you directory and program files correctly.

3. **Report.** You have to write up a report and submit a copy of the report through your ReggieNet. The report should include:

   - A cover page that contains assignment number, your name, ULID (not student ID) and **the name of the secret directory**.
   - A brief summary of the methods, algorithms and data structures, and the difficulties, if any, the project has faced and how to solve them.
   - The direct output of your programs on some nontrivial test inputs. The program code is not required.

4. **Total points:** 30 on program and 10 on report. Any program with syntax error receives 0 point on the programs part. An incorrect program may receive some partial credit no more then 70% (21) of the credit on the program part but you have to describe a reasonable self-diagnosis to your program in your summary. The score is based on the correctness and documentation of your program. **No late work will be accepted.**