Anansi

Guilherme Sganderla

Milena Lucas Dos Santos

Natã Rafael Cruz Jesus

**Descrição da linguagem**

Anansi é uma linguagem de programação estrutural baseada como um subconjunto da linguagem C, herdando assim algumas de suas características léxicas, sintáticas e semânticas.

**Expressões Regulares:**

Letra:

letra = [abc...zABC...Z]

Números:

num = [0 | 1 | 2 | …| 9]

Inteiro:

int = num(num)\*

Decimal:

float = num(num\*) , num(num)\*

Identificador:

id = letra(letra|num|\_

Incrementar:

incr = ++

Decrementar:

decr = --

Incrementar ou Decrementar:

inc\_dec = letra(incr | decr)

Tipo do identificador:

tipo: int | float

Biblio:

biblio = letra(letra)\*.h

Bibliotecas:

bibliotecas = ($include<biblio>)\*

Definir:

definir = id num(num)\*

Struct:

Struct = struct id { (criar\_var)\*}

Programa:

programa=(bibliotecas)\*(define definir)\*(struct)\*(procedimento)\*int main(){(chamadas)\*}

Parâmetro:

param = (tipo(g)<IDENTIFICADOR>)\*

Atribuidor:

atribuidor = ‘=’

Alocação Dinâmica:

aloca\_din = (tipo) malloc(sizeof(tipo))

Atribuição:

atribuicao = id atribuidor ((num|id) | aloca\_din | calc\_raiz | calc\_pot | fmod).

Operação de atribuição:

op\_atribuicao: "-=" | "+="

Atribuição resumida:

atr\_res = do id op\_atribuicao (num|id).

Atributo:

atributo = inteiro | decimal

Expressão Aritmetica:

exp\_aritmetica = ((atributo|id) op\_aritmetico atributo)

Expressão:

exp = ( (exp\_aritmetica| atributo) op\_artimetico (exps\_aritmetica | atributo)).

Condição Relacional:

condicao\_relaciona = id op\_relacional atributo

While:

while= while(condicao\_relacional) { (chamadas)\* (inc\_dec).}

For:

for= for atr\_for ; condicao\_relacional ; inc\_dec) cont\_for

Contador for:

cont\_for = { (chamadas)\* }

Atribuição for:

atr\_for = id atribuidor (int | id)

Chamadas:

chamadas = criar\_var

| atribuicao

| atr\_res

| exp

| if ( condicao ) { (chamadas)\* } (else)

| while

| for

| ler\_string

| copia\_string

| tam\_string

Procedimento:

procedimento = void id (param) { chamadas\* }

Criação de variáveis:

criar\_var = tipo id p

Atribuição:

atribuicao = id atr (num+ | inteiro) p

Operações:

op\_aritmetico = + | - | \* | /

Operadores relacionais:

op\_relacional = (> | < | >= | <= | != | ==)

Operadores lógicos:

op\_logicos = || | &&

Condição:

condicao = id operadores (id | num+)

If:

if = if(condicao) { (chamadas)+ } | if( condicao) chamadas

Else:

else = else { chamadas\* }

Ler string:

ler\_string = gets (id)

Cópia string:

copia\_string = strcpy(id,id)

Tamanho da string:

tam\_string = strlen(id)

Resto da divisão:

fmod = fmod( (inteiro | id) , (inteiro,id) )

Cálculo de raiz:

calc\_raiz = sqrt ( (id|inteiro) )

Cálculo de potência:

calc\_pot = pow( (inteiro|id), (inteiro,id) )

Gramática

A gramática inicia-se pelo símbolo não-terminal “programa” que gera uma sequência de não-terminais que executam a linguagem:

PROGRAMA -> BIBLIOTECAS\* $define DEFINIR\* CMD\_STRUCT\* PROCEDIMENTOS\* int main(){CMD\_CHAMADAS\*} | BIBLIOTECAS CMD\_STRUCT\* PROCEDIMENTO\* int main(){CHAMADAS\*}

