Shift Reduce Parser Assignment

Design

1. You will implement the shift reduce parser discussed in lecture. Your parser will use your lexical analyzer from Assignment 1. The sentences in the language recognized by the parser are simple expressions. For example, **(sum + additional) * total** is a sentence of the language. Any simple expression using the tokens recognized by your lexical analyzer can be test input for your parser. Your lexical analyzer does recognize tokens not in the language; but that will not be a problem for this assignment. Ignore that your lexical analyzer recognizes the tokens: **int_lit**, **–**, and **/**. Assume the sentences processed by your parser only contain lexemes in the language.

2. It is reasonable to create a class called ShiftReduceParser for this assignment's parser. You must declare private all the data members of the ShiftReduceParser class. You must declare private all the methods in the ShiftReduceParser class, except for the constructor and the method beginParse. You can create any additional classes for your parser, as needed. The grammar and the parse table for the language are given on page 3.

3. Your parser will produce output like the following two examples.

   This first example is a sentence in the language: `(sum + additional) * total`

   ```
   Parser output:
   Next lexeme: (
   Next lexeme: sum
   Next lexeme: +
   Next lexeme: additional
   Next lexeme: )
   Next lexeme: *
   Next lexeme: total
   Next lexeme: EOF
   The sentence is syntactically correct.
   ```

   This second example is not a sentence in the language: `x * y + + i`

   ```
   Parser output:
   Next lexeme: x
   Next lexeme: *
   Next lexeme: y
   Next lexeme: +
   Next lexeme: +
   The sentence is not syntactically correct.
   ```

   The blank entries in the parse table (page 3) represent syntax errors. During the parse, if the shift reduce parser accesses a blank entry in the table, then the parser has reached a syntax error in the sentence. As you can see from the second example above, you will have the parser print out that the sentence is not syntactically correct and then terminate.

4. Create a driver class and make the name of the driver class **Assignment2** containing only one method: `public static void main(String args[]).`
   The main method receives, via the command line arguments, the name of the file that the parser processes. In the main method, create the ShiftReduceParser object and then call its method beginParse to parse the sentence in the file.

   I compile and run your program via the command line using the Java JDK. Therefore, the command I type to execute your program is **java Assignment2**. I'll test your program with my own example file.

   For example, if the name of the example file is test.in then the command to run the program is:

   ```
   java Assignment2 test.in
   ```

5. You must declare public each class you create which means you define each class in its own file.

6. You must declare private all the data members in every class you create.

7. You **do not** need and **cannot** use **extends** in this assignment.

8. **Tip:** Make your program as modular as possible, not placing all your code in one .java file. You can create as many classes as you need in addition to the classes described above. Methods being reasonably small follow the guidance that "A function does one thing and does it well." You will lose a lot of points for code readability if you don't make your program as modular as possible. But, do not go overboard on creating classes and methods. Your common sense guides your creation of classes and methods.

9. Do **NOT** use your own packages in your program. If you see the keyword **package** on the top line of any of your .java files then you created a package. Create every .java file in the **src** folder of your Eclipse project, if you're using Eclipse.

10. Do **NOT** use any graphical user interface code in your program!

11. Do **NOT** type any comments in your program. If you do a good job of programming by following the advice in number 8 above then it will be easy for me to determine the task of your code.

Simple Expression Grammar:

```
1. <E> → <E> + <T>
2. <E> → <T>
3. <T> → <T> * <F>
4. <T> → <F>
5. <F> → ( <E> )
6. <E> → ident
```

Simple Expression Parse Table:

	Action						Goto		
State	IDENT	+	*	(	)	EOF	<E>	<T>	<F>
0	S5			S4			1	2	3
1		S6				accept			
2		R2	S7		R2	R2			
3		R4	R4		R4	R4			
4	S5			S4			8	2	3
5		R6	R6		R6	R6			
6	S5			S4				9	3
7	S5			S4					10
8		S6			S11				

Grading Criteria

The total assignment is worth 20 points, broken down as follows:

1. If your code does not implement the task described in this assignment then the grade for the assignment is zero.
2. If your program does not compile successfully then the grade for the assignment is zero.
3. If your program produces runtime errors which prevents the grader from determining if your code works properly then the grade for the assignment is zero.

If the program compiles successfully and executes without significant runtime errors then the grade computes as follows:
Followed proper submission instructions, 4 points:
1. Was the file submitted a zip file.
2. The zip file has the correct filename.
3. The contents of the zip file are in the correct format.
4. The keyword **package** does not appear at the top of any of the .java files.

Program execution, 4 points:
- Program input, the program properly reads, processes, and uses the input.
- Program output, the program produces the correct results for the input.

Code implementation, 8 points:
- The driver file has the correct filename, **Assignment2.java** and contains only the method **main** performing the exact tasks as described in the assignment description.
- The code performs all the tasks as described in the assignment description.
- The code is free from logical errors.

Code readability, 4 points:
- Good variable, method, and class names.
- Variables, classes, and methods that have a single small purpose.
- Consistent indentation and formatting style.
- Reduction of the nesting level in code.

Late submission penalty: assignments submitted after the due date are subjected to a 2 point deduction for each day late.

Late submission policy: you **CAN** submit your assignment early, before the due date. You are given plenty of time to complete the assignment well before the due date. Therefore, I do **NOT** accept any reason for not counting late points if you decide to wait until the due date (and the last possible moment) to submit your assignment and something happens to cause you to submit your assignment late.

Submission Instructions

Go to the folder containing the .java files of your assignment and select all (and **ONLY**) the .java files which you created for the assignment in order to place them in a Zip file. The file can **NOT** be a **7z** or **rar** file! Then, follow the directions below for creating a zip file depending on the operating system running on the computer containing your assignment's .java files.

Creating a Zip file in Microsoft Windows (any version):

1. Right-click any of the selected .java files to display a pop-up menu.
2. Click on **Send to**.
3. Click on **Compressed (zipped) Folder**.
4. Rename your Zip file as described below.
5. Follow the directions below to submit your assignment.

Creating a Zip file in Mac OS X:

1. Click **File** on the menu bar.
2. Click on **Compress ? Items** where ? is the number of .java files you selected.
3. Mac OS X creates the file **Archive.zip**.
4. Rename **Archive** as described below.
5. Follow the directions below to submit your assignment.

Save the Zip file with the filename having the following format:
your last name,
followed by an underscore _,
followed by your first name,
followed by an underscore _,
followed by the word **Assignment2**.
For example, if your name is John Doe then the filename would be: **Doe_John_Assignment2**

Once you submit your assignment you will not be able to resubmit it!
Make absolutely sure the assignment you want to submit is the assignment you want graded.
There will be **NO** exceptions to this rule!

You will submit your Zip file via your CUNY Blackboard account.
The only accepted submission method!

Follow these instructions:

Log onto your CUNY BlackBoard account.
Click on the CSCI 316 course link in the list of courses you're taking this semester.
Click on **Content** in the green area on the left side of the webpage.
You will see the **Assignment 2 – Shift Reduce Parser Assignment**.
Click on the assignment.
Upload your Zip file and then click the submit button to submit your assignment.

Due Date: Submit this assignment by Thursday, May 14, 2020.