CPSC 4660 PL Scanner

Steven Deutekom
Ricky Bueckert
University of Lethbridge

February 1, 2020

1 Purpose of Software

The software we are building is a PL compiler that will compile files in the PL language so that they can be run on a stack machine. The current phase is the implementation of the scanner. This program will take text files and scan them for tokens in the PL language. The tokens will be output to a file to verify that the correct tokens are being produced. In the future the scanner will be used by the parser to get tokens from PL programs for the parser to use when it verifies the code and creates the necessary object code for our stack machine. The current implementation allows for scanning test files to see what kind of tokens are produced.

2 Design

This section decribes each class, its function, and how it connects to other classes. More detailed documentation of the specific functions and members of each class is available in the **ReferenceDoc.pdf**.

2.1 Scanner

The Scanner class is the main class for this part of the project. It is responsible for converting a string of characters into Tokens. It collects groups of characters in the text based on the first character of the group and returns a token using the longest valid lexeme that can be made from the group. The groups are Identifiers and keywords, Integers, And various symbols, as defined by the PL language.

Once a group of characters is classified a token is created and it is returned to the process that called `Scanner::getToken()`. If the lexeme is a name it is added to the Symbol table and the token is updated to discover if the lexeme is a keyword or an Identifier.

If the scanner discovers a character that is not in the alphabet an bad character error token is returned. If the lexeme is a number and it is larger than a valid integer then a bad number error token is returned. If the symbol table is full then the scanner returns a full table error. All of these errors do not stop the scanning process or change the way subsequent lexemes are scanned.

2.2 Token

The token class holds information about a lexeme. In our implementation we have given it 3 fields. First a Symbol to identify what kind of token it is. Second there is the lexeme that the token was created from. Finally we have an integer value that stores the value of an integer token or the position of an identifier or keyword in the symbol table. The token class is only responsible for holding data and has public methods to access and mutate its data members. For debugging purposes it also has a method to return a string representation of itself that can be printed.

2.3 Symbol

Symbol is an enum that collects all the possible token types. These are used to classify and identify different tokens. In the same file there is also a table that maps token types to a string identifer so symbols can be printed in a readable way.

2.4 Administration

The Administraton class is responsible for all of the things that are not part of the previous classes. At this moment it is mostly responsible for using the scanner and calling `getToken()`. It outputs each token it receives as a string in a file for debugging. It is also responsible for keeping track of errors and printing error messages. Currently it will print only one error message per line of input. When it encounters a total of 10 errors it will terminate the scanning phase. It also keeps track of the number of the line that is being scanned.