LINGUAGEM MINI PASCAL

Guilherme Maganha Moreira 6284 - 4ºA

Rafael Cossovan de França 5536 - 4ºA

Índice

[Expressões Regulares (ER’s) 4](#_Toc340789744)

[BNF (Backus-Naur Form) 5](#_Toc340789745)

[EBNF (Extended Backus-Naur Form) 6](#_Toc340789746)

[Autômatos 7](#_Toc340789747)

[inicio 7](#_Toc340789748)

[comando 7](#_Toc340789749)

[condicional 7](#_Toc340789750)

[iterativo 8](#_Toc340789751)

[atribuição 8](#_Toc340789752)

[condição 8](#_Toc340789753)

[lógica 8](#_Toc340789754)

[identificador 8](#_Toc340789755)

[letra 8](#_Toc340789756)

[digito 9](#_Toc340789757)

[expressão 9](#_Toc340789758)

[operador 9](#_Toc340789759)

[número 9](#_Toc340789760)

[Código Fonte 10](#_Toc340789761)

[Técnica de Reconhecimento Sintático 11](#_Toc340789762)

[Testes 12](#_Toc340789763)

[Teste 1 12](#_Toc340789764)

[Código fonte 12](#_Toc340789765)

[Extração e classificação dos tokens 13](#_Toc340789766)

[Tabela de símbolos 14](#_Toc340789767)

[Árvore sintática abstrata 15](#_Toc340789768)

[Teste 2 16](#_Toc340789769)

[Código fonte 16](#_Toc340789770)

[Extração e classificação dos tokens 17](#_Toc340789771)

[Tabela de símbolos 19](#_Toc340789772)

[Árvore sintática abstrata 20](#_Toc340789773)

[Teste 3 23](#_Toc340789774)

[Código fonte 23](#_Toc340789775)

[Extração e classificação dos tokens 24](#_Toc340789776)

[Tabela de símbolos 26](#_Toc340789777)

[Árvore sintática abstrata 27](#_Toc340789778)

[Teste 4 29](#_Toc340789779)

[Código fonte 29](#_Toc340789780)

[Extração e classificação dos tokens 30](#_Toc340789781)

[Tabela de símbolos 31](#_Toc340789782)

[Árvore sintática abstrata 32](#_Toc340789783)

[Teste 5 34](#_Toc340789784)

[Código fonte 34](#_Toc340789785)

[Extração e classificação dos tokens 35](#_Toc340789786)

[Tabela de símbolos 36](#_Toc340789787)

[Árvore sintática abstrata 37](#_Toc340789788)

[Conclusão 39](#_Toc340789789)

# Expressões Regulares (ER’s)

*inicio* -> **begin** {*comando*}\* **end**\**.**

*comando* -> ({*condicional*}|{*iterativo*}|{*atribuicao*})?

*condicional* -> **if** {*condicao*} **then** {*atribuicao*}\* {*comando*}**;**

*iterativo* -> **while** {*condicao*} **do** {*atribuicao*}\* {*comando*}**;**

*atribuicao* -> {*identificador*} **:=** ({*expressao*}|{*identificador*}) ({*operador*} ({*expressao*}|{*identificador*}))\***;**

*condicao* -> {*identificador*} {*logica*} {*identificador*}

*logica* -> (**==**|**<=**|**>=**|**>**|**<**)

*identificador* -> {*letra*}({*letra*}|{*digito*})\*

*letra* -> [**a**-**zA**-**Z**]

*digito* -> [**0**-**9**]

*expressao* -> {*numero*}|\({*expressao*}{*operador*}{*expressao*}\)

*operador* -> (\**+**|\**\***|\**\**|**MOD**)

*numero* -> {*digito*}+|\**.**{*digito*}+|{*digito*}+\**.**{*digito*}+

|  |  |
| --- | --- |
| *Legenda* | |
| *{expressão} (itálico)* | Faz referência a outra ER. |
| **palavra (negrito)** | Palavra reservada (constante). |

# BNF (Backus-Naur Form)

<*inicio*> ::= **begin** <*comando*>\* **end.**

<*comando*> ::= (<*condicional*>|<*iterativo*>|<*atribuicao*>|ε)

