

COP 3503 Programming Fundamentals for CIS Majors II, Spring 2018 **Programming Assignment 3**

Out: Monday Mar. 19, 2018

Due: 8:30am (firm deadline) on Monday Apr. 09, 2018

Problem Description

In this problem, you must implement the **lexical analysis** task for a limited version (i.e., the depth of the nested loops) 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 the case of this simplified language, the keywords are restricted to the set { "BEGIN", "END", "FOR" }
- **Identifiers:** An identifier is used to describe the variables, which are all lower-case
- **Constants:** Numbers like 10,
- **Operators:** all arithmetic operations (i.e., +, -, *, and /), "++" and "="
- **Delimiters:** like ",", and ";

Your C++ codes should input a text 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 the 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.

2. (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 depth of the nested loops.

Example Run

Try to keep your output as close to the given format as possible: In this example, the input file is "code.txt" It contains the code segment below:

Text Case I

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

```
> ./pa3.out
```

```
INPUT> Please enter the name of the input file:  
code.txt
```

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

```
Keywords: FOR BEGIN END
```

```
Identifier: sum i j
```

```
Constant: 10
```

```
Operatros: ++ = +
```

```
Delimiter: ; ,
```

```
Syntax Error(s): BEGAN
```

Test Case II

```
FOR (i, 10, ++))
BEGIN
sum=sum + i + j;
INPUT> Please enter the name of the input file:
code.txt
OUTPUT> The depth of nested loop(s) is 0
```

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

Syntax Error(s): END)

Text Case III

```
FOR (i, 10, ++)  
BEGIN  
FOR (j, 10, ++)  
BEGIN  
sum=sum + i + j;  
END  
FOR (k, 5, ++)  
BEGIN  
mul=mul * k;  
END  
END  
INPUT> Please enter the name of the input file:  
code.txt  
OUTPUT> The depth of nested loop(s) is 2  
Keywords: FOR BEGIN END  
Identifier: sum i j mul k  
Constant: 10 5  
Operators: ++ = + *  
Delimiter: ; ,  
Syntax Error(s): NA
```

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, 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". **Both pa3.h and pa3.cpp are required, and penalty will be applied if you still use the single-file style of C++ source codes.**
2. Please test your program via the g++ compiler (i.e., g++ -std=c++11 -Wall) on the CISE machines to make sure it runs correctly. All source code must be able to run on C++11.
3. You must use the stacks for this assignment.
4. Please upload the source code file(s) to CANVAS 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.
5. Make sure you submit your assignment 30 MINUTES 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++ -std=c++11 -Wall" command without any error or warning. The output should be a valid executable. See lab tutorial for the commands. 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++ on CISE machines **THREE DAYS** before the CANVAS 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

Both header file and cpp file are required to be submitted. **Penalty will be applied if you still use the single-file style of C++ source codes.** If you are still not convinced for the separation of header file, please read the following article: <https://accu.org/index.php/journals/269>

Final Notes

1. AGAIN, remember to start the programming assignments as soon as possible. Unlike the conventional assignments, programming assignments sometimes take unpredictable 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 !!