Caspal v0.0.1α

Instruções de compilação

Para compilar o código fonte .csp basta executar o comando
$ python3 caspalc.py <src_code_location>
Existe um programa de exemplo na pasta test/.

Note que é necessário ter python3 instalado na máquina para executar caspalc.py.

Palavras Reservadas

A linguagem Caspal é baseada na estrutura do Pascal com algumas modificaçãoes, onde as palavras reservadas são descritas de forma com que a primeira sílaba vá para o final da palavra. Por exemplo, a palavra integer em Caspal é tegerin.

São palavras reservadas do Caspal:
grampro, rav, ginbe, ned, rof, od, fi, enth, seel, ot, adre e tewri.

Tipos de Variáveis

Existem três tipos de variáveis, inteiros, booleanos e strings, e são representados como:
tegerin, leanboo e ingstr.
É possivel também construir arrays uni e multidimensionais em cima de cada tipo de variavel, adicionando '[' e ']' no final da declaração:
tegerin[], leanboo[][] e ingstr[][]...[].
O acesso aos arrays é dado por *var_arr[0]*, por exemplo.

Variaveis do tipo tegerin devem receber valores inteiros, note, não é possivel atribuir valores que começam com 0 e que sejam diferentes de 0. Por exemplo é possivel fazer *var_a = 0;* mas não *var_b = 01;*

Variaveis do tipo leanboo devem ser *Etru* e *Sefal*.

Variaveis do tipo ingstr devem estar entre aspas duplas. Por exemplo, *var_a = "Hello";*

Operações

É possível realizar operações lógicas e aritméticas entre constantes e variáveis, além de realzar atribuições.
Como operadoes aritméticos, temos:

+, -, * e /.

Como operadores lógicos, temos:

>, <, !=, si, ro, nad e ton.

A Atribuição de variáveis pode ser feita com o símbolo =.

Identificadores

É possível nomear variáveis, nomes de métodos e o nome do programa com qualquer letra do alfabeto, minúscula ou maiuscula, além do símbolo _ seguido dos mesmos caracteres, com adição de números. Ou seja, é impossível iniciar um identificador com um dígito.

Separadores

Em Caspal, os seguintes separadores são utilizados:

., :, ; e ,.

O separador ‘ponto-e-virgula’ é utilizado no final de cada linha de operação(atribuição, chamada de função, etc).

O Separador ‘dois pontos’ é utilizado na etapa de declaração de variável, tanto após o comando *var:* quanto após cada identificação de variável do tipo *nome: <tipo>*.

O Separador ‘vírgula’ é utilizado na etapa de identificação de variável, quando se deseja que mais de uma variável seja do mesmo tipo.Por exemplo:

var:
var_1, var_2, …, var_n: tegerin

Por fim, o separador ‘ponto’ é utilizado apenas para identificar o final do programa, logo após a última instrução *ned*.

Note: Nas versões iniciais do compilador, a etapa de analise léxica considera espaço e fim de linha como um separador, então é obrigatório separar cada símbolo da linguagem por um espaço.