O não-terminal “bibliotecas” gera o include e o não-terminal “biblio”:

BIBLIOTECAS -> $include BIBLIO

Esse não-terminal gera uma palavra que refere-se a uma biblioteca da linguagem:

BIBLIO -> LETRALETRA+.h

O não-terminal “definir” gera uma identificador que define o número desse identificador

DEFINIR -> ID NUMNUM\*

O "cmd\_struct" gera uma struct onde é declarado as variáveis que podem ser do tipo inteiro ou float

CMD\_STRUCT -> struct ID { CRIAR\_VAR\*};

Esse não-terminal gera todas as possíveis tarefas que a linguagem pode fazer como criar variáveis, atribuição, ler uma string etc.

CMD\_CHAMADAS -> CRIAR\_VAR\* | ATRIBUICAO\* | ATR\_RES\* | EXP\* | CMD\_IF\* | CMD\_ELSE\* | CMD\_WHILE\* | CMD\_FOR\* | LER\_STRING\* | COPIA\_STRING\* | TAM\_STRING\*

O não-terminal "procedimento" gera uma função do tipo void

PROCEDIMENTO -> void ID (PARAM\*) { CHAMADAS\* }

Este não-terminal gera um parâmetro para a função void

PARAM -> TIPO ID

Cria variáveis do tipo inteiro ou float

CRIAR\_VAR-> TIPO VAR-LISTA

TIPO -> int | float

VAR-LISTA -> id, VAR-LISTA | id

O não-terminal "atribuicao" gera todas as regras possíveis de uma atribuição na linguagem

ATRIBUICAO -> ID = NUM. | ID = ID. | ID = ALOC\_DIN. | ID = CALC\_RAIZ. | ID = CMD\_FMOD. | ID = CALC\_POT.

Esse não-terminal gera uma operação juntamente com atribuição, por exemplo "s += soma."

ATR\_RES -> do ID OP\_ATRIBUICAO NUM. | do ID OP\_ATRIBUICAO ID.

O não-terminal "exp" gera todas as regras possíveis de uma operação na linguagem

EXP -> EXP\_ARITMETICA OP\_ARTIMETICO EXP\_ARITMETICA. | EXP\_ARITMETICA OP\_ARTIMETICO ATRIBUTO. | ATRIBUTO OP\_ARTIMETICO EXP\_ARITMETICA. | ATRIBUTO EXP\_ARITMETICA ATRIBUTO.

Esse não-terminal gera um número inteiro ou um decimal

ATRIBUTO -> INTEIRO | DECIMAL

Esse não-terminal gera regras de operações aritméticas

EXP\_ARITMETICA -> ATRIBUTO OP\_ARITMETICO ATRIBUTO | ID OP\_ARTIMETICO ATRIBUTO | ATRIBUTO OP\_ARITMETICO ID | ID OP\_ARITMETICO ID

Gera terminais relacionados a operações aritméticas

OP\_ARITMETICO -> + | - | \* | /

Esse não-terminal é utilizado nos não-terminais "cmd\_if" e "cmd\_while" para fazer a comparação

CONDICAO -> ID OP\_RELACIONAL ATRIBUTO | ID OP\_RELACIONAL ID

Gera terminais utilizados nos não-terminais "cmd\_if" e "cmd\_while"

OP\_LOGICO -> || | &&

Gera terminais de comparação

OP\_RELACIONAL -> == | != | > | < | >= | <= |

O não-terminal "cmd\_if" gera todas as possíveis regras para o terminal "if"

CMD\_IF -> if(CONDICAO) { CHAMADAS+ } | if(CONDICAO) CHAMADAS | if(CONDICAO OP\_LOGICO CONDICAO) { CHAMADAS+ } | if(CONDICAO OP\_LOGICO CONDICAO)

CHAMADAS

O não-terminal "cmd\_else" é utilizado após o comando "if"