<*condicional*> ::= **if** <*condicao*> **then** <*atribuicao*>\* <*comando*>**;**

<*iterativo*> ::= **while** <*condicao*> **do** <*atribuicao*>\* <*comando*>**;**

<*atribuicao*> ::= <*identificador*> **:=** (<*expressao*>|<*identificador*>) (<*operador*> (<*expressao*>|<*identificador*>))\***;**

<*condicao*> ::= <*identificador*> <*logica*> <*identificador*>

<*logica*> ::= (**>**|**<**|**==**|**<=**|**>=**)

<*identificador*> ::= <*letra*>(<*letra*>|<*digito*>)\*

<*letra*> ::= (**a**|**b**|**c**|**d**|**e**|**f**|**g**|**h**|**i**|**j**|**k**|**l**|**m**|**n**|**o**|**p**|**q**|**r**|**s**|**t**|**u**|**v**|**x**|**y**|**w**|**z**|**A**|**B**|**C**|**D**|**E**|**F**|**G**|**H**|**I**|**J**|**K**|**L**|**M**|**N**|**O**|**P**|**Q**|**R**|**S**|**T**|**U**|**V**|**X**|**Y**|**W**|**Z**)

<*digito*> ::= (**0**|**1**|**2**|**3**|**4**|**5**|**6**|**7**|**8**|**9**)

<*expressao*> ::= <*numero*>|(<*expressao*> <*operador*> <*expressao*>)

<*operador*> ::= (**+**|**\***|**/**|**MOD**)

<*numero*> ::= <*digito*>+|**.**<*digito*>+|<*digito*>+**.**<*digito*>+

|  |  |
| --- | --- |
| *Legenda* | |
| *<expressão> (itálico)* | Faz referência a outro BNF. |
| **palavra (negrito)** | Palavra reservada (constante). |

# EBNF (Extended Backus-Naur Form)

|  |
| --- |
|  |

*inicio* = “**begin**” {*comando*} “**end.”**

*comando* = [*condicional*|*iterativo*|*atribuição*]

*condicional* = “**if** ” *condicao* “**then**” {*atribuição*} *comando”***;**”

*iterativo* = “**while**” *condicao* “**do**” {*atribuição*} *comando”***;**”

*atribuicao* = *identificador* “**:=**” (*expressao*|*identificador*) {*operador* (*expressao*|*identificador*)}”**;**”

*condicao* = *identificador* *logica* *identificador*

*logica* = (“**<”**|”**>”**|”**==”**|”**<=”**|”**>=”**)

*identificador* = *letra*{*letra*|*digito*}

*letra* = (“**a**”|”**b**”|”**c**”|”**d**”|”**e**”|”**f**”|”**g**”|”**h**”|”**i**”|”**j**”|”**k**”|”**l**”|”**m**”|”**n**”|”**o**”|”**p**”|”**q**”|”**r**”|”**s**”|”**t**”|”**u**”|”**v**”|”**x**”|”**y**”|”**w**”|”**z**”|”**A**”|”**B**”|”**C**”|”**D**”|”**E**”|”**F**”|”**G**”|”**H**”|”**I**”|”**J**”|”**K**”|”**L**”|”**M**”|”**N**”|”**O**”|”**P**”|”**Q**”|”**R**”|”**S**”|”**T**”|”**U**”|”**V**”|”**X**”|”**Y**”|”**W**”|”**Z**”)

*digito* = (“**0**”|”**1**”|”**2**”|”**3**”|”**4**”|”**5**”|”**6**”|”**7**”|”**8**”|”**9**”)

*expressao* = *numero*|(*expressao* *operador* *expressao*)

*operador* = (“**+**”|“**\***”|“**/**”|“**MOD**”)

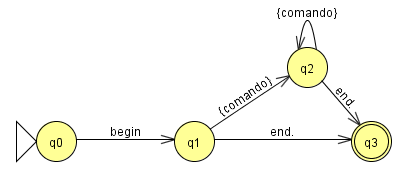
*numero* = *digito*+|”**.”***digito*+|*digito*+”**.**”*digito*+

|  |  |
| --- | --- |
| *Legenda* | |
| *expressão (itálico)* | Faz referência a outro EBNF. |
| **palavra (negrito)** | Palavra reservada (constante). |

