Computer Science Courses

Project 0: Building a simple tokenizer / scanner

- Project starts on: Wed, Aug 26, 2015
- Project due date: **Wed, Sep 9, 2015**

> Please read the description very carefully. You will stand to lose a significant number of points if you do not follow these instructions perfectly in your projects!

Description

The goal of this simple project is to get you acquainted with the submission procedure. Going forward, we want to be very efficient and accurate with our evaluations and thus would need your co-operation with regards to submissions.

Project skeleton

Your project skeleton is project0.zip. Download the files and start working

Do **not** change the original function names and input parameters in project skeleton. You are **not** allowed to refactor the project skeleton. In order to get full credit, you **must** use the same function names, same input and output parameters.

Token Class

A token (as modeled in `Token.java`) is created from a piece of text and understood as a syntactic entity. The following is a list of tokens that you are to construct from the input text. Tokens are grouped by attributes. Each token has two fields, namely

- token: An integer identifying the token,
- tokenValue: One of – a string, an integer, or a float

Token Specifications:

The following are token that come from single characters. The character to be translated into a token is shown first, quoted, token and token value are shown next:

```
quoted char   token, value
   '?'        20,0
   '('        21,0
   ')'        22,0
   '+'        23,0
```

```
'-'        24,0
'*'        25,0
'/'        26,0
'.'        27,0
';'        28,0
'='        29,0
```

Blanks, tabs and CR are considered white space and are ignored. There are two additional token, namely integer and floating point numbers. Integers are token 11 with the tokenValue the value of the integer; floats are token 12 with tokenValue the value of the float in the input.

Note that the input is processed up to and including the '?' (token 20). The remainder of that line, following the '?', would be ignored. Note also that the minus is not taken as the sign of the following integer or float .We will have to say more about the minus sign later-on. For now, any minus is considered a separate token.

Example

```
Input: 34+23 * 15 +- 10 // ** =?=

Output: (11,34)(23,0)(11,23)(25,0)(11,15)(23,0)(24,0)(11,10)(26,0)(26,0)(25,0)(25,0)(29,0)(20,0)
```

The output formatting (parens, comma etc) is not the internal representation.

Note It might be useful when developing this project to use symbolic constants in place token values. For example, when checking for a '+', look for a predefined constant called "TOKEN_PLUS", which has the value 23, instead of checking against the integer 23. Using these predefined symbolic constants be can very useful for readability and debugging your code.

Tasks

You are to implement the three empty methods inside `Project0.java`:

1. **Task 1** `read_input`: Continues to read input from user until a '?' is entered. A single line of input can contain no tokens or many tokens and there can be multiple lines of input.</todo>
2. **Task 2** `create_tokens`: Split the input into the separate tokens
3. **Task 3** `print_tokens`: Print all the tokens before and including the '?' token Print tokens from list in the following way, (token,tokenValue)"

Test Cases

```
//Note: hit Enter after each token

Input 1: -1.2++**/.()30=?
Expected output: (24,0)(12,1.2)(23,0)(23,0)(25,0)(25,0)(26,0)(27,0)(21,0)(22,0)(11,30)(29,0)(20,0)

Input 2: 34+23 * 15 +-10 // ** =?=
Expected output: (11,34)(23,0)(11,23)(25,0)(11,15)(23,0)(24,0)(11,10)(26,0)(26,0)(25,0)(25,0)(29,0)(20,0)

Input 3: 34+23 * 15
      +-10 / * ?
Expected output: (11,34)(23,0)(11,23)(25,0)(11,15)(23,0)(24,0)(11,10)(26,0)(25,0)(20,0)

Input 4: 1.1+32.02+-?
Expected output: (12,1.1)(23,0)(12,32.02)(23,0)(24,0)(20,0)
```

```
Input 5: 1.+-?
Expected output: (11,1)(27,0)(23,0)(24,0)(20,0)
```

