```
#include "Lexer.h"


Lexer::Lexer() : comment(false) {}

Lexer::~Lexer() {}

std::vector<Lexer::Token> Lexer::lex(std::stringstream &buffer, int lineNumber)
{
    std::vector<Token> tokens;
    Token *token;
    char c;
    int transition;
    std::string lexeme = "";
    std::string tokenStr = "";
    int prevState = 0;
    int currState = 0;

    while (buffer.get(c))
    {
        // Check if we are inside of a multiline comment,
        // or at the beginning of a new comment range.
        if (comment | (c == '[' && buffer.peek() == '*'))
        {
            // Iterate until we see a '*]'
            while (c != '*' | buffer.peek() != ']')
            {
                // If we hit the end of the line, set
                // comment switch to "true" and reset
                // the current character so it gets ignored.
                if (buffer.eof())
                {
                    comment = true;
                    c = ' ';
                    break;
                }

                buffer.get(c);
            }

            // If we haven't reached the end of the file,
            // and the current character is a '*', we know
            // we have reached the end of the comment section.
            if (!buffer.eof() && c == '*')
            {
                comment = false;

                // Get both characters '*]' out of the stream
                buffer.get(c).get(c);
            }
        }

        // Get the character type (transition)
        transition = getTransition(c);

        // Update state
        currState = Lexer::stateTable[currState][transition];

        // Terminating state
        if (currState == TRM)
        {
            tokenStr = stateToString(prevState);

            if (tokenStr != "Illegal")
            {

                if (tokenStr == "Identifier")
                {
                    // Check if this identifier is a keyword
                    if (isKeyword(lexeme))
                    {
                        tokenStr = "Keyword";
                    }
                }

                // Create token and add to list of tokens
                token = new Token(tokenStr, lexeme, lineNumber);
                tokens.push_back(*token);

                // reset state machine
                currState = NS;
                lexeme.clear();
                tokenStr.clear();

                // If we reached the terminating state by anything other
                // than whitespace, we need to put it back and re-examine
                // the character on the next iteration.
                if (!isspace(c))
                {
                    buffer.putback(c);
                }
            }
            else
            {
                // Push back rejected token
                if (!lexeme.empty())
```

```
                {
                    token = new Token(tokenStr, lexeme, lineNumber);
                    tokens.push_back(*token);
                }

                // reset state machine
                currState = NS;
                lexeme.clear();
                tokenStr.clear();
            }
        }
        else
        {
            if (!isspace(c))
            {
                lexeme.push_back(c);
            }
        }

        prevState = currState;
    }

    // Grab the last token
    tokenStr = stateToString(prevState);

    // Evaluate the last token
    if (tokenStr != "Illegal")
    {
        if (tokenStr == "Identifier")
        {
            // Check if this identifier is a keyword
            if (isKeyword(lexeme))
            {
                tokenStr = "Keyword";
            }
        }

        // Create token and add to list of tokens
        token = new Token(tokenStr, lexeme, lineNumber);
        tokens.push_back(*token);
    }

    return tokens;
}

int Lexer::getTransition(char c) const
{
    int transition = REJECT;

    if (isdigit(c))
    {
        transition = INTEGER;
    }
    else if (isalpha(c))
    {
        transition = IDENTIFIER;
    }
    else if (c == '.')
    {
        transition = REAL;
    }
    else if (c == '^')
    {
        transition = CARROT;
    }
    else if (c == '=')
    {
        transition = EQUALS;
    }
    else if (c == '>')
    {
        transition = GREATERTHAN;
    }
    else if (c == '<')
    {
        transition = LESSTHAN;
    }
    else if (c == '+')
    {
        transition = PLUS;
    }
    else if (c == '-')
    {
        transition = MINUS;
    }
    else if (c == '*')
    {
        transition = MULTIPLY;
    }
    else if (c == '/')
    {
        transition = DIVIDE;
    }
    else if (isValidSeparator(c))
    {
        transition = SEPARATOR;
```

```
193:     }
194:     else if (c == '$')
195:     {
196:         transition = FUNC_SEPARATOR;
197:     }
198:
199:     return transition;
200: }
201:
202: std::string Lexer::stateToString(int state) const
203: {
204:     std::string stateStr = "Illegal";
205:
206:     switch (state)
207:     {
208:     case S01:
209:     case S02:
210:         stateStr = "Identifier";
211:         break;
212:     case S04:
213:         stateStr = "Integer";
214:         break;
215:     case S06:
216:         stateStr = "Real";
217:         break;
218:     case S07:
219:     case S09:
220:         stateStr = "Separator";
221:         break;
222:     case S11:
223:     case S12:
224:     case S13:
225:     case S14:
226:         stateStr = "Operator";
227:         break;
228:     }
229:
230:     return stateStr;
231: }
232:
233: bool Lexer::isValidSeparator(char c) const
234: {
235:     return separators.count(c);
236: }
237:
238: bool Lexer::isKeyword(std::string token) const
239: {
240:     return keywords.count(token);
241: }
```