Seung Joon Hwang
2020-15313

Compiler Project Phase 1

Lexical Analysis

Introduction:

In this part, we scan every character of the input and convert them into sequence of tokens. Tokens are needed for syntax analysis which is Phase 2 of our project.

Implementation:

First, we should define appropriate token symbols fully enough to describe out language. Given some tokens from Snupl-1, we only need to add few more tokens. All token types are defined in scanner.h header file.

After initializing appropriate values for token name and keywords, we edit function
CToken CScanner::Scan()*
which is used to scan through the input stream and return appropriate token. It first consumes all the whitespaces and comments. This is done at the beginning of the method. Then, we start classifying tokens.

There are 4 points worth mentioning in classifying tokens. Rest are simple

1. Keywords
2. Number
3. Character
4. String

Keywords:
These are handled by *map<string, EToken> CScanner::keywords* which maps string and its corresponding Token type. While handling identifier, if the token value (*tokval*) matches some key in keywords, it means that it is a keyword of snupl-2 and returns corresponding token.

Number:
Since no identifier can start with a digit, if a lexeme starts with a digit, it is a *Number*. We then simply just go through while loop until the next character is not a digit. One corner case is when the last character is 'L'. Then, it is a longint variable so we also assign token type *Number* for it.

Character:
When the character scanned is a quote, we use *CScanner::GetCharacter()* method which is provided in skeleton code in order to parse the character constant or characters in a string. In this code I handled error by returning token *tInvCharConst* when there is one or more character inside quotes, or the line ended without closing the quote. Value returned is just a first character appeared after the first quote. And

String:
Similar to Character, we use *CScanner::GetCharacter()* method in String mode when the character scanned is a double quote. Until the string is valid and another double quote appears, we loop through the characters inside the string and add it to token value(*tokval*). *tInvStringConst* is returned when it contains invalid character or line ends without closing the double quote. Value returned is the string appeared after the first double quote (with double quotes when it is a valid String)