```
    1: #ifndef LEXER_H
    2: #define LEXER_H
    3:
    4: #include <sstream>
    5: #include <vector>
    6: #include <iostream>
    7: #include <unordered_set>
    8: #include <iomanip>
    9:
   10: class Lexer
   11: {
   12: public:
   13:     enum State
   14:     {
   15:         NS = 0, // NULL STATE
   16:         S01,    // ACCEPTABLE ID
   17:         S02,    // ACCEPTABLE ID
   18:         S03,
   19:         S04, // ACCEPTABLE INT
   20:         S05,
   21:         S06, // ACCEPTABLE REAL
   22:         S07,
   23:         S08,
   24:         S09,
   25:         S10,
   26:         S11, // ACCEPTABLE '$$'
   27:         S12,
   28:         S13,
   29:         S14,
   30:         TRM // TERMINATING
   31:     };
   32:
   33:     enum TransitionType
   34:     {
   35:         IDENTIFIER = 0,
   36:         INTEGER,
   37:         REAL,
   38:         CARROT,
   39:         EQUALS,
   40:         GREATERTHAN,
   41:         LESSTHAN,
   42:         PLUS,
   43:         MINUS,
   44:         MULTIPLY,
   45:         DIVIDE,
   46:         SEPARATOR,
   47:         FUNC_SEPARATOR,
   48:         REJECT
   49:     };
   50:
   51:     // State table
   52:     int stateTable[18][14] = { {S01, S04, TRM, S10, S12, S14, S14, S14, S14, S14, S14, S07, S08, TRM}, // INITIAL STATE
   53:                                {S02, S03, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM}, // ACCEPTABLE ID
   54:                                {S02, S03, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM}, // ACCEPTABLE ID
   55:                                {S02, S03, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM},
   56:                                {TRM, S04, S05, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM}, // ACCEPTABLE INT
   57:                                {TRM, S06, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM},
   58:                                {TRM, S06, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM}, // ACCEPTABLE REAL
   59:                                {TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM}, // ACCEPTABLE SEPA
RATOR
   60:                                {TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, S09, TRM},
   61:                                {TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM},  // ACCEPTABLE '$$
'
   62:                                {TRM, TRM, TRM, TRM, S11, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM},
   63:                                {TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM}, // ACCEPTABLE "^="
   64:                                {TRM, TRM, TRM, TRM, S13, S13, S13, TRM, TRM, TRM, TRM, TRM, TRM, TRM}, // ACCEPTABLE "="
   65:                                {TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM}, // ACCEPTABLE DOUB
LE OP
   66:                                {TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM}, // ACCEPTABLE SING
LE OF
   67:                                {TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM, TRM} }; // TERMINATING
   68:
   69:     std::unordered_set<std::string> keywords = { "while", "whileend", "int", "function", "if", "ifend", "return", "get",
 "put", "true", "false", "boolean", "real", "else" };
   70:     std::unordered_set<char> separators = { '(', ')', '{', '}', ',', ':', ';' };
   71:
   72:     struct Token
   73:     {
   74:         Token() : token("nil"), lexeme("nil"), lineNumber(-1) {};
   75:         Token(std::string token, std::string lexeme, int lineNumber)
   76:         {
   77:             this->token = token;
   78:             this->lexeme = lexeme;
   79:             this->lineNumber = lineNumber;
   80:         }
   81:
   82:         std::string token;
   83:         std::string lexeme;
   84:         int lineNumber;
   85:     };
   86:
   87:     // Constructor
   88:     Lexer();
   89:
   90:     // Destructor
   91:     ~Lexer();
```

```
 92:
 93:     std::vector<Token> lex(std::stringstream &buffer, int lineNumber);
 94:
 95: private:
 96:     bool comment;
 97:
 98:     int getTransition(char tokenChar) const;
 99:
100:     std::string stateToString(int state) const;
101:
102:     bool isValidSeparator(char c) const;
103:
104:     bool isKeyword(std::string token) const;
105: };
106:
107: #endif // LEXER_H
```