RELATÓRIO – LEXER, SINTÁTICO

Aluno 1: Caio de Souza Célio

Aluno 2: Gabriel Henrique Dias Werneck

CLASSES, MÉTODOS E ATRIBUTOS

1. <*classe*> **Compiler:** Classe principal, responsável por iniciar o Lexer, bem como as futuras partes do compilador.

2. <*classe*> **TabelaSim:** Classe que manipula a Tabela de Símbolos, com o uso de um HashMap abrange o Token e o Identifier das palavras lidas.

    - <*atributo*>HashMap symbolTable: Um HashMap responsável por armazenar os elementos da tabela de símbolos;

    2.1 <*métodо*> Token returnToken(String lexeme): Pesquisa na tabela de símbolos se há algum token com determinado lexema;
    2.2 <*métodо*> Identifier getIdentifier(Token w): Retorna a identificação do token;
    2.3 <*métodо*> String show(): Utilizado para imprimir as informações da tabela de símbolos;
    2.4 <*métodо*> put(Token w, Identifier i): Utilizado para adicionar elementos no HashMap.

3. <*classe*> **Token**: Classe responsável por abranger a impressão de um token e o controle de linhas e colunas.

Inclui os getters e setters das seguintes variáveis:

- <*atributo*>String lexeme: Recebe o lexema, ou seja, a palavra lida no arquivo que está sendo compilado;
- <*atributo*> Type classType: Recebe o tipo do lexema, ou seja, identifica se ela é uma Keyword, um char e etc;
- <*atributo*> int line: Recebe o valor da linha;
- <*atributo*>int column: Recebe o valor da coluna.

3.1 <*métodо*> Token(Type classType, String lexeme, int line, int column): Imprime o tipo da palavra lida, a palavra lida, e o seu posicionamento no aspecto de linha e coluna;
3.2 <*métodо*> toString(): Formata a saída de parte do Token;
3.3 <*métodо*> getLexeme(): Utilizado para ler o lexeme;
3.4 <*métodо*> getClassType(): Utilizado para ler o tipo de token;

3.5 <*método*> getLine(): Utilizado para ler a quantidade de linhas;
3.6 <*método*> getColumn(): Utilizado para ler a quantidade de colunas;
3.7 <*método*> setLexeme(): Utilizado para modificar o lexeme;
3.8 <*método*> setClassType(): Utilizado para modificar o tipo de token;
3.9 <*método*> setLine(): Utilizado para modificar a quantidade de linhas;
3.10 <*método*> setColumn(): Utilizado para modificar a quantidade de colunas.

4. <*classe*> **Type**: Classe que enumera os tipos de dados lidos no arquivo pelo lexer.
5. <*classe*> **Identifier**: Classe que contém uma string para ser identificadora no HashMap na Tabela de Símbolos (<*classe*> **SymbolTable**).

6. <*classe*> **ErrorMessages**: Classe responsável por fornecer métodos com mensagens de erro.
   6.1 <*método*> openFileError() : Informar erro de abertura do arquivo.
   6.2 <*método*> programError() : Informar erro no programa ou falha na tabela de símbolos.
   6.3 <*método*> closeFileError() : Informar erro ao fechar o programa.
   6.4 <*método*> readFile() : Informar erro na leitura do arquivo.
   6.5 <*método*> lexerError(String message) : Informar erro léxico.
      6.5.1 <*String*> message: Parâmetro para a mensagem de erro.

7. <classe> **Lexer**: Classe responsável pela manipulação do Lexer. Os atributos são:

   - <*atributo*> int EOF: Aponta o final do arquivo.
   - <*atributo*> int lastChar: O último caractere lido.
   - <*atributo*> int line: Número de linhas.
   - <*atributo*> int column: Número de colunas.
   - <*atributo*> TabelaSim TS: Tabela de Símbolos.
   - <*atributo*>RandomAcessFile file_reference: Referência para arquivo de texto.
   - <*atributo*> int estado: Utilizado para transitar entre os estados do autômato.
   - <*atributo*> StringBuilder lexeme: Armazena o lexema.
   - <*atributo*> boolean erroas: Auxilia na verificação do erro multi linhas.

As instâncias realizadas são:

TK = <*classe*> **Token**: Retorna os métodos para manipular as linhas e colunas. TK. <*método*> resetColumn (int column) e outros.

EM = <*classe*> **ErrorMessages**: Retorna as mensagens de erro. EM. <*método*> openFileError(), EM. <*método*> programError() e outros.

7.1 <*método*> Lexer(String file): Abertura do arquivo;

7.1.1 <*String*> file: Recebe como parâmetro o nome do arquivo a ser aberto.

7.2 <*método*> closeFile(): Encerra a instância do arquivo;

7.3 <*método*> returnCharPosition(): Retorna uma posição de leitura de char;

7.4 <*método*> proxToken(): Avança a leitura do token.