CMD\_ELSE -> else { CHAMADAS\* }

Esse não-terminal gera todas as possíveis regras para esta linguagem

CMD\_WHILE -> while(CONDICAO) { (CHAMADAS)\* INC\_DEC.} | while(CONDICAO OP\_LOGICO CONDICAO) { (CHAMADAS)\* (INC\_DEC).}

O não-terminal "inc\_dec" gera um identificador juntamente com o terminal de incrementar ou decrementar

INC\_DEC -> ID++ | ID--

Gera o comando "for"

CMD\_FOR -> for(ATR\_FOR;CONDICAO;INC\_DEC)CONT\_FOR

O não-terminal "cont\_for" chama o não-terminal "chamadas" onde estão todos os possíveis comandos na linguagem

CONT\_FOR -> { CHAMADAS\*}

O não-terminal "atr\_for" gera uma variável que recebe um valor para iniciar o for

ATR\_FOR -> ID ATRIBUIDOR INTEIRO | ID ATRIBUIDOR ID

Gera uma função que lê uma string

LER\_STRING -> gets (ID).

Gera uma função que copia uma string para outra

COPIA\_STRING -> strcpy(ID,ID).

Gera uma função que calcula o tamanho de uma string

TAM\_STRING -> strlen(ID).

Gera não-terminais do tipo inteiro e do tipo ponto-flutuante

TIPO -> int | float

Gera terminais com atribuição e soma ao mesmo tempo

OP\_ATRIBUICAO -> -= | +=

Gera números naturais

INTEIRO -> INTEIRO DIGITO | DIGITO

Gera números naturais ou decimais

DECIMAL -> INTEIRO,INTEIRO

Esse não-terminal gera uma sequência de tokens que referem-se a alocação dinâmica

ALOC\_DIN -> (TIPO)malloc(sizeof(TIPO)).

Gera uma função que calcula a raiz do identificador ou número natural

CALC\_RAIZ -> SQRT (ID) | SQRT (INTEIRO).

Gera uma função que calcula o resto da divisão

CMD\_FMOD -> fmod(INTEIRO, ID). | fmod(INTEIRO, inteiro). | fmod(ID INTEIRO). | fmod(ID, ID).

Gera uma função que calcule a potência de duas variáveis ou números

CALC\_POT -> pow(INTEIRO, INTEIRO). | pow(INTEIRO, ID). | pow(ID, ID). | pow(ID, INTEIRO).

ID -> LETRALETRA\*NUM\*\_\* | LETRANUM\*LETRA\*\_\* | LETRA\_\*LETRA\*NUM\* | LETRALETRA\*\_\*NUM\* | LETRANUM\*\_\*LETRA\* | LETRA\_\*NUM\*LETRA\*

LETRA -> a| b | c | … | z | A | B | C | … | Z

NUM -> 0 | 1 | 2| … | 9

**Gramática de atributos**

Nesta gramática, o atributo valor gera números naturais ou decimais, onde val é sintetizado.

|  |  |
| --- | --- |
| Regras Gramaticais | Regras semânticas |
| ATRIBUTO -> INTEIRO | atributo.val = inteiro.val |
| ATRIBUTO -> DECIMAL | atributo.val = decimal.val |
| INTEIRO -> INTEIRO DIGITO | inteiro1.val = num.val\*10 + digito.val |
| INTEIRO -> DIGITO | num.val = digito.val |
| FRACIONARIO - > FRACIONARIO DIGITO | fracionaria.val = num.val\*10 + digito.val  cont.val = cont.val + cont.val |
| FRACIONARIO -> DIGITO | num.val = digito.val  cont.val = digito.val |
| DECIMAL -> INTEIRO,FRACIONARIO | decimal.val = inteiro.val + num.val  num.val = fracionaria.val / num.cont  num.cont =  for(i=0; i< cont.val; i++)  num.cont \*= 10; |
| DIGITO -> 0 | 1 | 2| … | 9 | digito.val = 0 | 1 | 2 | ... | 9 |

