Lexical Analysis

> Lab1 of Compilers, FDU, 2016, Modern Compiler Implementation in C.

comment handling

- As we know from the book, comment in tiger-language is wrapped in ***/.../*** formula, so we need to use regex ***/**** to detect if we encounter the start of a comment string.
- We define a ***IN_COMMENT*** state, when we find comment happens using the ***/**** regex, we start the ***IN_COMMENT*** state.
- In my design, if we encounter more ***/**** in the ***IN_COMMENT*** state, most programming languages don't allow nested comment. As there are no requirement of that in the textbook and testcases, I implement both cases.
- If we allow nested comment, we use a variable ***commentStateNum***, inside the ***IN_COMMENT*** state, if we meet another ***/****, we increase the variable by 1, if we meet one ****/***, we decrease the variable by 1 on contrast. When the variable is equal to zero, then there are no nested comment, and we just quit to the ***INITIAL*** state.
- If nested comment is not allowed, we just ignore any ***/**** in the comment state, and quit to the ***INITIAL*** state when we meet the first ****/***
- If we meet ****/*** inside ***INITIAL_STATE***, we regard it as illegal since we are not in the comment state.
- I have also handled some other common status inside ***IN_COMMENT*** state, like the ***EOF*** and ***/n*** state, in order to pass the testcases. Other inputs are all considered legal.

string handling

- The way we use to store the string content is by using a ***buffer*** variable, which is a char array(buffer). The default size of the ***buffer*** is set to be 16, using const int to make it immutable.
- When the buffer length comes up to the limit, we just double the buffer size and copy the old content to the new buffer to continue storing.
- Specially, we also maintain a ***len*** variable to store the starting char position. Every time we call ***adjust()***, the ***EM_tokPos*** increases. If we calculate the starting position using string length, sometimes it will come to incorrect results.
- Regex ***/"*** is used to identify the start of string literals. There are several special situations inside ***IN_STRING*** state:
- ***EOF***: Cause error as it illegally terminate the state.
- Escape character like ***\n, \t, ...***: push it's real character, for example if we encounter '\n' symbol, we just append '\n' symbol to the buffer.
- ***"***: It means the end of string. Return "(null)" to satisfy the test cases when we meet an empty string, it's a special case.
- The other symbols we read are all consider legal in my design.

error handling

- In my design, most of the regular errors are all handled, including:
- ***EOF*** sign in ***IN_COMMENT*** and ***IN_STRING*** states.
- Try to dangerously close a comment when we are not in ***IN_COMMENT*** state.
- Illegal tokens.

file end(EOF) handling

- ***EOF*** is handled in ***IN_STRING*** and ***IN_COMMENT*** states. When we encounter it in such states, we consume it as error as it forces to terminate special state while it's running, so we throw an error and terminate.
- The program will definitely end at an ***EOF*** sign, so we don't need to handle the EOF sign in the regular occasions.

extra tests

- Test some other extra escape characters, for example: "\a", "\v", "\t", \b", "\f", "\r" and so on. They are not used in testcases but also perform right.