# **Project 2 .** Building a Parser

11/13/2015

## 1 Important information

- **Team:** This is an individual project. You have to finish it by yourself although you are encouraged to discuss with other people.
- **Due date:** The design report is due Midnight 10/29/2015. An interview will be conducted on the same day, and you must bring with you a hard copy of your report. The whole project is due Midnight 11/19/2015.
- **Grade:** 10% of the final grade of this course.
- **Deliverables:** One design report (as specified in Section 3) including pseudo code and one program.
- **Grading:** Program 50% and design report 50%.
- **Submission:**
  1. On Midnight 10/29/2015, submit your design/report as PDF file to the `project` folder at Piazza with `summary` line of the form: "Proj 2 Design Yuanlin Zhang" where my first name Yuanlin and last name Zhang will be replaced by your first and last name.
  2. Before Midnight 11/19/2015, post the following to Piazza `project` folder ( Please do not wait until last minute. Even you want to make it perfect until the last minute; you can upload multiple times; only the last one will be graded. If Piazza is not working for you, email grader at cc.liang@ttu.edu. ):
     (a) Summary line of the form: "Proj 2 Final Yuanlin Zhang" where my first name Yuanlin and last name Zhang will be replaced by your first and last name.
     (b) Please prepare to do a demo for grader on the coming Monday(11/23) or Tuesday(11/24). You are expected to compile your submitted program and run it against some test cases. Because this required demo, the coding ground is no longer a consideration.
     (c) The submission is expected to include THREE files: (1) one single compressed program file, containing all your source code, with name "sourcecode"; (2) the report in PDF format whose name is of the form "YZhangProj2FinalReport.pdf" where the initial and last name should be replaced by yours; and (3) a readme in

plain text file format with file name ”readme.txt”. The readme file is about your development evironment, like Microsoft Visual Studio 2013, or gcc 4.9.0.

(d) Your name is supposed to appear in every file.

Your project may not be graded (and thus the grade will be 0) if the requirement above is not followed. Raise any questions about the project and the submission procedure during class.

## 2 The parser

Write a parser for the following context free grammar based on the tokens (e.g., `id`, `lparen` etc.) given in project 1.

```
<program> → <stmt_list> $$
<stmt_list> → <stmt> <stmt_list> | ε
<stmt> → id assign <expr> | read id | write <expr>
<expr> → <term> <term_tail>
<term_tail> → <add_op> <term> <term_tail> | ε
<term> → <factor> <fact_tail>
<fact_tail> → <mult_op> <factor> <fact_tail> | ε
<factor> → lparen <expr> rparen | id | number
<add_op> → plus | minus
<mult_op> → times | div
```

Note that `$$` is a pseudo-token (i.e., not a real token in the input program) which is returned by the scan function when it hits the end of the input file and there is no lexical error.

For the parser,

1. its input is a program (a text file).
2. it prints to the console an XML-like tree structure indicating the parse tree of the input program. If the parser meets an unexpected token, your parser should output to the console the following information only and stops: `Error.` Error recovery and handling multiple errors are beyond the scope of this project.
3. you MUST use the recursive descent approach.
4. `scan` function developed for the scanner in project 1 must be used. **Note** the output of `scan` is one token instead of a sequence of tokens.

One way to generate the XML-like tree is as follows. At the beginning of each function for a non-terminal symbol, print to a string buffer a new line containing `[indent]<[Non-Terminal]>`. Before ending the function, print to the buffer a new line of `[indent]</[Non-Terminal]>`. The `indent` is the spaces one should keep before the `Non-Terminal` symbol. When a token is recognized you will print to the buffer `[indent]<[Token-Type]>[Token]</[Token-Type]>`.

**Commandline:** `parser <input file name>`

Assume the input file has the following program

```
read A
```

**Example Output:**

```
<Program>
  <stmt_list>
    <stmt>
        <read>
            read
        </read>
        <id>
            A
        </id>
    </stmt>
    <stmt_list>
    </stmt_list>
  </stmt_list>
</Program>
```

Note. Here, you output both token type and its content. For example, the `id` token type here has a content `A`.

## 3 Report and Pseudo-code Design

The report consists of four components: 1) Introduction: important ideas and/or data structures you use to build the parser, 2) Pseudo-code: the detailed pseudo-code, 3)Test cases: the test cases with explanation why you select them, and 4) Acknowledgement: acknowledgement of people and their contribution to your project. You may also include any non trivial knowledge you have learned in this project.

Grading considerations/interview questions will be centered around how you print the XML-parse tree correctly and how recursive descent parsing works. One is expected to get a *significant grade deduction if s/he can NOT explain or do NOT understand any part of her/his own report/design*. Therefore, dont write anything into the report and pseudo-code without a reason.

## 4 Implementation

The programming languages used in this project are restricted to **C/C++, Java, or Python**. As for implementation (program), the minimal expectation is a finished implementation which means the source code is compilable to an executable and the executable behaves correctly at least most of the time.

## 5 Acknowledgement

Edward Wertz has contributed to the writing of this project.