Gramática Oficial do Caspal v.0.0.1α

<S>x → grampro <ID>; ginbe <PROGRAM> ned.|

grampro <ID>; rav ‘\n’<RAV> ginbe <PROGRAM> ned.
<RAV> →<ID_LIST>: <DECLVAR> | <ID_LIST>: <DECLVAR> ‘\n’ <RAV>
<ID_LIST> → <ID> | <ID>, <ID_LIST>
<DECLVAR> → tegerin<ARR> | leanboo<ARR> | ingstr<ARR>
<ARR> → []<ARR> | ε
<OFFSET_ARR> → [<EXP>]
<ID> → <ID_PREFIX> <OFFSET_ARR> | <ID_PREFIX>
<ID_PREFIX> → a<ID_SUFFIX> | b<ID_SUFFIX> | … | z<ID_SUFFIX> | A<ID_SUFFIX> | … |
Z<ID_SUFFIX> | _<ID_SUFFIX>
<ID_SUFFIX> → a<ID_SUFFIX> | b<ID_SUFFIX> | … | z<ID_SUFFIX> | A<ID_SUFFIX> | … |
Z<ID_SUFFIX> | _<ID_SUFFIX> | 0<ID_SUFFIX> | … | 9<ID_SUFFIX> | ε
<PROGRAM> → <ATTR> | <LOOP> | <COND> | <IO> | ε
<ATTR> → <ID><reedtfh> := <EXP>;’ \n’ <PROGRAM>
<EXP> → <EXP> <OP> <EXP> | <ID> | <CONST> | <NOT_OP><EXP>
<OP> →+ | - | * | / | > | < | si | != | nad | ro
<CONST> → <INTEGER> | <BOOLEAN> | “<STRING>”
<INTEGER> → <NUMBERS> | -<NUMBERS>
<NUMBERS> → 1<NUMBERS_SUFFIX> | … | 9<NUMBERS_SUFFIX>
<NUMBERS_SUFFIX> → 0<NUMBERS_SUFFIX> | … | 9<NUMBERS_SUFFIX> | ε
<BOOLEAN> → Sefal | Etru
<STRING> → a<STRING> | b<STRING> | … | z<STRING> | A<STRING> | … |
Z<STRING> | _<STRING> | 0<STRING> | … | 9<STRING> | <STRING> | ε
<LOOP> → rof <ATTR> ot <INTEGER> od ‘\n’ ginbe <PROGRAM> ned; ’\n’ <PROGRAM>
<COND> → fi <EXP> enth <PROGRAM><ELSE> ’\n’ <PROGRAM>
<ELSE> → seel <PROGRAM> ned; | ned;
<IO> → tewri <EXP>;’\n’ <PROGRAM> | <ID> := adre <STRING>;’\n’ <PROGRAM>
<NOT_OP> → ton

Gramática Oficial do Caspal v.0.0.2α

Changelog:

- Arrumados alguns erros na gramática.

<S> → grampro <ID_PREFIX>; ginbe <PROGRAM> ned.|
grampro <ID_PREFIX>; rav ‘\n’<RAV> ginbe <PROGRAM> ned.
<RAV> →<ID_LIST>: <DECLVAR> | <ID_LIST>: <DECLVAR> ‘\n’ <RAV>
<ID_LIST> → <ID_PREFIX> | <ID_PREFIX>, <ID_LIST>

<DECLVAR> → tegerin<ARR> | leanboo<ARR> | ingstr<ARR>
<ARR> → []<ARR> | ε
<OFFSET_ARR> → [<NUMBERS>]
<ID> → <ID_PREFIX> <OFFSET_ARR> | <ID_PREFIX>
<ID_PREFIX> → a<ID_SUFFIX> | b<ID_SUFFIX> | … | z<ID_SUFFIX> | A<ID_SUFFIX> | … |
Z<ID_SUFFIX> | _<ID_SUFFIX>
<ID_SUFFIX> → a<ID_SUFFIX> | b<ID_SUFFIX> | … | z<ID_SUFFIX> | A<ID_SUFFIX> | … |
Z<ID_SUFFIX> | _<ID_SUFFIX> | 0<ID_SUFFIX> | … | 9<ID_SUFFIX> | ε
<PROGRAM> → <ATTR>' \n' <PROGRAM> | <LOOP> | <COND> | <IO> | ε
<ATTR> → <ID> := <EXP>; | <ID> := <STR_QUOTES>;
<EXP> → <EXP> <OP> <EXP> | <ID> | <CONST> | ton <EXP>
<OP> → + | - | * | / | > | < | si | != | nad | ro
<CONST> → <INTEGER> | <BOOLEAN>
<INTEGER> → <NUMBERS> | -<NUMBERS>
<NUMBERS> → 1<NUMBERS_SUFFIX> | … | 9<NUMBERS_SUFFIX> | 0
<NUMBERS_SUFFIX> → 0<NUMBERS_SUFFIX> | … | 9<NUMBERS_SUFFIX> | ε
<BOOLEAN> → Sefal | Etru
<STR_QUOTES> → **"**<STRING>"
<STRING> → a<STRING> | b<STRING> | … | z<STRING> | A<STRING> | … |
Z<STRING> | _<STRING> | 0<STRING> | … | 9<STRING> | <STRING> | ε
<LOOP> → rof <ID> := <EXP> ot <INTEGER> od '\n' ginbe <PROGRAM> ned; '\n'
<PROGRAM>
<COND> → fi <EXP> enth <PROGRAM><ELSE> '\n' <PROGRAM>
<ELSE> → seel <PROGRAM> ned; | ned;
<IO> → tewri <EXP>;'\n' <PROGRAM> | tewri <STR_QUOTES>;'\n' <PROGRAM> | <ID> :=
adre <STR_QUOTES>;'\n' <PROGRAM>

Gramática Oficial do Caspal v.0.0.3α

Changelog:

- Removidos os não-determinismos.