# Autômatos

|  |  |
| --- | --- |
| **Legenda** | |
| C:\Users\Rafael\Desktop\leg1.png | O autômato do estado q0 para o q1 consome os símbolos descritos no autômato “expressao”.  Para evitar que o autômato fique complexo. |
| C:\Users\Rafael\Desktop\leg2.png | O autômato do estado q0 para o q1 consome os símbolos especificados pela expressão regular.  Para evitar a criação de várias transições. |

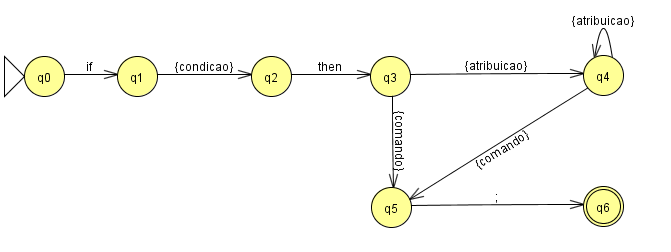
## inicio



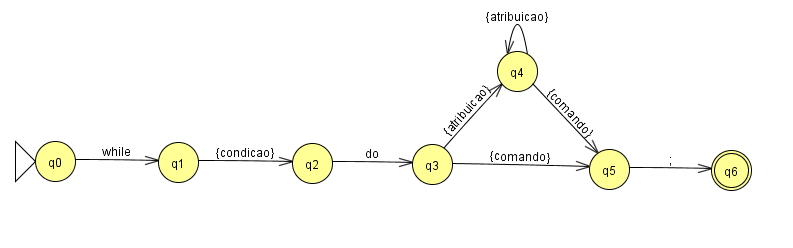
## comando



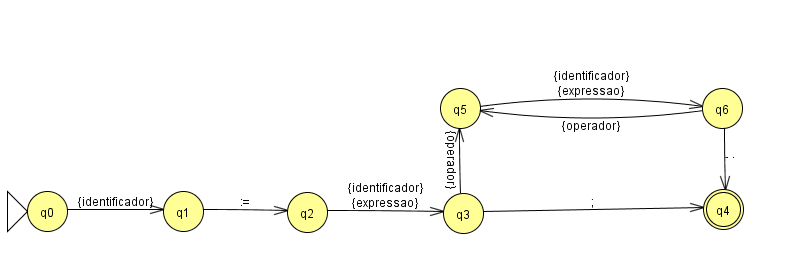
## condicional



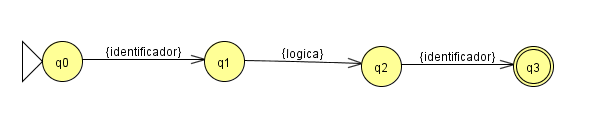
## iterativo



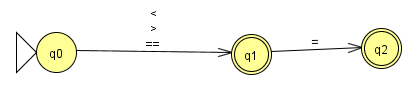
## atribuição



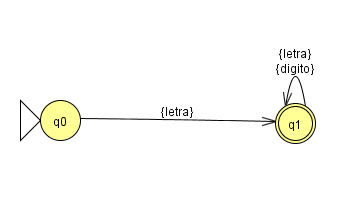
## condição



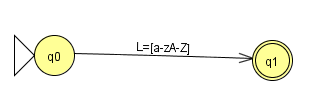
## lógica



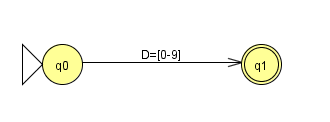
## identificador



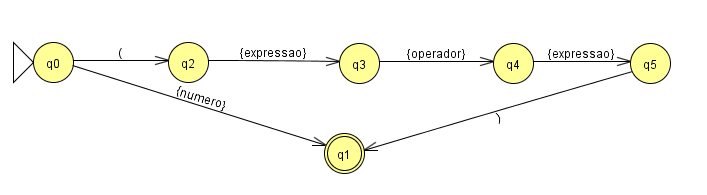
## letra



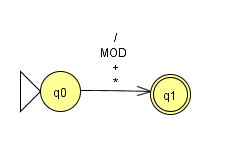
## digito



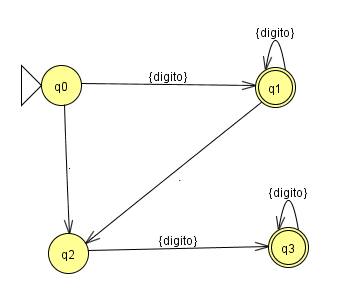
## expressão



## operador



## número



# Código Fonte

O projeto foi criado no [*Eclipse*](http://www.eclipse.org/) usando o [*JavaCC Eclipse Plug-in*](http://eclipse-javacc.sourceforge.net/) que facilita o desenvolvimento de projetos utilizando o [*JavaCC*](http://javacc.java.net/)*.*

O código fonte do projeto está localizado na pasta javacc, enquanto os exemplos da linguagem mini pascal estão na pasta samples no CD.

Foi utilizado o [*JJTree*](http://javacc.java.net/doc/JJTree.html) que é um complemento do *JavaCC* para facilitar a criação da árvore sintática.

Foi utilizado o [*JUNG Framework*](http://jung.sourceforge.net/) para a criação visual da árvore sintática abstrata.

O arquivo LMP.jjt é o arquivo principal do projeto que contém a definição da gramática e da sintaxe da linguagem usando as regras do *JavaCC*.

Este arquivo é processado pelo *JJTree* que gera as classes necessárias para a construção da árvore e cria o arquivo LMP.jj com o código necessário para inserir os nós na árvore.

Por fim, o arquivo LMP.jj é processado pelo *JavaCC* que gera as classes do analisador léxico e sintático, todo esse processo é realizado automaticamente pelo plug-in.

