Code Report: A Lexical Scanner for VC

Nguyen Tuan Anh - 20021292 Tran Trong Thinh - 22028073
Hoang Duc Minh - 22028051

April 17, 2024

1 Introduction

This is a report for the project *A Lexical Scanner for VC*

2 Design

2.1 Transition table

We define the state transition graph. The graph is based on the VC Language Definition. After that, we wrote the transition table in the form of a excel file. Then reformat it in the .dat file. The .dat file includes the start state, transition table, and end states.

The format of the file is as follows:

```
starting_state 0

TRANSITIONS
s0 input s1
...

ENDSTATES
s0 flag
...
```

2.2 Functions

The utility functions are written in the main.py file. It provides methods for reading the source code, getting state, parsing the transition table, and writing the output files.

`readAutomation()`: Read the .dat file that stores the transition table and end states.

`find_next_state(graph, token, state, char)`: Determines next state from the current state and input.

`check_end_state(end, state)`: Check if the current state is the finished state.

`output(end, vc_tok, vc_tok_verbose, state, current_word, count_line, count_col)`: Writes the tokens to an output file.

`scan(graph, end, token, input_file)`: Tokenize the source code using the transition table, state and various utility methods.

`generate_token(file)`: Use the above functions to create output file

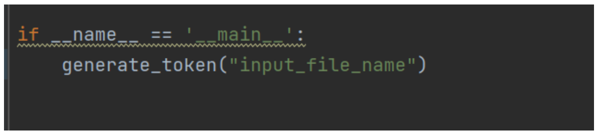
2.3 Detailed workflow

Loop through each line with each line looping over each character:

- If in multiply comment skip to next line.
- If the state starts reading until the non-alphabetic character or the last character of the line and stores those characters in current_word
- If current_word is not empty, check whether the next state is a keyword or string literal
- Check 2 consecutive characters:
  - If you see a comment, skip to the next line
  - If previous state is end state and current state is None reset start state and current_word, if previous state is different Space write output
  - If the current state with 2-character input is the output state and reset to the original state, skip this character and the next character.
- Check input 1 character
  - If the current character is not a space
    - If previous state is end state and current state is None reset start state and current_word, if previous state is different Space write put
    - If is the end-of-line character and is the end state of writing output
    - If not the end of line character
      - Check the next status, if not None, then move to the next character with the corresponding status
      - If current state is end state and not space write output and reset state
  - If the current character is a space, reset state and current_word

At the end of the file, an end token ($) is added to the output file.

3 How to run



```
if __name__ == '__main__':
    generate_token("input_file_name")
```

- Starting file is main.py
- Put `input_file_name.vc` file in the same directory as main.py and put file name in `generate_token` function.
- Run the main.py file.

4 Conclusion

The lexer is able to tokenize the source code correctly, but lacks compilation error detection which needs to be improved in the future.