Nessa gramática, o atributo tipo\_dados gera o tipo de dado sendo inteiro ou real sendo em sua maior parte herdado

|  |  |
| --- | --- |
| Regras Gramaticais | Regras semânticas |
| CRIAR\_VAR-> TIPO VAR-LISTA | criarvar.tipo\_dado = tipo.tipo\_dado |
| TIPO -> int | tipo.tipo\_dado = int.tipo\_dado |
| TIPO -> float | tipo.tipo\_dado = float.tipo\_dado |
| VAR-LISTA -> id, VAR-LISTA | id.tipo\_dado = var-lista.tipo\_dado  var-lista.tipo\_dado = var-lista.tipo\_dado |
| VAR-LISTA -> id | id.tipo\_dado = var-lista.tipo\_dado |

A gramática a seguir está o atributo valor onde calcula a incrementação ou decrementação de um certo identificador

|  |  |
| --- | --- |
| Regras Gramaticais | Regras semânticas |
| INC\_DEC -> ID++ | inc.val = id.val = id.val + 1 |
| INC\_DEC -> ID-- | dec.val = id.val = id.val -1 |

**Atributos herdados ou sintetizados**

|  |  |
| --- | --- |
| Atributo herdado | Atributo sintetizado |
| criarvar.tipo\_dado | inc.val |
| tipo.tipo\_dado | dec.val |
| id.tipo\_dado | atributo.val |
| var-lista.tipo\_dado | inteiro.val |
|  | decimal.val |
|  | num.val |
|  | digito.val |
|  | cont.val |
|  | fracionaria.val |
|  | num.cont |
|  | digito.val |
|  | id.val |

Funcionamento do código e tratamento de erros

O Anansi, com o auxílio da ferramenta JAVACC lê o arquivo fonte selecionado pelo usuário e faz as análises léxicas e sintáticas, informando o usuário se foi possível ou não compilar o programa fonte. Mostrando também a quantidade de erros léxicos e/ou sintáticos

Na análise sintática e léxica o software percorre um caractere por vez, identificando tokens que podem ou não ser erros sintáticos. Ao terminar de percorrer as cadeias é retornado o numero de erros encontrados durante essa varredura. A ferramenta JAVACC auxilia nessa analise por meio de opções como o lookahead e alguns tratadores de exceções como o ParseException

No tratamento da análise semântica foi criada uma tabela de símbolos para auxilio na identificação de tokens, Keywords e categorias de cadeias, fazendo por exemplo comparações entre variáveis para verificação de tipo, entre outras funções auxiliares

No tratamento de erros léxicos, que são conjuntos de símbolos que não pertencem a linguagem, e sintáticos, que são erros estruturais, como a falta de um ; no final do comando ou parênteses não fechados, foi utilizado o método de ressincronização (o recovery), que quando detecta um erro tenta ressincronizar sua entrada com o não-terminal que se espera reconhecer, ele continua consumindo os tokens seguintes ate encontrar um que pertença a uma regra ou conjunto seguinte ao de onde o token errado foi encontrado.

//na chamada da primeira função

pos = 4

[ funcChar, id, proc, char, None, Global]

[ a, id, par, int, 0, Local]

[ b, id, par, int, 0, Local]

[ res, id, var, char, 0, Local]

pos = 1

[ funcChar, id, proc, char, None, Global]

pos = 7

[ funcChar, id, proc, char, None, Global]

[ main, keyword, proc, int, None, Global]

[ a, id, var, int, 0, Local]

[ b, id, var, int, 0, Local]

[ c, id, var, int, 122, Local]

[ d, id, var, int, 140, Local]

[ i, id, var, int, 1, Local]

pos = 2

[ funcChar, id, proc, char, None, Global]

[ main, keyword, proc, int, None, Global]

