

UNIVERSIDADE FEDERAL DE SERGIPE
CENTRO DE CIENCIAS EXATAS E TECNOLOGIA
DEPARTAMENTO DE COMPUTAÇÃO

Relatório do projeto Compilador Portugol da disciplina *Linguagens Formais e tradutores*, ministrada pelo Prof. Kalil Araujo Bispo, feito pelos alunos **Gilcley de Carvalho Silva** e Jéssica **Profeta da Silveira**.

São Cristóvão
08 de Agosto

Introdução

O SableCC é um framework orientado a objetos para o desenvolvimento de linguagens de programação e implementado em Java. Com ele é possível gerar automaticamente o analisador léxico e o analisador sintático a partir de um documento de especificação da linguagem. Além disso, utiliza uma versão estendida do Visitor. Após a execução SableCC são gerados quatro subdiretórios: *Lexer* (Análise léxica), *Parser* (Análise Sintática), *Node*, *Analysis* (Análise Contextual).

O objetivo deste trabalho é a criação da parte léxica, sintática e semântica de um compilador da linguagem Portugol como forma de avaliação da disciplina Linguagens Formais e Tradutores. Para isso o SableCC foi utilizado como ferramenta de apoio para geração do analisador léxico e sintático.

Desenvolvimento

1- Gramatica:

A base para criação de um compilador SableCC é a especificação da gramática da linguagem. A gramática é dividida em 4 partes: *Helpers*, *Tokens*, *Productions* e *Abstract Syntax Tree*(AST).

Os *Helpers* funcionam como constantes e são utilizados para outras declarações. Os *Tokens* são as definições dos terminais e “Tokens” para serem utilizadas nas produções. As *Productions* são as produções que definem as estruturas da linguagem e como essas estruturas se relacionam. E a AST cria a árvore sintática abstrata das produções. Em nosso projeto chamamos nossa gramática de “**portugol.grammar**” e ela é composta por todas essas partes.

2- Diretórios e organização:

Após a definição da gramática, de acordo com a especificação do Portugol, ao executar o SableCC foram gerados quatro diretórios como descrito anteriormente- *Lexer* (Análise léxica), *Parser* (Análise Sintática), *Node*, *Analysis* (Análise Contextual) - dentro do pacote “Portugol” como definido na gramática.

Criamos um pacote “**Main**” contendo os subdiretórios “**Sintático**” e “**Lexico**”. O sintático contém o **MainSintático.java**, e o lexico contém o **Mainlexico**. Cada classe main irá passar um arquivo txt como argumento contendo programas escritos na linguagem portugol para serem testados.

Outro pacote criado foi “**ArquivosTeste**” que contém todos os arquivos de teste utilizados separados por léxico ou sintático.

Temos também um pacote denominado “**AnaliseLexica**” contendo a classe **Mylexer.java** que trata dos comentários aninhados.

Adicionamos um pacote “**Doc**” contendo a especificação da linguagem e este documento.