7.5 <método>AddToken(Type type): Adiciona os tokens na tabela de símbolos.

7.6 <método>AddToken(Type type, String sequence): Adiciona os tokens na tabela de símbolos.

7.7 <método> IsASCII (Character c): Expressão regular para verificar se o char é ASCII.

7.7.1 <Character> c: Parâmetro que abrange o char lido.

8. <classe> **Parser**: Classe responsável por receber os tokens e verificar a ordem dos mesmos. Os atributos são:

- <*atributo*> final Lexer: Responsável por receber o lexer..
- <*atributo*> Token token: Recebe os tokens enviados pelo lexer.
- <*atributo*> int erros: Contador de erros do parser.
- <*atributo*> ArrayList<Type> tokensSincronizantes: Armazena os tokens sincronizantes para o modo pânico.

8.1 <*método*> Parser(Lexer lexer): Requisição do token;

8.1.1 <Lexer> lexer: Recebe o token do lexer.

8.2 <*método*> fechaArquivos(): Fecha o arquivo no final da leitura;

8.3 <*método*> erroSintatico(String mensagem): Exibe os erros sintáticos;

8.3.1 <String> mensagem: Armazena a mensagem de erro.

8.4 <*método*> advance(): Avança o token;

8.5 <*método*> eat(Type t): Consome o token e avança a entrada;

8.5.1 <Type> t: Armazena o token.

8.6 <*método*> Skip(String mensagem): Mostra o erro e avança a entrada;

8.6.1 <String> mensagem: Armazena a mensagem de erro.

8.7 <*método*> sincronizaToken(String mensagem): Responsável por procurar um token sincronizante, caso não ache chama o método skip;

8.7.1 <String> mensagem: Armazena a mensagem de erro.

8.8 <*método*> Prog(): Primeira regra da gramática, consome os tokens “program”, “id” e chama o método Body();

8.9 <*método*> Body(): Chama o método Decllist(), consome o token “{“, chama o método Stmtlist() e consome “}”;

8.10 <*método*> Decllist(): Chama o método Decl(), consome o token ";", chama o método Decllist(). Pode se tornar vazio;
8.11 <*método*> Decl(): Chama o método Type() e o método Idlist();
8.12 <*método*> Type(): Consome o token "num" ou "char";
8.13 <*método*> Idlist(): Consome o token "id" e chama o método Idlistlinha();
8.14 <*método*> Idlistlinha(): Consome "," e chama o método Idlist(). Pode se tornar vazio;
8.15 <*método*> Stmtlist(): Chama o método Stmt(), consome ";" e chama o método Stmtlist(). Pode se tornar vazio;
8.16 <*método*> Stmt(): Chama o método Assignstmt() ou Ifstmt() ou Whilestmt() ou Readstmt() ou Writestmt();
8.17 <*método*> Assignstmt(): Consome "id" e "=" e chama o método Simpleexpr();
8.18 <*método*> IfStmt(): Consome "if" e "(" , chama o método Condition(), consome ")" e "{", chama o método Stmtlist() e consome "}";
8.19 <*método*> Ifstmtlinha(): Consome "else" e "{", chama o método Stmtlist e consome "}". Pode se tornar vazio;
8.20 <*método*> Condition(): Chama o método Expression();
8.21 <*método*> Whilestmt(): Chama o método Stmtprefix(), consome "{", chama o método Stmtlist() e consome "}";
8.22 <*método*> Stmtprefix(): Consome "while" e "(", chama o método Condition() e consome ")";
8.23 <*método*> Readstmt(): Consome "read" e "id";
8.24 <*método*> Writestmt(): Consome "write" e chama o método Writable();
8.25 <*método*> Writable(): Chama o método Simpleexpr() ou consome "literal";
8.26 <*método*> Expression: Chama o método Simpleexpr() e o método Expressionlinha();
8.27 <método> Expressionlinha(): Chama o método Relop() e o método Simpleexpr(). Pode se tornar vazio;
8.28 <*método*> Simpleexpr(): Chama o método Term() e Simpleexprlinha();
8.29 <*método*> Simpleexprlinha(): Chama o método Addop(), Term() e Simpleexprlinha(). Pode se tornar vazio;
8.30 <*método*> Term(): Chama o método Factora() e Termlinha();
8.31 <*método*> Termliha(): Chama o método Mulop(), Factora() e Termlinha(). Pode se tornar vazio;
8.32 <*método*> Factora(): Chama o método Factor() ou consome "not" e chama o método Factor();
8.33 <*método*> Factor(): Consome "id" ou "(" Expression() ")" ou chama o método Constant();

8.34 <*método*> Relop(): Consome “==” ou “>” ou “>=” ou “<” ou “<=” ou “!=”;

8.35 <*método*> Addop(): Consome “+” ou “-” ou “or”;

8.36 <*método*> Mulop(): Consome “*” ou “/” ou “and”;

8.37 <*método*> Constant(): Consome “char_const” ou “num_const”.