SimpleNode2.java – Responsável por armazenar a árvore sintática abstrata e fornecer algumas funcionalidades extras para a manipulação da árvore. Essa classe estende SimpleNode.java gerada pelo *JJTree*.

No pacote tools, estão as classes responsáveis por extrair informações do código fonte de entrada (linguagem mini pascal).

AST.java – Responsável por criar o grafo da árvore sintática abstrata (visual).

Report.java – Responsável por extrair e classificar os símbolos (tokens).

Symbols.java – Responsável por gerar a lista de identificadores (variáveis).

# Técnica de Reconhecimento Sintático

No projeto foi utilizado o analisador sintático do tipo LL(1) que é o analisador padrão do *JavaCC*, entretanto, na definição da sintaxe do comando iterativo e condicional foi utilizado LL(2) (Lookahead = 2), devido ao fato desses comandos utilizarem o comando atribuição zero ou mais vezes, fazendo com que o analisador não pudesse prever qual seria o token esperado.

Por isso, usando o LL(2) nesses dois comandos, o analisador consegue tomar decisões melhores, pois sabe qual é o próximo token a ser utilizado, evitando a necessidade do backtracking.

# Testes

## Teste 1

### Código fonte

Exemplo descrito no PDF do projeto.

|  |
| --- |
| begin  if a > b then  b:=b+1;  ;  end. |

### Extração e classificação dos tokens

|  |  |  |  |
| --- | --- | --- | --- |
| Linha | Coluna | Token | Classificação |
| 1 | 1 | begin | palavra reservada |
| 2 | 9 | if | palavra reservada |
| 2 | 12 | a | identificador |
| 2 | 14 | > | símbolo especial |
| 2 | 16 | b | identificador |
| 2 | 18 | then | palavra reservada |
| 3 | 17 | b | identificador |
| 3 | 18 | := | símbolo especial |
| 3 | 20 | b | identificador |
| 3 | 21 | + | símbolo especial |
| 3 | 22 | 1 | numero |
| 3 | 23 | ; | símbolo especial |
| 4 | 9 | ; | símbolo especial |
| 5 | 1 | end | palavra reservada |
| 5 | 4 | . | símbolo especial |

### Tabela de símbolos

|  |
| --- |
| Símbolos |
| a |
| b |

### Árvore sintática abstrata

|  |
| --- |
| inicio  begin  comando  condicional  if  condicao  identificador  a  logica  >  identificador  b  then  atribuicao  identificador  b  :=  identificador  b  operador  +  expressao  1  ;  comando  ;  end  . |

