Project #1 — Lexical Analyzer

Jared Dyreson, Chris Nutter

California State University, Fullerton

Contents

1 Introduction

This project was started to create a lexical analyzer for our compiler and was aptly named “Lexi”. The goal of Lexi is to parse out the contents of a source document and generate meaningful lexemes. These lexemes were to adhere to a specific set of regular expressions to define a token. Here are the following expressions used:

- Comments:

  ```
      (^\!.*\!)
  ```

- Yo dady

2 How to use

¡++¿

3 Design

3.1 Regular Expression

The way *Lexi* parses each line and determines the identifier type is through the use of *regular expressions*. Being able to determine the identifier is crucial in defining the token's contents. *Lexi* after processing the file and creating a vector of strings that parses line by line which is then fed through a function that reads each character and determines one of the each lexeme types.

1. ***Comments*** determines any line that has *!* and ends with a trailing *!*. Multiple comments in a line are supported.

   ```
   (!.*!)
   ```

2. ***Keywords*** finds any word that is considered reserved for the structure of the language including data types, control-flow operators, and other key-defining words for the language.

   ```
   (int|float|bool|true|false|(end)?if|else|then|while(end)?
   |do(end)?|for(end)?|(in|out)put|and|or|not)
   ```

3. ***Number*** is any integer, float, double, size_t, (etc.) value for identifying amount.

   ```
   (?:b)([-+]?d*.?\\d+)?(?=b)
   ```

4. ***Identifier*** grabs any word that is not within a *comment* or *keyword* field.

   ```
   ([a-zA-Z]+(d*)?)
   ```

5. ***Separators*** finds any symbol that helps keep the contents contained.

   ```
   (+|-|*|/|=|>|<|>=|<=|&+||+|%|^!$|^)
   ```

6. ***Operators*** obtains symbols that the language uses for operation.

   ```
   ((|)|{|}|[|]|"|'|,)
   ```

7. ***Terminators*** are symbols signalling end of a line.

   ```
   (;|$)
   ```

4 Limitations

5 Shortcomings