|  |  |
| --- | --- |
| **CMPILER Problem Set: Creating a Simple Parser** | Instructor: Neil Patrick Del Gallego |

**[100 pts] General Instructions:** From techniques you’ve picked up in class, implement a **recursive-descent** parser that follows the following specifications.

**Restrictions**

In this hands-on activity, you are not allowed to use any scanner or parsing libraries. This includes ANTLR, Java/C++/C# RegEx utilities, and the like. Only your chosen programming language, native string operations such as concat, format, split, and similar, and a UI library (optional) is allowed. You must write your own scanner from scratch. Using any restricted library will automatically earn a 0 for this exam score.

**Input**

The input will be the following:

* A text file containing **n** arithmetic expressions consisting add and multiply operations only, separated line-by-line.
* The terminal symbols are [0 – 9], [ +, \*, (, ), [, ]]. NOTE: ‘[‘ and ‘]’ are included.
* Sample expressions:
  + 1 + 10
  + 1 \* 10 \* 10 + 4 + (6 + 12345)
* A text file containing your own devised grammar rule for supporting arithmetic expressions. This must follow the Extended Backus-Naur form, the grammar notation used by ANTLR.

**Output**

The parser should create a text file containing **n** outputs:

* ACCEPT, if the input string from the user, is a valid expression.
* REJECT + first offending token if it is an invalid expression.
* The input is also printed for cross-checking.

**Sample Test Case**

A parser should print ACCEPT, or REJECT + first offending token, for every input string in the text file. The following test cases you may use are given in the table below

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 + 10  1 \* 10 \* 10 + 4 + (6 + 12345) | 1 + 10 – ACCEPT  1 \* 10 \* 10 + 4 + (6 + 12345) - ACCEPT |
| [1 + 10] = 11 | 1 + 10 = 11 – REJECT. Offending token ‘=’ |
| 1 + + 10 \* 5 \* 4 \* 3 + 2 \* 1 = = | 1 + + 10 \* 5 \* 4 \* 3 + 2 \* 1 = = - REJECT. Offending token ‘+’ |
| ++++++++++++++++ | ++++++++++++++++ - REJECT. Offending token ‘+’ |
|  | //or offending token is ‘++++++++++++++++’ |
| 5 | 5 - ACCEPT |
| (1 + 4) \* (6 + 4) | (1 + 4) \* (6 + 4) - ACCEPT |
| (1 + x) + (6 + 4) | (1 + x) + (6 + 4) – REJECT. Offending token ‘x’ |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

**Grading Scheme**

The **actual** test case will be different from the sample provided. It will consist of 15 test cases, each worth 4 points each. 4 points are given if the actual result yields the expected result (ACCEPT or REJECT).

Deductions will be given if the following requirements were not met:

|  |
| --- |
| **Syntax Analysis** |
| Grammar rules are not clearly implemented using a data structure. Grammar rules are hardcoded. |
| Does not support error recovery. Program halts if an invalid RE is present in the text file. |
| Grammar rules are ambiguous. A different derivation can be produced. |

60 pts will be for the test cases. 40 pts will be for the PPT submission.

**Submission Details**

*You may prepare some of the requirements in advance.*

The **actual** test case will be given as a timed quiz in Canvas. You are to run the given test case and submit it online. After you are done running the program using the test case provided, submit a GDrive URL containing the following:

* SOURCE – Contains your source code. Add a README.txt that has your name and instructions how to run your program. Also indicate the entry class file, where the main function is located. Alternative can be a Github link.
* PPT – A PowerPoint showing the following set of slides.
  + Title page – Your name + section + a video demo in MP4 format. The video contains a recording of your running program, showing the **actual** test case file as input, and showing the output listing down the results in detail. Should be in MP4 file. Recording must be seamless and not cut. Please ensure that your video is embedded in your PPT.
  + Syntax analysis – Discussion of your syntax analyzer implementation which contains the following. Provide code snippets wherever necessary:
    - Reading the text input. How do you detect terminal symbols?
    - Show your grammar rules. How do you read your grammar file?
    - Show implementation of recursive-descent parser.
    - How do you recover from parsing errors?
* FORM AND OUTPUT –
  + Text file produced by your program which should have a similar format as discussed in the specs.
  + Academic honesty agreement form declaration. See below.

**APPENDIX A: Academic Honesty Form**

==========Copy and paste the following section below and sign the form. Save in PDF file================

**ACADEMIC HONESTY AGREEMENT**

I am answering this exam myself and to the best of my ability, without any assistance from other persons, materials, or resources that I am not allowed to access during the exam period. I declare that the software is fully written by me and without any assistance from my peers. I am fully aware and hereby agree to the clause that violation of this agreement is considered cheating and will result in a 0.0 for this course.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_­­­­­\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Signature over Printed Name**

===========================================END===========================================