Lexical Analysis

Documentation for the lexical analysis section of my project

Transition Table

My transition table is in `files/transition_table.txt`. It consists of 128 columns for each ascii character and 25 rows for each state I have. Out of the 25 states, 14 of them are accept states and 2 are panic states.

File Handling

I have 3 input files in my project and 2 output files. All of these files are stored in the `files` folder in my project.

2 of the input files are `keywords.txt`, to store information on keywords and their relative token, and `transition_table.txt` which stores the transition table.

The third input file is the file to read the code. It's currently written in `lexer.py` on line 13 and the filename can be replaced with any file that has code to be analyzed.

```
# enter code file here!
code_data = open('test cases/Test9.cp', 'r')
```

The 2 output files are `lexemes.txt` to store my <token, lexeme> pairs and `errors.txt` to store any lexical errors that occur within the project. The lexical errors in my project are characters which are not included in the grammar of the language (for example: ~).

Double Buffers

A double buffer has been implemented in this lexical analyzer, the two buffers are labeled `buffA` and `buffB`. They are arrays that store up to 2048 characters read from the code. There is also `currBuff`, to track which buffer is currently being used, and `BUFFER_SIZE` to save the length of the buffer as a constant.

There are 2 main functions for the buffer. `fillBuffer()` reads from the code file up to 2048 characters to fill a specific buffer. `readBuffer()` contains loops that will go through each character in the buffer and handle each according to the transition table.

There is also handling for 2 specific buffer cases implemented.

- If one buffer overflows into the next buffer `contBuff` is used to flag whenever the previous buffer will continue into the next. When that is `True` then there is handling on all accept and panic states so that the lexeme that's used includes text from the previous buffer.

  ```
  if contBuff:
      if currBuff == 0:
          fullLex = buffB[cont_point:] + buffA[:curr_point]
      else:
          fullLex = buffA[cont_point:] + buffB[:curr_point]
  final_lexer.append('<int, {}>\n'.format(''.join(chr(c) for c in fullLex)))
  contBuff = False
  cont_point = 0
  ```

- If one lexeme is greater than the length of both buffers combined In this case, an error will be recorded to say that the length is exceeding the character limit and the buffers will move on to the following characters.

  ```
  if contBuff:
      final_errors.append("error line {}: value exceeds character limit\n".format(line))
  ```