Final Project

Part 1: Identifying Assignment statements

If you didn't do the bonus for assignment 3, you need to extract all the assignment statements from the list of tokens. This will be used in part 3 of your project.

The general form of an assignment statement is: variable = expression;

The *expression* can be a single variable, a single constant or involve variables and constants combined by the arithmetic operators. Rounded brackets () may also be used in matched pairs in expressions to indicate the order of evaluation. ***Note: an assignment statement must end with a semi-colon.***

For example:

sum = 0.0;

ratio = (a + b)/(c + d);

To implement this functionality, you have to define the **getAssignmentStatements()** function in file **assignment3.cpp**

Each statement should be stored as a linked list of the tokens that comprise it. How you store and index the statements (or head pointers to statements is up to you).

Part 2: Identifying Function Declarations

Function declarations list the input types and return variable (if it exists for a function).

int Fibbonacci(int n);

Note that all function declarations must end with a semi-colon and that the declarations of class member functions counts as function declarations. ***Hints***: **1)** The quickest way to find a function declaration in your program is to parse your token list for a '('. If the '(' is followed by a keyword for a type or "const" then you be looking at a function declaration. **2)** Additionally, you might want to pre-parse your program for user-defined types ("typedef", "struct", "class") and keep a list of these user-defined types (or add a field to the token class to identify them if you prefer).

Part 3: Error Detection & Analysis

For each of the assignment statements you detected, you will analyze for the following errors:

1.) Unmatched types: Note, even if there is a conversion function for a type (e.g. int to float), you should still detect the mismatched type in the statement. (Hint: To do this, you are going to

want to identify the type of each identifier and store it. Possible ways of doing this include using a table or adding an additional field to the token.)

2.) Unmatched braces (i.e. '(' and ')')

You should also be able to return statistics on any program you analyze in both a verbose and non-verbose mode.

1. In non-verbose mode, you should return:
    a. the number of assignment statements (of the type we asked you to identify in Part1);
    b. the number of function declarations, a
    c. a summary of the total number of tokens in the program that is parsed, and
    d. a breakdown of the total number of each type of token detected.
2. In the verbose mode you should return:
    a. the number of assignment statements (of the type we asked you to identify in Part1);
    b. the number of function declarations, and
    c. a summary of the total number of tokens in the program,
    d. for each token type, give the total followed by a *list of all the tokens from the program of that type*.

BONUS:

You may add bonus features to your program. Possible bonus features you might want to consider:

- Detecting additional types of errors in assignment statements (e.g. missing operators or operands).
- Detecting errors in function declarations (e.g. missing input type definitions).

INSTRUCTIONS FOR SUBMITTING YOUR CODE

- Copy the directory structure you submitted for assignment 3 into a new directory
- Rename "assignment3.cpp" as "project.cpp"
- Update your Makefile accordingly.

- **Maintain this new directory structure based off Assignment 3. Do not modify any additional names or locations of any of the files you used in Assignment 3.**
- For the project you will be submitting the c++ files (i.e. "*.cpp" and "*.h"), the Makefile for your project, and a README file (if necessary).
- To submit your assignment, update the Makefile with your student number and run:

make tar

- Submit the resulting tar.gz file to Canvas (using the Final Project submission link).