<S> → grampro <ID_PREFIX>; <S'>
<S'> → ginbe <PROGRAM> ned. | rav '\n'<RAV> ginbe <PROGRAM> ned.
<RAV> → <ID_LIST>: <DECLVAR> <RAV'>
<RAV'> → '\n' <RAV> | ε
<ID_LIST> → <ID_PREFIX><ID_LIST'>
<ID_LIST'> → , <ID_LIST> | ε

<DECLVAR> → tegerin<ARR> | leanboo<ARR> | ingstr<ARR>
<ARR> → []<ARR> | ε
<OFFSET_ARR> → [<NUMBERS>]
<ID> → <ID_PREFIX> <ID'>
<ID'> → <OFFSET_ARR> | ε
<ID_PREFIX> → a<ID_SUFFIX> | b<ID_SUFFIX> | … | z<ID_SUFFIX> | A<ID_SUFFIX> | … | Z<ID_SUFFIX> | _<ID_SUFFIX>
<ID_SUFFIX> → a<ID_SUFFIX> | b<ID_SUFFIX> | … | z<ID_SUFFIX> | A<ID_SUFFIX> | … | Z<ID_SUFFIX> | _<ID_SUFFIX> | 0<ID_SUFFIX> | … | 9<ID_SUFFIX> | ε
<PROGRAM> → <ID> := <PROGRAM'> | <LOOP> | <COND> | tewri <IO'> | ε
<PROGRAM'> → <ATTR'>' \n' <PROGRAM> | adre <STR_QUOTES>;'\n' <PROGRAM>
<ATTR> → <ID> := <ATTR'>
<ATTR'> → <EXP>; | <STR_QUOTES>;
<EXP> → <EXP> <OP> <EXP> | <ID> | <CONST> | ton <EXP>
<OP> → + | - | * | / | > | < | si | != | nad | ro
<CONST> → <INTEGER> | <BOOLEAN>
<INTEGER> → <NUMBERS> | -<NUMBERS>
<NUMBERS> → 1<NUMBERS_SUFFIX> | … | 9<NUMBERS_SUFFIX> | 0
<NUMBERS_SUFFIX> → 0<NUMBERS_SUFFIX> | … | 9<NUMBERS_SUFFIX> | ε
<BOOLEAN> → Sefal | Etru
<STR_QUOTES> → **"**<STRING>"
<STRING> → a<STRING> | b<STRING> | … | z<STRING> | A<STRING> | … |
Z<STRING> | _<STRING> | 0<STRING> | … | 9<STRING> | <STRING> | ε
<LOOP> → rof <ID> := <EXP> ot <INTEGER> od '\n' ginbe <PROGRAM> ned; '\n' <PROGRAM>
<COND> → fi <EXP> enth <PROGRAM><ELSE> '\n' <PROGRAM>
<ELSE> → seel <PROGRAM> ned; | ned;
<IO> → tewri <IO'> | <ID> := adre <STR_QUOTES>;'\n' <PROGRAM>
<IO'> → <EXP>;'\n' <PROGRAM> | <STR_QUOTES>;'\n' <PROGRAM>

Gramática Oficial do Caspal v.0.0.4α

Changelog:

- Removidas as recursões à esquerda.

