**RELATÓRIO**

**ESTRUTURA DO PROJETO**

**root**

**PasC**

**Documentation** # Documentação do projeto

**Automata** # Desenho do autômato e projeto do JFLAP

**Reports** # Relatórios

**Tests** # Testes realizados

**Models** # Modelos e estruturas de dados

**Modules** # Módulos do compilador

**Properties** # Informações da solução

**Program.cs** # Classe principal

**pasc\_test.pc** # Arquivo de teste geral

**1. Models**

O *namespace* Models guarda todos os modelos e estruturas de dados do compilador, aqui é onde ficam as classes para controle da tabela de símbolos, tokens, tags e identificadores.

* 1. **Grammar.cs**

Regex **DIGIT:** Expressão regular para verificação de dígitos.

Regex **LETTER:** Expressão regular para verificação de letras.

Regex **NUMCONST:** Expressão regular para verificar decimais e não decimais.

Regex **CHARCONST:** Expressão regular para verificar char entre aspas simples.

Regex **LITERAL:** Expressão regular para verificar string entre aspas duplas.

Regex **ID:** Expressão regular para verificar um identificador.

Dictionary **SYMBOL\_TABLE<Token, Identifier>:** Estrutura hash que guarda a tabela de símbolos. O Token é a *key* e o Identifier é o *value*, todos os símbolos que serem adicionados terão que possuir estes dois parâmetros.

Método **Show()**

Imprime a SYMBOL\_TABLE no console.

Método **Add(Token key, Identifier value)**

Adiciona um novo símbolo na SYMBOL\_TABLE de acordo com os parâmetros *key* e *value*.

Método **Identifier GetID(Token key)**

Retorna a ID do identificador na SYMBOL\_TABLE de acordo com a *key* passada por um token.

Método **Token GetToken(String lexeme)**

Retorna um token da SYMBOL\_TABLE de acordo com um lexema e se o mesmo já está contido na tabela.

**1.2 Identifier.cs**

String **Tag:** Recebe o tipo do identificador.

**1.3 Tag.cs**

enum **Tag:** Identificadores enumerados para serem usados em conjunto com a tabela de símbolos.

**1.4 Token.cs**

int **Row:** Propriedade da linha.

Tag **TTag:** Propriedade do tipo de acordo com o **enum Tag**.

int **Column:** Propriedade da coluna.

String **Lexeme:** Propriedade do lexema.

Método **Token(Tag TTag, String Lexeme, int Row, int Column)**

Construtor do token.

Método Override **ToString()**

Sobrescreve o ToString() padrão para retornar a saída do token formatada.

**2. Modules**

O *namespace* Modules guarda todos os módulos do compilador, como o Analisador Léxico, Sintático e Semântico.

**2.1 Lexer.cs**

char **CURRENT\_CHAR:** Caractere atual que está sendo passado para o autômato.

StringBuilder **LEXEME:** Lexema atual que está sendo moldado pelo autômato.

int **ROW:** Linha atual.

int **COLUMN:** Coluna atual.

int **LAST\_CHAR:** Caractere antecessor do CURRENT\_CHAR.

char **LAST\_CHAR\_CHECK:** Controle auxiliar para contagem de linhas.

readonly int **EOF:** Constante para verificação de fim de arquivo.

int **STATE:** Estado atual do autômato.

FileStream **SOURCE:** Arquivo fonte que foi passado como parâmetro.

Método **Set(string source)**

Abre e transforma o arquivo *source* passado como parâmetro e inicia o autômato a procura de tokens a partir do primeiro caractere lido. Grava os tokens na SYMBOL\_TABLE que são retornados do método *NextToken()*.

Método **Read()**

Avança um caractere do arquivo, conta linhas e colunas e verifica fim de arquivo.

Método **Restart()**

Volta um caractere e reinicia o autômato a partir do estado inicial.

Método bool **IsASCII(char c)**

Verifica se um char está contido na tabela ASCII.

Método bool **IsNewLine()**

Verifica o se o CURRENT\_CHAR é um char especial de quebra de linha de acordo com o sistema operacional.

Método bool **IsNotSpecialChar()**

Verifica se o CURRENT\_CHAR não é uma sequência de e*scape* especial.

Método **LexicalError(string message)**

Escreve no console uma mensagem de erro léxico formatada a partir do parâmetro *message*.

Método **MultilineCommentErrorCheck()**

Verifica se o comentário de várias linhas não foi fechado ao chegar ao EOF, essa verificação é chamada no método Read().

Método Token **NextToken()**

Controla as condições de estados do autômato a partir do CURRENT\_CHAR lido, monta o lexema atual e retorna tokens encontrados em estados finais para o método *Set(string source)*.

Método **SetState(int currentState, bool appendLexeme)**

Seta o estado atual e adiciona o CURRENT\_CHAR ao LEXEME se solicitado pelo parâmetro *appendLexeme*.

Método **GetLexeme()**

Retorna o lexema atual.

**3. Program.cs**

Classe principal do projeto, onde é realizado o controle do parâmetro do arquivo fonte passado pelo usuário, para iniciar o processo de compilação.