## Teste 2

### Código fonte

Conta quantos números primos existem entre 2 e 100 (variável max).

|  |
| --- |
| begin  primos := 0;  i := 2;  max := 100;  zero := 0;  while i <= max do  j := 2;  primo := 0;  while j < i do  resto := i MOD j;  if resto == zero then  if j < i then  primo := 1;  ;  ;  j := j + 1;  ;  if primo == zero then  primos := primos + 1;  ;  i := i + 1;  ;  end. |

### Extração e classificação dos tokens

|  |  |  |  |
| --- | --- | --- | --- |
| Linha | Coluna | Token | Classificação |
| 1 | 1 | begin | palavra reservada |
| 2 | 9 | primos | identificador |
| 2 | 16 | := | símbolo especial |
| 2 | 19 | 0 | numero |
| 2 | 20 | ; | símbolo especial |
| 3 | 9 | i | identificador |
| 3 | 11 | := | símbolo especial |
| 3 | 14 | 2 | numero |
| 3 | 15 | ; | símbolo especial |
| 4 | 9 | max | identificador |
| 4 | 13 | := | símbolo especial |
| 4 | 16 | 100 | numero |
| 4 | 19 | ; | símbolo especial |
| 5 | 9 | zero | identificador |
| 5 | 14 | := | símbolo especial |
| 5 | 17 | 0 | numero |
| 5 | 18 | ; | símbolo especial |
| 6 | 9 | while | palavra reservada |
| 6 | 15 | i | identificador |
| 6 | 17 | <= | símbolo especial |
| 6 | 20 | max | identificador |
| 6 | 24 | do | palavra reservada |
| 7 | 17 | j | identificador |
| 7 | 19 | := | símbolo especial |
| 7 | 22 | 2 | numero |
| 7 | 23 | ; | símbolo especial |
| 8 | 17 | primo | identificador |
| 8 | 23 | := | símbolo especial |
| 8 | 26 | 0 | numero |
| 8 | 27 | ; | símbolo especial |
| 9 | 17 | while | palavra reservada |
| 9 | 23 | j | identificador |
| 9 | 25 | < | símbolo especial |
| 9 | 27 | i | identificador |
| 9 | 29 | do | palavra reservada |
| 10 | 25 | resto | identificador |
| 10 | 31 | := | símbolo especial |
| 10 | 34 | i | identificador |
| 10 | 36 | MOD | símbolo especial |
| 10 | 40 | j | identificador |
| 10 | 41 | ; | símbolo especial |
| 11 | 25 | if | palavra reservada |
| 11 | 28 | resto | identificador |
| 11 | 34 | == | símbolo especial |
| 11 | 37 | zero | identificador |
| 11 | 42 | then | palavra reservada |
| 12 | 33 | if | palavra reservada |
| 12 | 36 | j | identificador |
| 12 | 38 | < | símbolo especial |
| 12 | 40 | i | identificador |
| 12 | 42 | then | palavra reservada |
| 13 | 41 | primo | identificador |
| 13 | 47 | := | símbolo especial |
| 13 | 50 | 1 | numero |
| 13 | 51 | ; | símbolo especial |
| 14 | 33 | ; | símbolo especial |
| 15 | 25 | ; | símbolo especial |
| 16 | 25 | j | identificador |
| 16 | 27 | := | símbolo especial |
| 16 | 30 | j | identificador |
| 16 | 32 | + | símbolo especial |
| 16 | 34 | 1 | numero |
| 16 | 35 | ; | símbolo especial |
| 17 | 17 | ; | símbolo especial |
| 18 | 17 | if | palavra reservada |
| 18 | 20 | primo | identificador |
| 18 | 26 | == | símbolo especial |
| 18 | 29 | zero | identificador |
| 18 | 34 | then | palavra reservada |
| 19 | 25 | primos | identificador |
| 19 | 32 | := | símbolo especial |
| 19 | 35 | primos | identificador |
| 19 | 42 | + | símbolo especial |
| 19 | 44 | 1 | numero |
| 19 | 45 | ; | símbolo especial |
| 20 | 17 | ; | símbolo especial |
| 21 | 17 | i | identificador |
| 21 | 19 | := | símbolo especial |
| 21 | 22 | i | identificador |
| 21 | 24 | + | símbolo especial |
| 21 | 26 | 1 | numero |
| 21 | 27 | ; | símbolo especial |
| 22 | 9 | ; | símbolo especial |
| 23 | 1 | end | palavra reservada |
| 23 | 4 | . | símbolo especial |