<S> → grampro <ID_PREFIX>; <S'>
<S'> → ginbe <PROGRAM> ned. | rav '\n'<RAV> ginbe <PROGRAM> ned.
<RAV> → <ID_LIST>: <DECLVAR> <RAV'>
<RAV'> → '\n' <RAV> | ε
<ID_LIST> → <ID_PREFIX><ID_LIST'>
<ID_LIST'> → , <ID_LIST> | ε

<DECLVAR> → tegerin<ARR> | leanboo<ARR> | ingstr<ARR>
<ARR> → []<ARR> | ε
<OFFSET_ARR> → [<NUMBERS>]
<ID> → <ID_PREFIX><ID'>
<ID'> → [<NUMBERS>] | ε
<ID_PREFIX> → a<ID_SUFFIX> | b<ID_SUFFIX> | … | z<ID_SUFFIX> | A<ID_SUFFIX> | … |
Z<ID_SUFFIX> | _<ID_SUFFIX>
<ID_SUFFIX> → a<ID_SUFFIX> | b<ID_SUFFIX> | … | z<ID_SUFFIX> | A<ID_SUFFIX> | … |
Z<ID_SUFFIX> | _<ID_SUFFIX> | 0<ID_SUFFIX> | … | 9<ID_SUFFIX> | ε
<PROGRAM> → <ID> := <PROGRAM'> | <LOOP> | <COND> | tewri <IO'> | ε
<PROGRAM'> → <ATTR'>' \n' <PROGRAM> | adre <STR_QUOTES>;'\n' <PROGRAM>
<ATTR> → <ID> := <ATTR'>
<ATTR'> → <EXP>; | <STR_QUOTES>;
<EXP> → <ID><EXP'> | <CONST><EXP'> | ton <EXP><EXP'>
<EXP'> → <OP> <EXP><EXP'> | ε
<OP> → + | - | * | / | > | < | si | != | nad | ro
<CONST> → <INTEGER> | <BOOLEAN>
<INTEGER> → <NUMBERS> | -<NUMBERS>
<NUMBERS> → 1<NUMBERS_SUFFIX> | … | 9<NUMBERS_SUFFIX> | 0
<NUMBERS_SUFFIX> → 0<NUMBERS_SUFFIX> | … | 9<NUMBERS_SUFFIX> | ε
<BOOLEAN> → Sefal | Etru
<STR_QUOTES> → **"**<STRING>"
<STRING> → a<STRING> | b<STRING> | … | z<STRING> | A<STRING> | … |
Z<STRING> | _<STRING> | 0<STRING> | … | 9<STRING> | <STRING> | ε
<LOOP> → rof <ID> := <EXP> ot <INTEGER> od '\n' ginbe <PROGRAM> ned; '\n'
<PROGRAM>
<COND> → fi <EXP> enth <PROGRAM><ELSE> '\n' <PROGRAM>
<ELSE> → seel <PROGRAM> ned; | ned;
<IO> → tewri <IO'> | <ID> := adre <STR_QUOTES>;'\n' <PROGRAM>
<IO'> → <EXP>;'\n' <PROGRAM> | **"**<STRING>";'\n' <PROGRAM>

Gramática Oficial do Caspal v.0.1α

Changelog:

- Simplificação da gramatica.

<S> → grampro ID; ginbe <PROGRAM> ned. | grampro ID; rav <VAR> ginbe <PROGRAM>
ned.
<VAR> → <LIST_VAR>: VAR_TYPE; <VAR> | &
<LIST_VAR> → ID, <LIST_VAR> | ID

<PROGRAM> → <ATTR><PROGRAM> | <LOOP><PROGRAM> | <COND><PROGRAM> | <IO><PROGRAM> | &
<ATTR> → ID := <EXP>; | ID OP_ARR := <EXP>; | ID := UNARY_OP <EXP>;
<LOOP> → rof ID := <EXP> ot CONST od <PROGRAM> ned;
<COND> → fi <EXP> enth <PROGRAM> <ELSE>
<ELSE> → seel <PROGRAM> ned; | ned;
<IO> → tewri <EXP>; | ID := adre CONST;
<EXP> → <EXP> OP <EXP> | ID | CONST

TODO

- criar OP_ARR
- criar UNARY_OP
- renomear DECLVAR para VAR_TYPE

Gramática Oficial do Caspal v.0.1.1α

Changelog:

- Modificada a produção <EXP> para ajudar na transformação em LL(1)