//fim

//os pos indicam quando ele começa outra função

**Arvore sintática**

fonte1.txtPROGRAMA

fonte1.txt FUNCT\_LIST

fonte1.txt FUNCT\_DECL

fonte1.txt FUNCT\_TYPE

fonte1.txt PARAM\_LIST

fonte1.txt FUNCT\_TYPE

fonte1.txt FUNCT\_TYPE

fonte1.txt FUNCT\_BODY

fonte1.txt VAR\_LIST

fonte1.txt IF //sintetizado

fonte1.txt EXPRESSAO //sintetizado

fonte1.txt NUMERO\_EXPRESSAO //sintetizado

fonte1.txt TERM //sintetizado

fonte1.txt UNARY\_EXPR //sintetizado

fonte1.txt FACTOR //sintetizado

fonte1.txt NUMERO\_EXPRESSAO //sintetizado

fonte1.txt TERM //sintetizado

fonte1.txt UNARY\_EXPR //sintetizado unary\_expr.val = fator.val

fonte1.txt FACTOR //sintetizado

fonte1.txt FUNCT\_BODY

fonte1.txt RETURN //sintetizado

fonte1.txt EXPRESSAO //sintatizado

fonte1.txt NUMERO\_EXPRESSAO //sintetizado

fonte1.txt TERM sintetizado

fonte1.txt UNARY\_EXPR //sintetizado

fonte1.txt FACTOR //sintizado

fonte1.txt FUNCT\_BODY

fonte1.txt RETURN

fonte1.txt EXPRESSAO

fonte1.txt NUMERO\_EXPRESSAO

fonte1.txt TERM

fonte1.txt UNARY\_EXPR

fonte1.txt FACTOR

fonte1.txt FUNCT\_LIST

fonte1.txt FUNCT\_DECL

fonte1.txt FUNCT\_TYPE

fonte1.txt PARAM\_LIST

fonte1.txt FUNCT\_BODY

fonte1.txt VAR\_LIST

fonte1.txt VAR\_LIST

fonte1.txt WHILE

fonte1.txt EXPRESSAO

fonte1.txt NUMERO\_EXPRESSAO

fonte1.txt TERM

fonte1.txt UNARY\_EXPR

fonte1.txt FACTOR

fonte1.txt NUMERO\_EXPRESSAO

fonte1.txt TERM

fonte1.txt UNARY\_EXPR

fonte1.txt FACTOR

fonte1.txt FUNCT\_BODY

fonte1.txt VAR\_LIST

fonte1.txt VAR\_LIST

fonte1.txt ATRIBUICAO

fonte1.txt EXPRESSAO

fonte1.txt NUMERO\_EXPRESSAO

fonte1.txt TERM

fonte1.txt UNARY\_EXPR

fonte1.txt FACTOR

fonte1.txt ATRIBUICAO

fonte1.txt EXPRESSAO

fonte1.txt NUMERO\_EXPRESSAO

fonte1.txt TERM

fonte1.txt UNARY\_EXPR

fonte1.txt FACTOR

fonte1.txt TERM

fonte1.txt UNARY\_EXPR

fonte1.txt FACTOR

fonte1.txt VAR\_LIST

fonte1.txt FOR

fonte1.txt ATRIBUICAO

fonte1.txt EXPRESSAO

fonte1.txt NUMERO\_EXPRESSAO

fonte1.txt TERM

fonte1.txt UNARY\_EXPR

fonte1.txt FACTOR

fonte1.txt EXPRESSAO

fonte1.txt NUMERO\_EXPRESSAO

fonte1.txt TERM

fonte1.txt UNARY\_EXPR

fonte1.txt FACTOR

fonte1.txt NUMERO\_EXPRESSAO

fonte1.txt TERM

fonte1.txt UNARY\_EXPR

fonte1.txt FACTOR

fonte1.txt ATRIBUICAO

fonte1.txt EXPRESSAO

fonte1.txt NUMERO\_EXPRESSAO

fonte1.txt TERM

fonte1.txt UNARY\_EXPR

fonte1.txt FACTOR

fonte1.txt PROGRAMA