### Tabela de símbolos

|  |
| --- |
| Símbolos |
| i |
| j |
| max |
| primo |
| primos |
| resto |
| zero |

### Árvore sintática abstrata

|  |
| --- |
| inicio  begin  comando  atribuicao  identificador  primos  :=  expressao  0  ;  atribuicao  identificador  i  :=  expressao  2  ;  atribuicao  identificador  max  :=  expressao  100  ;  atribuicao  identificador  zero  :=  expressao  0  ;  iterativo  while  condicao  identificador  i  logica  <=  identificador  max  do  atribuicao  identificador  j  :=  expressao  2  ;  atribuicao  identificador  primo  :=  expressao  0  ;  comando  iterativo  while  condicao  identificador  j  logica  <  identificador  i  do  atribuicao  identificador  resto  :=  identificador  i  operador  MOD  identificador  j  ;  comando  condicional  if  condicao  identificador  resto  logica  ==  identificador  zero  then  comando  condicional  if  condicao  identificador  j  logica  <  identificador  i  then  atribuicao  identificador  primo  :=  expressao  1  ;  comando  ;  ;  atribuicao  identificador  j  :=  identificador  j  operador  +  expressao  1  ;  ;  condicional  if  condicao  identificador  primo  logica  ==  identificador  zero  then  atribuicao  identificador  primos  :=  identificador  primos  operador  +  expressao  1  ;  comando  ;  atribuicao  identificador  i  :=  identificador  i  operador  +  expressao  1  ;  ;  end  . |

## Teste 3

### Código fonte

Diversas operações aritméticas.

|  |
| --- |
| begin  if a > b then  b:=b+1;  c:=c\*1;  d:=d+2\*3;  e:=4+5\*6/7+f;  g:=a MOD b;  h:=3\*(1+2)\*4;  ;  end. |

### Extração e classificação dos tokens

|  |  |  |  |
| --- | --- | --- | --- |
| Linha | Coluna | Token | Classificação |
| 1 | 1 | begin | palavra reservada |
| 2 | 9 | if | palavra reservada |
| 2 | 12 | a | identificador |
| 2 | 14 | > | símbolo especial |
| 2 | 16 | b | identificador |
| 2 | 18 | then | palavra reservada |
| 3 | 17 | b | identificador |
| 3 | 18 | := | símbolo especial |
| 3 | 20 | b | identificador |
| 3 | 21 | + | símbolo especial |
| 3 | 22 | 1 | numero |
| 3 | 23 | ; | símbolo especial |
| 4 | 17 | c | identificador |
| 4 | 18 | := | símbolo especial |
| 4 | 20 | c | identificador |
| 4 | 21 | \* | símbolo especial |
| 4 | 22 | 1 | numero |
| 4 | 23 | ; | símbolo especial |
| 5 | 17 | d | identificador |
| 5 | 18 | := | símbolo especial |
| 5 | 20 | d | identificador |
| 5 | 21 | + | símbolo especial |
| 5 | 22 | 2 | numero |
| 5 | 23 | \* | símbolo especial |
| 5 | 24 | 3 | numero |
| 5 | 25 | ; | símbolo especial |
| 6 | 17 | e | identificador |
| 6 | 18 | := | símbolo especial |
| 6 | 20 | 4 | numero |
| 6 | 21 | + | símbolo especial |
| 6 | 22 | 5 | numero |
| 6 | 23 | \* | símbolo especial |
| 6 | 24 | 6 | numero |
| 6 | 25 | / | símbolo especial |
| 6 | 26 | 7 | numero |
| 6 | 27 | + | símbolo especial |
| 6 | 28 | f | identificador |
| 6 | 29 | ; | símbolo especial |
| 7 | 17 | g | identificador |
| 7 | 18 | := | símbolo especial |
| 7 | 20 | a | identificador |
| 7 | 22 | MOD | símbolo especial |
| 7 | 26 | b | identificador |
| 7 | 27 | ; | símbolo especial |
| 8 | 17 | h | identificador |
| 8 | 18 | := | símbolo especial |
| 8 | 20 | 3 | numero |
| 8 | 21 | \* | símbolo especial |
| 8 | 22 | ( | símbolo especial |
| 8 | 23 | 1 | numero |
| 8 | 24 | + | símbolo especial |
| 8 | 25 | 2 | numero |
| 8 | 26 | ) | símbolo especial |
| 8 | 27 | \* | símbolo especial |
| 8 | 28 | 4 | numero |
| 8 | 29 | ; | símbolo especial |
| 9 | 9 | ; | símbolo especial |
| 10 | 1 | end | palavra reservada |
| 10 | 4 | . | símbolo especial |

