CS 315
Project 1

Assigned: Feb. 14, 2025
Due: Feb. 23, 2025, 23:59

Lexical Analyser for a Programming Language for Real Numbers

This semester's projects are about the design of a new language that is limited to the real numbers. This newly designed language will be similar to the imperative languages. The main difference is that variables in this language can only take values from the set of real numbers, $\mathbb{R}$. String constants can be used in the input (for promting) and output operations. The first part of the project is about the design of the language and the implementation of its lexical analyzer.

Part A - Language Design (30 points)

First, you will give a name to your language and design its syntax. Note that the best way to hand in your design is with its grammar in the BNF (not EBNF) form, followed by a description of each of your language components. The following is a list of minimal features that are required in your language:

- Constant real values
- Variables, which may be assigned real values
- Operators (the four arithmetic operators, modula and exponentiation)
- Expressions may contain matching parentheses along with real values, variables, operators, and function calls.
- Conditional statements (e.g., if-then, if-then-else)
- looping statement(s)
- Input/output statements (string constants can be displayed on the console)
- A data structure to store a collection of real values
- Function definitions (functions take any number of, and return a single, real values)
- Comments.

You are encouraged to use your imagination to extend the list given above.

Discuss any other features of your language that you think is important.

Do not panic! You will have a chance to do minor revisions to your syntax design for Project 2 (later in the semester). Language designs are almost never exactly right in the first iteration. Just try your best to make it as readable/writable/reliable as you can and keep your eyes open for what does and what does not work :)

Part B - Lexical Analyzer (40 points)

Having designed the language, you will design and implement a lexical analyzer for your language, using the lex tool available on all Unix-style systems. Your scanner should read its input stream and output a sequence of tokens corresponding to the lexemes you will define in your language. Since at this stage, you will not be able to connect the output to a parser, your scanner will print the names of the tokens on the screen. For instance, if we were designing a C-like syntax, for the input

```
if (a - b) x <- 7.5 ; display "OK" x ;
```

the lexical analyzer should produce the output similar to the following:

```
IF LP IDENTIFIER OP IDENTIFIER RP IDENTIFIER ASSIGN CONST SC DISPLAY STR IDENTIFIER SC
```

For conditional statements, such as if, you may follow the convention in the C language, that is, the value 0 is considered to be false and any non-zero value is considered to be true.

Part C - Example Programs (30 points)

Finally, you will prepare 3 test programs of your choice that exercise all of the language constructs in your language, including the ones that you may have defined in addition to the required list given above. Be creative, and have some fun. Additional to your test programs, write the programs for the following two programs given as pseudo-code:

Program 1:

```
1. prompt the user to enter the values of x, y z
2. read the value of x, y and z from the keyboard
3. if any of them is zero, ask the user to enter only non-zero values
4. keep reading new values until proper values are entered.
5. display the value of the expression (x times y times z)
```

Program 2:

```
1. define a function foo that has two parameters p and q,
   and displays its name, the names and the values of the parameters, then
   it returns the largest of the parameters.
2. for each value of a in the list of {3.15, 7, 0.03, -1.7} // assuming your data structure has such a syntax
2.1.   for each value of b in the list of {9, -1.2, +1.2}
2.1.1.    c gets foo(a, b)
2.1.2     display the values of a, b and c
```

Make sure your lex implementation correctly identifies all the tokens. The TA will test your lexical analyzer with these example programs along with other programs written in your language.

Do not panic! You are not required to write an interpreter or compiler for this language. Just write a few programs in the language you designed and make sure that the lexical analyzer produces the right sequence of tokens.

Teams

The project will be implemented in teams of two or three students. The members of the teams will be the same for this and the next project, in which you will write a parser for your programming language. Note that all members of a team will get the same grade in this part. We will assume each member has contributed to the project equally.

Submission

- There are several parts that you will hand in.
    1. A project report including the following components:
        - Name, ID, and section for all of the project group members.
        - The name of your programming language.
        - The complete BNF description of your language.
        - About one paragraph explanation for each language construct (i.e. variables and terminals) detailing their intended usage and meaning, as well as **all of the associated conventions.**
        - Descriptions of how nontrivial tokens (comments, identifiers, literals, reserved words, etc) are defined in your language. For all of these, explain what your motivations and constraints were and how they relate to various language criteria such as readability, writability, reliability, etc.
    2. Your lex description file
    3. The example program described above, written in your language
- Make sure your lexical analyzer compiles and runs on dijkstra.cs.bilkent.edu.tr . The TA will test your project on the dijkstra machine, and any project that does not compile or run on this machine will get 0 on Part-B.
- Your password is the same as the one for your @ug.bilkent.edu.tr e-mail account.
- Please upload all of the above items, as a single zip file, to *Moodle* before the due date. PDF format is preferred for the project reports.
- Late submissions will be accepted, with 20 points (out of 100) deduction for each extra day.

Resources

- Running lex and yacc on Linux systems (accessible in Bilkent Campus)
- The Lex & Yacc Page (dinosaur.compilertools.net)

Good Luck! - Have fun.
