UNIVERSIDADE ESTADUAL PAULISTA
"JÚLIO DE MESQUITA FILHO"

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:

1.1. Alfabeto

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

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

2. Analisador léxico

2.1. 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`
- **Decaração do laço “for”:** `for`
- **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])*`

2.2. 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

2.3. 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.

3. Analisador sintático

3.1. 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.
   | OPN_PARENTH expression CLS_PARENTH
   | NUMBER
   | ID
   allocation -> ID EQUAL expression command
   conditional -> IF comparison block command
   | else_conditional;
   else_conditional -> ELSE comparison block command
   | ELSE block command
   loop -> FOR loop_for block
   | WHILE comparison block command
   loop_for -> OPN_PARENTH allocation SEMICOLON allocation CLS_PARENTH
   | OPN_PARENTH allocation SEMICOLON expression COMPARATOR expression CLS_PARENTH
   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.

4. 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.

Temos 3 arquivos ./testes para analisarmos o funcionamento do compilador.

Para o primeiro arquivo “programa.txt” passe como parâmetro do script o número 1 ou seja, digite:

./compile-unix.sh 1

Para o segundo arquivo “erro_lex.txt” passe como parâmetro do script o número 2 ou seja, digite:

./compile-unix.sh 2

Para o terceiro arquivo “erro_sint.txt passe como parâmetro do script o número 3 ou seja, digite:

./compile-unix.sh 3

Vale ressaltar que o arquivo compile-unix.sh deve ser execútavel . Caso o arquivo esteja disponível apenas para leitura e/ou escrita (você pode ver quais modos estão habilitados para o arquivo dgitando **ls -lh**, onde será listado todos os arquivos do diretório, os modos e a permissão de cada usuário), escreva o comando **chmod 755 compile-unix.sh** e logo após poderá executar normalmente o arquivo como dito acima**.**