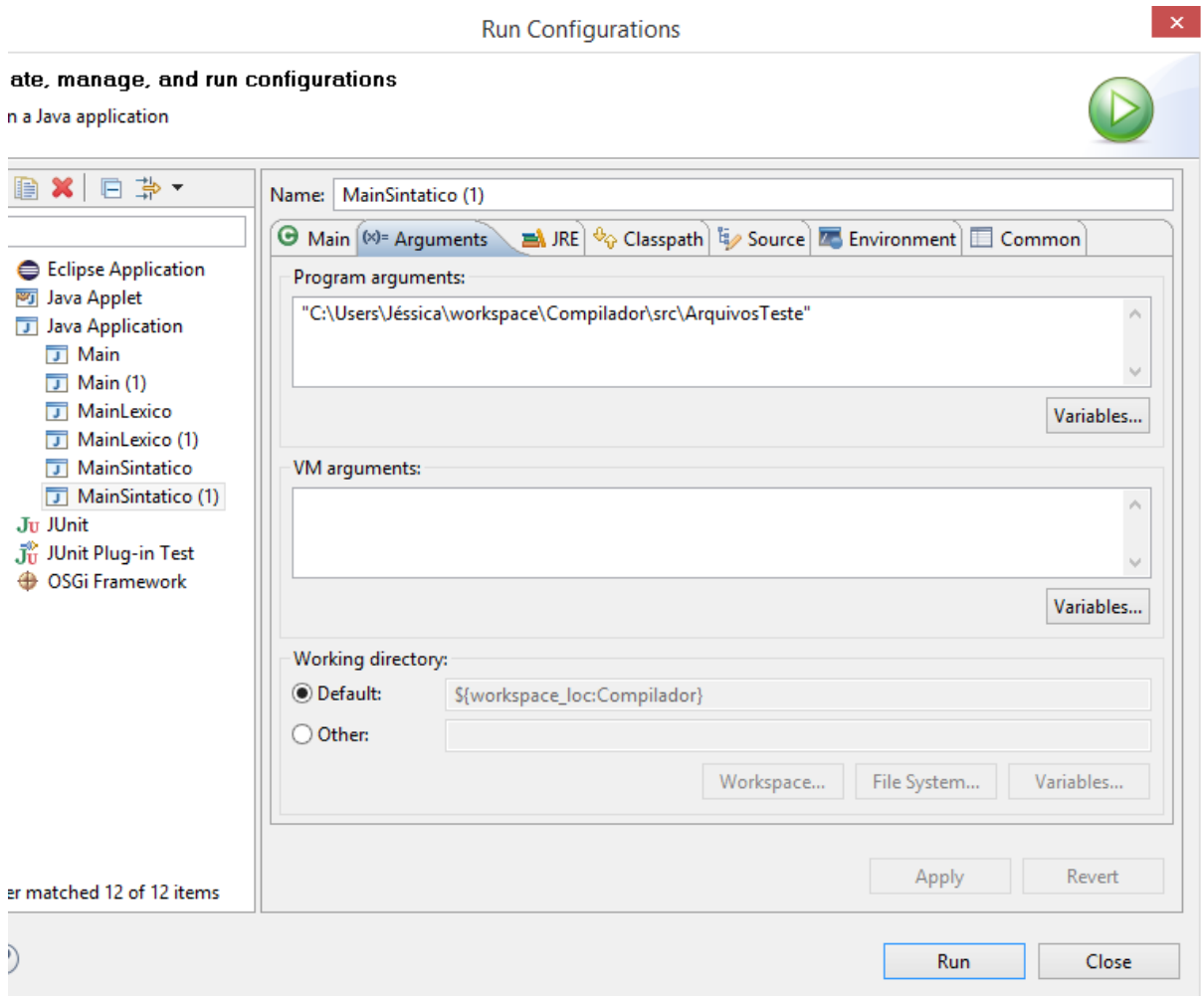
Resumindo, a organização dos diretórios é da seguinte forma:

1- Portugol

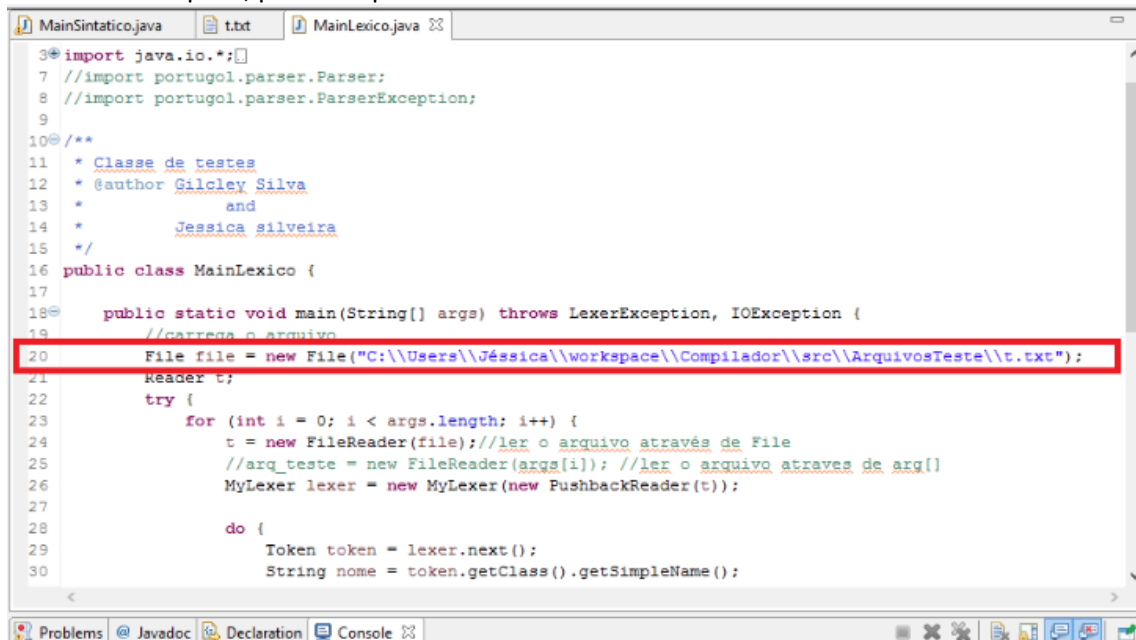
- 2- Main
- 3- AnaliseLexica
- 4- ArquivosTeste
- 5- Doc

Como rodar os arquivos da pasta Main:

Clica com botão direito do main que quer rodar, vai na opção “*Run Configurations*”. Nessa opção deve-se especificar o caminho do arquivo que se quer testar no guia “*Arguments*”. Em nosso projeto os arquivos de teste ficam na pasta ArquivosTeste.