<S> → grampro ID; <S'>
<S'> → ginbe <PROGRAM> ned. | rav <VAR> ginbe <PROGRAM> ned.
<VAR> → <LIST_VAR>: VAR_TYPE; <VAR> | &
<LIST_VAR> → ID<LIST_VAR'>
<LIST_VAR'> → , <LIST_VAR> | &
<PROGRAM> → ID <PROGRAM'> | <LOOP><PROGRAM> | <COND><PROGRAM> | tewri <EXP>; <PROGRAM> | &
<PROGRAM'> → := <PROGRAM''> | OP_ARR := <EXP>;<PROGRAM>
<PROGRAM''> → <EXP>;<PROGRAM> | adre CONST; <PROGRAM> | UNARY_OP <EXP>;<PROGRAM>
<LOOP> → rof ID := <EXP> ot CONST od <PROGRAM> ned;
<COND> → fi <EXP> enth <PROGRAM> <ELSE>
<ELSE> → seel <PROGRAM> ned; | ned;
<EXP> → ID OP <EXP> | CONST OP <EXP> | ID | CONST

Gramática Oficial do Caspal v.0.2α

Changelog:

- Removidos os não-determinismos.

<S> → grampro ID; <S'>
<S'> → ginbe <PROGRAM> ned. | rav <VAR> ginbe <PROGRAM> ned.
<VAR> → <LIST_VAR>: VAR_TYPE; <VAR> | &
<LIST_VAR> → ID<LIST_VAR'>
<LIST_VAR'> → , <LIST_VAR> | &
<PROGRAM> → ID <PROGRAM'> | <LOOP><PROGRAM> | <COND><PROGRAM> | tewri <EXP>; <PROGRAM> | &
<PROGRAM'> → := <PROGRAM''> | OP_ARR := <EXP>;<PROGRAM>
<PROGRAM''> → <EXP>;<PROGRAM> | adre CONST; <PROGRAM> | UNARY_OP <EXP>;<PROGRAM>
<LOOP> → rof ID := <EXP> ot CONST od <PROGRAM> ned;
<COND> → fi <EXP> enth <PROGRAM> <ELSE>
<ELSE> → seel <PROGRAM> ned; | ned;
<EXP> → ID <EXP'> | CONST <EXP'>
<EXP'> → OP <EXP> | &

/*
<S> → grampro ID; <S'>
<S'> → ginbe <PROGRAM> ned. | rav <VAR> ginbe <PROGRAM> ned.
<VAR> → <LIST_VAR>: VAR_TYPE; <VAR> | &
<LIST_VAR> → ID<LIST_VAR'>
<LIST_VAR'> → , <LIST_VAR> | &
<PROGRAM> → ID <PROGRAM'> | <LOOP><PROGRAM> | <COND><PROGRAM> | tewri <EXP>; <PROGRAM> | &
<PROGRAM'> → := <PROGRAM''> | OP_ARR := <EXP>;<PROGRAM>
<PROGRAM''> → <EXP>;<PROGRAM> | adre CONST <PROGRAM>
<LOOP> → rof ID := <EXP> ot CONST od <PROGRAM> ned;
<COND> → fi <EXP> enth <PROGRAM> <ELSE>
<ELSE> → seel <PROGRAM> ned; | ned;
<EXP> → <EXP> OP <EXP> | ID | CONST | UNARY_OP <EXP>
*/

TODO

- criar OP_ARR
- criar UNARY_OP
- renomear DECLVAR para VAR_TYPE

Gramática Oficial do Caspal v.0.3α

Changelog:

- Removidas as recursões à esquerda.

<S> → grampro ID; <S'>
<S'> → ginbe <PROGRAM> ned. | rav <VAR> ginbe <PROGRAM> ned.
<VAR> → <LIST_VAR>: VAR_TYPE; <VAR> | &
<LIST_VAR> → ID<LIST_VAR'>
<LIST_VAR'> → , <LIST_VAR> | &
<PROGRAM> → ID <PROGRAM'> | <LOOP><PROGRAM> | <COND><PROGRAM> | tewri <EXP>; <PROGRAM> | &
<PROGRAM'> → := <PROGRAM''> | OP_ARR := <EXP>;<PROGRAM>
<PROGRAM''> → <EXP>;<PROGRAM> | adre CONST; <PROGRAM> | UNARY_OP <EXP>;<PROGRAM>
<LOOP> → rof ID := <EXP> ot CONST od <PROGRAM> ned;
<COND> → fi <EXP> enth <PROGRAM> <ELSE>
<ELSE> → seel <PROGRAM> ned; | ned;
<EXP> → ID <EXP'> | CONST <EXP'>
<EXP'> → OP <EXP> | &