General Instructions

1. We have provided skeleton code and ancillary files. Your changes will have to be on top of what has already been given to you. Writing your own code instead will cause you to lose points.
2. Do not change file structures, file names, folder names or folder structures.
3. You are free to create as many Java classes and files as you wish, and as many functions as you wish. But do not change what is already written in the skeleton. You can also create test scripts if you like.
4. Always, you will need to write code only when you see "TODO" markers. Whenever there is a TODO marker, we will make sure to write some extra comments under the TODO markers and this will tell you what you really need to do. What this means is that parts of the task will already be done for you in the skeleton. Just to re-iterate, you can create extra classes/functions that you might want to use/call.
5. Please carefully note the input and output formats. Also note the exact casing of the alphabets and the exact number of spaces and newlines to be added at various points in the output. Even one missing or extra character will cause you to lose points in the projects. In this project, the skeleton code is doing this for you; you only need to focus on the data structures and algorithmic aspects. We expect that this will be the case with most projects.
6. Different people write/edit code in different ways, so:
    - If you are going to be using Eclipse, read this.
    - If you are going to be using Dr. Java, read this.
    - If you are going to be coding in a low-tech manner using system editors like Vim, read this.
    - If you are going to be using something else, you are on your own.

Project Specific Instructions

Testing: You will begin by using the sanity test script provided to you. It is present in project0test.zip. Place `sanity_test` and `data` in the same directory as `Project0.java`. Assuming you are in the directory mentioned previously, you must use it as shown below:

```
$sh sanity_test.sh
The output for this should be:
$Output perfect!
```

Furthermore, you should also create your own test cases and test your program against them. The program takes input from standard input, so you should have to feed in your test cases accordingly. It is highly recommended that you use redirection to test your programs. Talk to your TA if you want to learn how to do it. You could also just look at what `sanity_test.sh` is doing to understand how to use redirection. Move ahead only once you are convinced of the correctness of your work. PLEASE REMEMBER THAT `sanity_test.sh` IS THERE ONLY TO TEST YOUR PROGRAM AGAINST ELEMENTARY TEST CASES. It is up to you to construct all kinds of test cases and validate your work.

Future projects will also use this idea of a `sanity_test`. The point of this is to test your code against some basic inputs that might not be in the earlier examples. To use `sanity_test`, run it in linux using the above instructions, and compare your output with the expected output. The expected output can be found in the 'data' file that is a part of the `Project0.zip` file.

PLEASE NOTE Just because your code passes `sanity_test` does not necessarily mean that your code is perfect code; these are only elementary test cases.

Grading Rubrics

Overview

Tests	Points
Program Compiled and Run	10
Coding Standard	10
Passing All Test Cases	80
Total	100

Details

Program Compiled and Run: 10 pts

We can compile your program and run it successfully, according to our requirements.
We **DO** care about warnings. You will lose points if warnings are raised even though your code compiles and runs.
You should **NOT** change the signatures of the given methods.This can even lead to worse situation: failing most test cases.

Coding Standard: 10pts

Your code should be well structured.
Rule of the thumb: TA can understand any method in less than 10 seconds.

Suggestions:

1. Add Comments. Usually you will have the same amount of comments as code.
2. Use friendly variable names.
3. Have lots of small methods rather than several large methods.
4. **Indentation. You will lose all 10 points if your code is not well indented.**

Passing All Test Cases: 80pts

1. read_input 35 pts
2. create_token 35 pts
3. print_token 10 pts

You are responsible for the robustness of the program. Passing only the provided vanilla test cases may result in very low scores.

Submission

Suggestion: We suggest you create a folder for CS251 projects like CS251_Project0, CS251_Project1 and so on. Store all you project files in that directory. To create such a folder, type "mkdir CS251_Project…"

Note:

1. you should **not** literally copy the following commands.
2. Replace yourLogin with your login ID like "zhan1015".
3. Replace "directoryContainsProject0" with the directory that contains project0 folder like "CS251_Project0". Do **NOT** cd into project0.

To resubmit, you just need to retype the following commands and type "yes" after the third command.

```
ssh yourLogin@data.cs.purdue.edu
cd directoryContainsProject0
turnin -v -c cs251 -p project0 project0
```

After the third command is executed, the system will give you some feedback about which files and folders have been submitted. If you resubmit a project, the previously submitted files will be overwritten.

---

cs25100/fall15/project/project0.txt · Last modified: 2015/08/29 17:00 by zhan1015