# COMS540

## Benjamin Yen Kit Lee

# 1 Project Details

## 1.1 Part 0

There is a source file named ”mycc.py”. All codes are written in this file only for this part of the assignment. In the terminal, for this part of the assignment, there will only be one mode switch, which is ”-0” where it display the compiler name, author and contact information, a version number and a date of ”release”. There will be no input file for this mode. If no argument are given it will display the usage information.

## 1.2 Part 1

There is a source file named ”mycc.py”. All codes are written in this file only for part 1 of the assignment. In the terminal, for this part of the assignment, there will be two mode switch, which is ”-0” and ”-1”. For ”-1” mode, the purpose of the file is to read an input file, and divide it into tokens, which is similar to a lexer. The output stream should contain a line for each token, showing the file name, line number and text corresponding to the token if it is a valid input file. The output file will be in the form of ”.lexer”. When an error is read, it will generate an error file in the form of ”.error1”. The file is in python, such that there is a function part_1() which take parameter of a file name (file_name), a 2D list that splits every element in a line to individual character (line_character), and an output list that stores the output stream that contain a line for each token (output). The lexeme and its value are stored in a dictionary data structure.

Since the code is written in python, it reads every character in a double for loop, and iterates through every character in the 2D list. For C style comments, when it sees two consecutive character in the form of ”/*”, there will be a boolean variable which will be set to True from False, meaning anything it reads will be discarded. When it reads a closing comment in the form of ”*/”, the boolean variable will set back to False. Every character will be evaluated when the boolean variable is set to False and no character will be evaluated when the boolean variable is set to True. It will output an error when it reads an opening comment but doesnt read a closing comment when it reaches the end of file. For C++ style comments, when it sees two consecutive character in the form of ”//” it will discard everything it reads and break to the next line or the end of file.

For each character in the 2D list, it will be checked if it is an alphabet. If it is an alphabet, there will be a boolean variable that will track it until it read a non-alphabet, that way it will be evaluated later if the word is in the Keywords or Attributes. If the word is not in Keywords or Attributes, it will be considered an identifier.

For each character in the 2D list, it will be checked if it is a digit. If it is a digit, there will be a boolean variable that will track it until it read a non-digit that is not a real literal form too. After reading the digit, it will check if the digit is an integer literal or real literal. If it has any ”.”, ”e”, ”E”, ”e” or ”E” followed by a plus or minus sign, it will be evaluated as a real literal, integer literal otherwise.

For character literals, it will be checked if it is a in the form of ’ ’, which will consist of one character in the space. It will also accept two character in the space such as ’\a’, ’\b’, ’\n’, ’\r’, ’\t’, ’\\’, and

'\' '. There will be a boolean variable that will be set to true when it reads a ' and will be set to False when it reads a ' again. If the form of a character is wrong, it will output an error.

For checking string, it will have a boolean variable that will be set to true when it reads a " and will be set to False when it reads a " again. It will accept escape sequence \\ and \" as string too. If it reads an opening " and doesn't see read a closing ", it will output an error.

For checking single character symbol, if it reads a symbol, it will check if it is a valid symbol in the single character dictionary. If is it not a valid symbol, it will output an error. If it is a valid symbol, it will check if it is an operator with two character by looking at the next character. If it is not, it will output the character, if the next character matches the operators with two character, it will output both character as an operator with two character.

For checking the size limits of the lexemes, it will provide an error if it reads an integer literals with more than 48 character, a real literals with more than 48 character, an identifier with more than 48 character and string literal with more than 1024 character.

For #include directives, when it read #include, it will read the indicated file in double quotes and run the token input stream on the indicated file. When the file is exhausted, it will switch back to the original file.

There is three function in the class p1, a read_file(file_name) function and part_1(file_name, line_character, output, error, include_counter) function and run_p1(file_name). The read_file function should open and read a file, splitting everything into a character, similar to C using the getc method and return a 2D list. However, there is no ungetc method in python, therefore this will be an iterative method. Besides that, part_1 function should iterate through the entire 2D list and check every character. If an error exists, it will immediately print to standard error in the terminal. If there are no error, it will output a .lexer file and the file will contain every token with their information line by line. The run_p1 function will execute the code and output the final lexer file if everything works correctly or print to standard error in the terminal.