

LINGUAGEM **NAJA**

Trabalho de Compiladores

Douglas A. C. Canevarollo

Gabriel A. P. Nunes

1. **Descrição**

A linguagem NAJA foi desenvolvida tendo como base duas das principais linguagens do mercado: Python e JavaScript.

Abaixo estão descritas as características de sua confecção:

* 1. **Alfabeto**

O alfabeto ∑ consiste de letras do alfabeto inglês, números e alguns caracteres especiais:

∑ = {a, ..., z, A, ..., Z, 0, ..., 9, -, +, \*, /, (, ), ., `, ‘, “, &, |, <, >, =, !, ;, \_}

1. **Analisador léxico**
   1. **Expressões regulares**

As expressões regulares que denotam o analisador léxico da linguagem Naja são as seguintes:

* **Início de programa**: init
* **Final de programa**: end
* **Tipos de variáveis**: integer|string|double|float
* **Vírgula**: ,
* **Ponto e vírgula**: ;
* **Abre parêntese**: [(]
* **Fecha parêntese**: [)]
* **Abre chave**: [{]
* **Fecha chave**: [}]
* **Declaração condicional "se"**: if
* **Declaração condicional "se não"**: else
* **Declaração do laço "enquanto"**: while
* **Impressão**: print
* **Geral**: ([a-zA-Z])([a-zA-Z]|[0-9\\_])\*
* **Texto**: "'".\*"'"
* **Números**: [0-9]+([\.][0-9]+)\*
* **Aritmética**: [+|-|/|\*|(|)]
* **Atribuição de variáveis**: <-|=
* **Comparação**: >=|<=|==|!=|<|>
* **Tabulações**: [" "|\n|\t]+
* **Comentários**: "#".\*"#"
* **Erros específicos**: (~|"?"|@|([0-1]+))([a-zA-Z])\*

As demais cadeias que não forem capturadas por essas expressões regulares acima terão um erro léxico lançado na sua leitura.

* 1. **Tokens gerados**

Ao capturar uma cadeia, um *token* é retornado ao analisador sintático. Os *tokens* capturados e retornados são:

* **Início de programa**: INIT
* **Fim de programa**: END
* **Tipo de variável**: TYPE
* **Vírgula**: COMMA
* **Ponto e vírgula**: SEMICOLON
* **Abre parêntese**: OPN\_PARENTH
* **Fecha parêntese**: CLS\_PARENTH
* **Abre chave**: OPN\_BRACKET
* **Fecha chave**: CLS\_BRACKET
* **Geral**: ID
* **Operador aritmético**: OPERATOR
* **Número**: NUMBER
* **Atribuição**: EQUAL
* **"Se"**: IF
* **"Se não"**: ELSE
* **"Enquanto"**: WHILE
* **Impressão**: PRINT
* **Símbolo de comparação**: COMPARATOR
* **Texto**: TEXT
* **Comentário**: COMMENT
  1. **Geração de erros**

Ao ler uma cadeia que não pertence a linguagem, a seguinte mensagem de erro será lançada:

A cadeia <cadeia> nao faz parte da linguagem

Além disso, o compilador terá sua execução encerrada.

1. **Analisador sintático**
   1. **Gramática**

O analisador sintático recebe os *tokens* do analisador léxico e realiza as derivações segundo a gramática livre de contexto especificada abaixo:

P: {

**start** -> INIT **command** END;

**command** -> **declaration**

| **expression**

| **allocation**

| **loop**

| **conditional**

| **comparison**

| **text**

| **comment**

| **print**

**declaration** -> TYPE ID **continuation** **command**

| TYPE ID EQUAL **expression** **continuation** **command**

**continuation** -> COMMA ID **continuation** | Ɛ

**expression** -> **expression** OPERATOR **expression**

| NUMBER

| ID

**allocation** -> ID EQUAL **expression** **command**

**conditional** -> IF **comparison** **block** **command**

| **else\_conditional**;

**else\_conditional** -> ELSE **comparison** **block** **command**

| ELSE **block** **command**

**loop** -> WHILE **comparison** **block** **command**

**comparison** -> OPN\_PARENTH **expression** COMPARATOR **expression** CLS\_PARENTH

**text** -> TEXT

**comment** -> COMMENT

**block** -> OPN\_BRACKET **command** CLS\_BRACKET

**print** -> PRINT OPN\_PARENTH TEXT CLS\_PARENTH **command**

}

Vale ressaltar que a regra de expressões aritméticas possui um identificador %left no *token* OPERAND para indicar ao gerador de códigos que a derivação deve ser feita da esquerda para a direita.

**Nota:** os símbolos **não-terminais** estão destacados em negrito.

1. **Manual de uso**

Estando em um ambiente **Unix**, certifique-se de ter o **Flex** e o **Bison** instalado. Caso positivo, basta executar o arquivo **compile-unix.sh** para que a compilação seja executada.