Na linha 20 de cada main, tanto em Mainlexico.java quanto em MainSintatico.java deve-se colocar o caminho do arquivo, por exemplo:

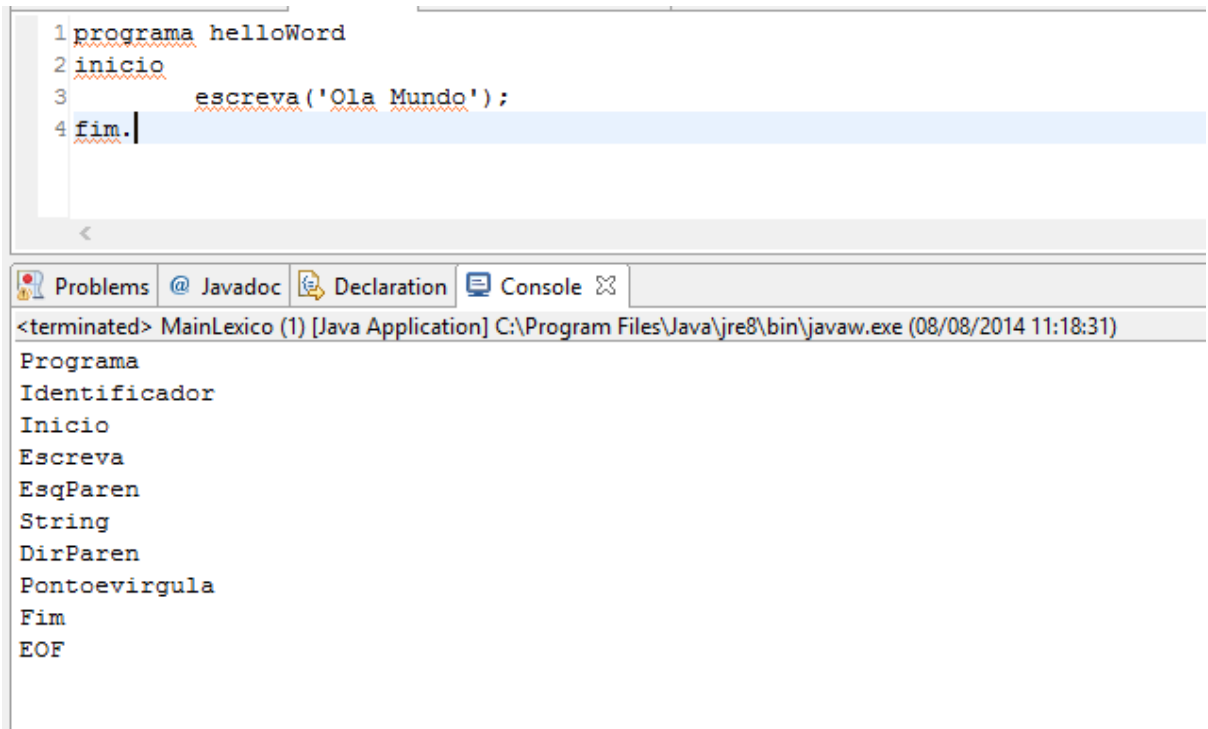


```
3* import java.io.*;
7 //import portugol.parser.Parser;
8 //import portugol.parser.ParserException;
9
10 /**
11  * Classe de testes
12  * @author Gilcley Silva
13  * and
14  * Jessica silveira
15  */
16 public class MainLexico {
17
18     public static void main(String[] args) throws LexerException, IOException {
19         //carrega o arquivo
20         File file = new File("C:\\Users\\Jéssica\\workspace\\Compilador\\src\\ArquivosTeste\\t.txt");
21         Reader t;
22         try {
23             for (int i = 0; i < args.length; i++) {
24                 t = new FileReader(file); //ler o arquivo através de File
25                 //arq_teste = new FileReader(args[i]); //ler o arquivo através de arg[]
26                 MyLexer lexer = new MyLexer(new PushbackReader(t));
27
28                 do {
29                     Token token = lexer.next();
30                     String nome = token.getClass().getSimpleName();
```

Resultados Obtidos

1- Análise léxica:

Nessa etapa o arquivo é lido caractere por caractere e os *tokens* são impressos linha por linha. Os tokens ignorados em nossa linguagem foram os comentários e espaços em branco.