### Tabela de símbolos

|  |
| --- |
| Símbolos |
| a |
| b |
| c |
| d |
| e |
| f |
| g |
| h |

### Árvore sintática abstrata

|  |
| --- |
| inicio  begin  comando  condicional  if  condicao  identificador  a  logica  >  identificador  b  then  atribuicao  identificador  b  :=  identificador  b  operador  +  expressao  1  ;  atribuicao  identificador  c  :=  identificador  c  operador  \*  expressao  1  ;  atribuicao  identificador  d  :=  identificador  d  operador  +  expressao  2  operador  \*  expressao  3  ;  atribuicao  identificador  e  :=  expressao  4  operador  +  expressao  5  operador  \*  expressao  6  operador  /  expressao  7  operador  +  identificador  f  ;  atribuicao  identificador  g  :=  identificador  a  operador  MOD  identificador  b  ;  atribuicao  identificador  h  :=  expressao  3  operador  \*  expressao  expressao  1  operador  +  expressao  2  operador  \*  expressao  4  ;  comando  ;  end  . |

## Teste 4

### Código fonte

Calcula a função fatorial de 12 (variável max).

|  |
| --- |
| begin  max := 12;  cont := 1;  fatorial := 1;  while cont <= max do  fatorial := fatorial \* cont;  cont := cont + 1;  ;  end. |

### Extração e classificação dos tokens

|  |  |  |  |
| --- | --- | --- | --- |
| Linha | Coluna | Token | Classificação |
| 1 | 1 | begin | palavra reservada |
| 2 | 9 | max | identificador |
| 2 | 13 | := | símbolo especial |
| 2 | 16 | 12 | numero |
| 2 | 18 | ; | símbolo especial |
| 3 | 9 | cont | identificador |
| 3 | 14 | := | símbolo especial |
| 3 | 17 | 1 | numero |
| 3 | 18 | ; | símbolo especial |
| 4 | 9 | fatorial | identificador |
| 4 | 18 | := | símbolo especial |
| 4 | 21 | 1 | numero |
| 4 | 22 | ; | símbolo especial |
| 5 | 9 | while | palavra reservada |
| 5 | 15 | cont | identificador |
| 5 | 20 | <= | símbolo especial |
| 5 | 23 | max | identificador |
| 5 | 27 | do | palavra reservada |
| 6 | 17 | fatorial | identificador |
| 6 | 26 | := | símbolo especial |
| 6 | 29 | fatorial | identificador |
| 6 | 38 | \* | símbolo especial |
| 6 | 40 | cont | identificador |
| 6 | 44 | ; | símbolo especial |
| 7 | 17 | cont | identificador |
| 7 | 22 | := | símbolo especial |
| 7 | 25 | cont | identificador |
| 7 | 30 | + | símbolo especial |
| 7 | 32 | 1 | numero |
| 7 | 33 | ; | símbolo especial |
| 8 | 9 | ; | símbolo especial |
| 9 | 1 | end | palavra reservada |
| 9 | 4 | . | símbolo especial |

### Tabela de símbolos

|  |
| --- |
| Símbolos |
| cont |
| fatorial |
| max |

### Árvore sintática abstrata

|  |
| --- |
| inicio  begin  comando  atribuicao  identificador  max  :=  expressao  12  ;  atribuicao  identificador  cont  :=  expressao  1  ;  atribuicao  identificador  fatorial  :=  expressao  1  ;  iterativo  while  condicao  identificador  cont  logica  <=  identificador  max  do  atribuicao  identificador  fatorial  :=  identificador  fatorial  operador  \*  identificador  cont  ;  atribuicao  identificador  cont  :=  identificador  cont  operador  +  expressao  1  ;  comando  ;  end  . |