TODO

- criar OP_ARR
- criar UNARY_OP
- renomear DECLVAR para VAR_TYPE
- renomear = para :=

First e Follows da Gramática

first(grampro) = {grampro}
first(ID) = {ID}
first(;) = {;}
first(ginbe) = {ginbe}
first(ned) = {ned}
first(.) = {.}
first(rav) = {rav}
first(:) = {:}
first(VAR_TYPE) = {VAR_TYPE}
first(,) = {,}

first(tewri) = {tewri}
first(:=) = {:=}
first(OP_ARR) = {OP_ARR}
first(adre) = {adre}
first(rof) = {rof}
first(ot) = {ot}
first(CONST) = {CONST}
first(od) = {od}
first(fi) = {fi}
first(enth) = {enth}
fist(seel) = {seel}
first(UNARY_OP) = {UNARY_OP}
first(OP) = {OP}

first(<S>) = {grampro}
first(<S'>) = {ginbe, rav}
first(<VAR>) = {ID, &}
first(<LIST_VAR>) = {ID}
first(<LIST_VAR'>) = {,, &}
first(<PROGRAM>) = {ID, rof, fi, tewri, &}
first(<PROGRAM'>) = {:=, OP_ARR}
first(<PROGRAM''>) = {ID, CONST, adre, UNARY_OP}
first(<LOOP>) = {rof}
first(<COND>) = {fi}
first(<ELSE>) = {seel, ned}
first(<EXP>) = {ID, CONST}
first(<EXP'>) = {OP, &}

FOL(<S>) = {$}
FOL(<S'>) = {$}
FOL(<VAR>) = {ginbe}
FOL(<LIST_VAR>) = {:}
FOL(<LIST_VAR'>) = {:}
FOL(<PROGRAM>) = {ned, seel}
FOL(<PROGRAM'>) = {ned, seel}
FOL(<PROGRAM''>) = {ned, seel}
FOL(<LOOP>) = {ID, rof, fi, tewri, ned, seel}
FOL(<COND>) = {ID, rof, fi, tewri, ned, seel}
FOL(<ELSE>) = {ID, rof, fi, tewri, ned, seel}
FOL(<EXP>) = {;, ot, enth}
FOL(<EXP'>) = {;, ot, enth}

Tabela de análise

1: **<S>** → grampro ID; <S'>
2: **<S'>** → ginbe <PROGRAM> ned.
3: **<S'>** → rav <VAR> ginbe <PROGRAM> ned.
4: **<VAR>** → <LIST_VAR>: VAR_TYPE; <VAR>
5: **<VAR>** → &
6: **<LIST_VAR>** → ID<LIST_VAR'>
7: **<LIST_VAR'>** → , <LIST_VAR>
8: **<LIST_VAR'>** → &
9: **<PROGRAM>** → ID <PROGRAM'>
10: **<PROGRAM>** → <LOOP><PROGRAM>
11: **<PROGRAM>** → <COND><PROGRAM>
12: **<PROGRAM>** → tewri <EXP>; <PROGRAM>
13: **<PROGRAM>** → &
14: **<PROGRAM'>** → := <PROGRAM''>
15: **<PROGRAM'>** → OP_ARR := <EXP>;<PROGRAM>
16: **<PROGRAM''>** → <EXP>;<PROGRAM>
17: **<PROGRAM''>** → adre CONST; <PROGRAM>
18: **<PROGRAM''>** → UNARY_OP <EXP>;<PROGRAM>
19: **<LOOP>** → rof ID := <EXP> ot CONST od ginbe <PROGRAM> ned;
20: **<COND>** → fi <EXP> enth <PROGRAM> <ELSE>
21: **<ELSE>** → seel <PROGRAM> ned;
22: **<ELSE>** → ned;
23: **<EXP>** → ID<EXP'>
24: **<EXP>** → CONST<EXP'>
25: **<EXP'>** → OP <EXP>
26: **<EXP'>** → &