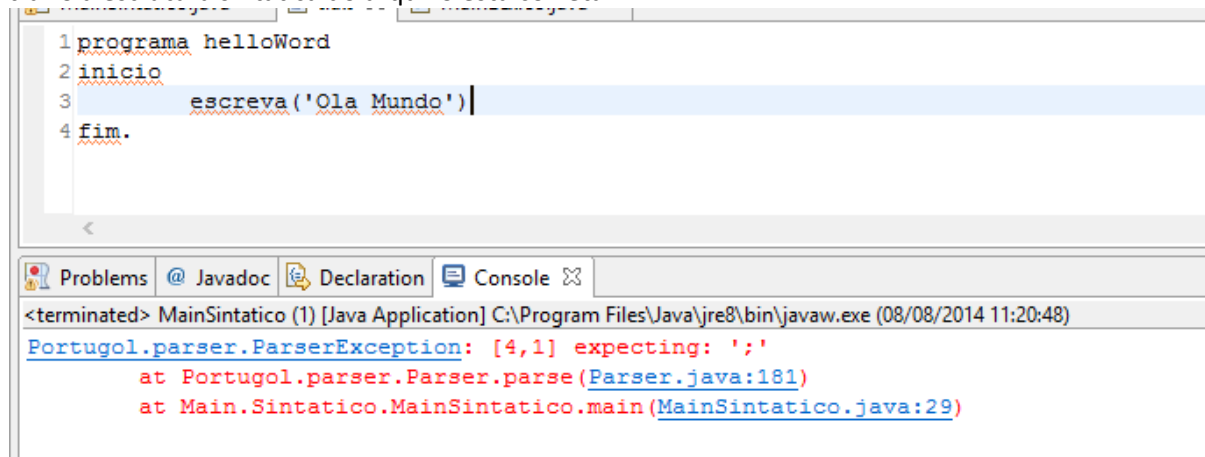
```
1 programa helloWord
2 inicio
3 escreva('Ola Mundo');
4 fim.
```

<terminated> MainLexico (1) [Java Application] C:\Program Files\Java\jre8\bin\javaw.exe (08/08/2014 11:18:31)

Programa
Identificador
Inicio
Escreva
EsqParen
String
DirParen
Pontoevirgula
Fim
EOF

2- Analise Sintática Concreta:

Nessa etapa o arquivo é lido, mas leva-se em consideração a estrutura. Ou seja, se a estrutura do arquivo lido for diferente do que é esperado um erro é acusado, caso contrário a estrutura sintática do arquivo está correta.



The screenshot shows an IDE with a Java file named `MainSintatico.java`. The code is as follows:

```
1 programa helloWord
2 inicio
3 escreva('Ola Mundo')
4 fim.
```

The IDE's console shows the following error message:

```
<terminated> MainSintatico (1) [Java Application] C:\Program Files\Java\jre8\bin\javaw.exe (08/08/2014 11:20:48)
Portugol.parser.ParserException: [4,1] expecting: ';'
    at Portugol.parser.Parser.parse(Parser.java:181)
    at Main.Sintatico.MainSintatico.main(MainSintatico.java:29)
```

3- Analise Sintática Abstrata:

Usada como referência na descrição semântica do programa. Não gera frases, mas se baseia na estrutura das frases do programa. Cada nó representa uma produção, com uma sub-árvore para cada subfrase.

```
exp_logica = {nlog} nao exp_logica|
              {oubin} [esq]:exp_logica [dir]:exp_logica|
              {xorbin}[esq]:exp_logica [dir]:exp_logica|
              {ebin}[esq]:exp_logica [dir]:exp_logica|
              {igual} [esq]:exp [dir]:exp|
              {diferente}[esq]:exp [dir]:exp|
              {maior} [esq]:exp [dir]:exp|
              {maior_igual}[esq]:exp [dir]:exp|
              {menor}[esq]:exp [dir]:exp|
              {menor_igual} [esq]:exp [dir]:exp;
```

4- Analise Semântica:

Um exemplo da análise semântica é verificar se uma variável que está sendo usada foi declarada ou não. Além disso, ela também pode descrever regras de escopo, regras de visibilidade e consistência de tipos. Infelizmente não foi possível completar essa etapa até a data de entrega do trabalho.