## Teste 5

### Código fonte

Calcula a série de Fibonacci.

|  |
| --- |
| begin  u := 0;  v := 1;  i := 2;  while i <= n do  t := u + v;  u := v;  v := t;  i := i + 1;  ;  end. |

### Extração e classificação dos tokens

|  |  |  |  |
| --- | --- | --- | --- |
| Linha | Coluna | Token | Classificação |
| 1 | 1 | begin | palavra reservada |
| 2 | 9 | u | identificador |
| 2 | 11 | := | símbolo especial |
| 2 | 14 | 0 | numero |
| 2 | 15 | ; | símbolo especial |
| 3 | 9 | v | identificador |
| 3 | 11 | := | símbolo especial |
| 3 | 14 | 1 | numero |
| 3 | 15 | ; | símbolo especial |
| 4 | 9 | i | identificador |
| 4 | 11 | := | símbolo especial |
| 4 | 14 | 2 | numero |
| 4 | 15 | ; | símbolo especial |
| 5 | 9 | while | palavra reservada |
| 5 | 15 | i | identificador |
| 5 | 17 | <= | símbolo especial |
| 5 | 20 | n | identificador |
| 5 | 22 | do | palavra reservada |
| 6 | 17 | t | identificador |
| 6 | 19 | := | símbolo especial |
| 6 | 22 | u | identificador |
| 6 | 24 | + | símbolo especial |
| 6 | 26 | v | identificador |
| 6 | 27 | ; | símbolo especial |
| 7 | 17 | u | identificador |
| 7 | 19 | := | símbolo especial |
| 7 | 22 | v | identificador |
| 7 | 23 | ; | símbolo especial |
| 8 | 17 | v | identificador |
| 8 | 19 | := | símbolo especial |
| 8 | 22 | t | identificador |
| 8 | 23 | ; | símbolo especial |
| 9 | 17 | i | identificador |
| 9 | 19 | := | símbolo especial |
| 9 | 22 | i | identificador |
| 9 | 24 | + | símbolo especial |
| 9 | 26 | 1 | numero |
| 9 | 27 | ; | símbolo especial |
| 10 | 9 | ; | símbolo especial |
| 11 | 1 | end | palavra reservada |
| 11 | 4 | . | símbolo especial |

### Tabela de símbolos

|  |
| --- |
| Símbolos |
| i |
| n |
| t |
| u |
| v |

### Árvore sintática abstrata

|  |
| --- |
| inicio  begin  comando  atribuicao  identificador  u  :=  expressao  0  ;  atribuicao  identificador  v  :=  expressao  1  ;  atribuicao  identificador  i  :=  expressao  2  ;  iterativo  while  condicao  identificador  i  logica  <=  identificador  n  do  atribuicao  identificador  t  :=  identificador  u  operador  +  identificador  v  ;  atribuicao  identificador  u  :=  identificador  v  ;  atribuicao  identificador  v  :=  identificador  t  ;  atribuicao  identificador  i  :=  identificador  i  operador  +  expressao  1  ;  comando  ;  end  . |

# Conclusão

Escolhemos desenvolver o analisador da linguagem mini pascal utilizando a linguagem Java em conjunto com o *JavaCC*, pois a ferramenta possui o analisador léxico e sintático integrados, o que facilitou o desenvolvimento do projeto.

Embora o *JavaCC* e o *JJTree* sejam muito flexíveis, nós encontramos alguns problemas durante a especificação da linguagem, como a necessidade de modificar o código gerado, de modo que pudéssemos implementar alguns recursos que o mesmo não fornecia por padrão, por ex: a AST contendo os símbolos terminais e o tratamento de erros.

Decidimos utilizar o *JUNG* para desenhar a árvore sintática abstrata, justamente por possuirmos experiência utilizando-o no projeto anterior (Otimizador de Autômatos Finitos).

Durante o desenvolvimento, colocamos em prática o conhecimento adquirido em sala de aula sobre compiladores, facilitando a compreensão da matéria.