COP 3503 Programming Fundamentals for CIS Majors II, Spring 2015
Programming Assignment 3

Out: Mar. 23 (Wednesday), 2016

Due: 9:35am (hard deadline) on Apr. 11 (Monday), 2016

Problem Description

In this programming language, you should implement the **lexical analysis** task for the limited version (i.e., the maximum depth of the nested loop(s)) of a programming language. Lexical analysis is the first stage that compilers parse and detect the possible syntax errors.

Ideally, any new (programming) languages can be designed and analyzed in the similar manner. You will need to analyze a Pascal-and-C-like language in this programming assignment.

Given a segment of the source code, your C++ code should analyze the code and extract all tokens, which include:

- **Keywords:** keywords are the words reserved by the language. They are all upper-case. In our language, we have only three keywords "BEGIN", "END" and "FOR".
- **Identifiers:** An identifier is used to describe the variables, which are all lower-case. Only letters 'a'-'z' are legal in an identifier.
- **Constants:** Numbers like 10 ... No negative numbers.
- **Operators:** all arithmetic operations (i.e., +, -, *, and /), "++" and "="
- **Delimiters:** like "," and ";"

Your C++ code should input a string/file from user, which contains the expression he/she wants the compilers to analyze. Then, your code should parse the input, detect the tokens, classify them, and print out the results.

With this assignment, you will get practice with the stack implementation which is one of the most widely used data structures. Besides, you will be familiar with string processing and input parsing, which are of crucial importance in most C++ projects.

Details

1. (Data Structures:) You need to implement a stack data structure to keep track of the processing and compute maximum depth of the nested loops. Adding elements to the

stack (push) and removing objects from it (pop) are two essential methods that must be implemented. You can use any data structure to implement the stack, e.g., arrays, linked-lists, etc. You need to implement the data structure by yourself, and the only standard libraries you can include are: <iostream>, <fstream>, <string>, <cstring>, <cstdlib>, <cstddef>(don't have to use all of them). We will deduct ten points from your total credit for each extra standard library.

2. (Inputs:) Source code for testing will be given in a file, your program needs to support file input. We will run your program by "./pa3.out filename", see examples for more details.

3. (Algorithms:) Once the input expression is given, your program should decide which character should be inserted to the stack, and when the result needs to be computed. You need to detect the possible syntax errors while tracing the maximum depth of the nested loops.

Example Run

Try to keep your output as close to the given format as possible:

```
(1) Inside test1.txt:

FOR (i, 10, ++)
BEGIN
  FOR (j, 10, ++)
  BEGAN
    sum=sum + i + j;
  END
END


> ./pa3.out test1.txt

OUTPUT> The maximum depth of nested loop(s) is 2

Keywords: FOR BEGIN END
Identifier: sum i j
Constant: 10
Operatros: ++ = +
Delimiter: ; ,
```

```
Syntax Error(s): BEGAN


(2) Inside test2.txt:

IF (i, 10, ++)
BEGIN
END
FOR (j, 10, ++)
BEGIN
  sum=sum + i + j;
END


> ./pa3.out test2.txt

OUTPUT> The maximum depth of nested loop(s) is 1

Keywords: FOR BEGIN END
Identifier: sum i j
Constant: 10
Operatros: ++ = +
Delimiter: ; ,

Syntax Error(s): IF


(3) Inside test3.txt:

FOR (i, 10, ++)
BEGIN
  FOR (j, 10, ++)
  BEGIN
    FOR (k, 10, ++)
    BEGIN
      sum=sum + i * j + k;
    END
  END
END
FOR (i, 10, ++)
BEGIN
```

```
  FOR (j, 10, ++)
  BEGIN
    sum=sum + i / j;
  END
END

> ./pa3.out test3.txt

OUTPUT> The maximum depth of nested loop(s) is 3

Keywords: FOR BEGIN END
Identifier: sum i j k
Constant: 10
Operatros: ++ = + * /
Delimiter: ; ,

Syntax Error(s):
```

Hints

1. You should get your stack data structure working well before implementing the lexical analysis task.

2. The string processing to parse the input is an essential part of this assignment. You should make sure that you parse the input correctly, and you take care of all edge cases, e.g., more than one spaces between characters, no spaces, tab, etc.

Submission Guide-lines

1. You must finish this assignment with your individual effort. Your C++ source code file MUST be named as "pa3.cpp" and C++ header/class file should be named as "pa3.h".

2. Please test your program via g++ compiler (i.e., g++ -Wall) on the CISE machines to make sure it runs correctly.

3. Please upload the source code file(s) to SAKAI system as the attachment(s). Please submit the source code file(s) ONLY. NO need to compress the source code file(s) if the size is small.

4. Make sure you submit your assignment BEFORE the deadline. Late submission will NOT be graded !!

Grading Criteria

1. Successful Compilation (30%): Your source code should be able to compile using "g++ -Wall" command without any error or warning. The output should be a valid executable. Please note that we will be using g++ compiler on Linux to grade your programs. If you are using other compilers or IDE (e.g., Visual C++), it is recommended that you test the source codes with g++ before the SAKAI submission (i.e., make sure there is no warning).

2. Program Correctness (40%): The executable should be able to run correctly by giving out the required output.

3. Programming Style (30%): Good coding style is a key to efficient programming. We encourage you to write clear and readable codes. You should adopt a sensible set of coding conventions, including proper indentation, necessary comments and more. Here are some guidelines of good programming style: http://en.wikipedia.org/wiki/Programming_style

Final Notes

1. AGAIN, remember to start the programming assignments as soon as possible. Unlike the conventional assignments, programming assignments sometimes take un-predictable amount of time to finish. Thus, have the code running first, then polish it later with the extra time before the deadline.

2. Remember you should always write your own code and never copy-and-paste from other students' work or other sources. There are indeed many tools (like Stanford Moss) to detect the code similarity.

3. Programming assignments are usually designed by TAs. If you have any question or concern, please feel free to contact TAs. Our goal is to let you experience the fundamentals of computer science. If you like the programming experience, you are one of us !!