Universidade Federal de Alagoas - UFAL
Instituto de computação - IC
Disciplina: Compiladores
Prof: Drº Alcino Dall Igna Junior

Especificação da Linguagem ENL

Alunos: Thiago Emmanuel G. Rodrigues
Erivaldo Lourenço Mariano

Maceió, 05 de Março de 2018

ENL - E Not a Language

Especificação mínima da linguagem

Na linguagem ENL a função *`begin()`* será o ponto de entrada do programa. Se a existir um programa na linguagem ENL ela tem que ter a função *begin().* Ex de um programa em ENL:

```
void begin()
{
}
```

Todo comando em ENL tem por obrigação terminar com ‘;’.

A declaração de variável é feita antes da declaração da função *begin()*, como segue:

```
variable{
     int a, b, … , z;
}
void bagin()
{
     a = 1;
}
```

Variáveis do mesmo tipo podem ser declaradas na mesma linha. Toda declaração de variável termina com ‘;’.

Tipos de dados

A Linguagem ENL é estaticamente tipada e é necessário a declaração explícita do tipo das variáveis em sua declaração. As palavras que definem seu tipo são reservadas.

Inteiro

Palavra **int** define um tipo inteiro. A declaração de uma variável do tipo inteiro é feita da seguinte forma:

int variavel;

Variáveis do tipo inteiro podem fazer as seguintes operações: soma, subtração, multiplicação, divisão e resto.

Ponto flutuante

Palavra **ptf** define um tipo ponto flutuante. A declaração de uma variável de ponto flutuante é da seguinte forma:

ptf variavel;

Variáveis do tipo ponto flutuante podem fazer as seguintes operações: soma, subtração, multiplicação e divisão.

Caractere

A palavra **char** define um tipo caractere que está representado no padrão ASCII puro. A declaração é da seguinte forma:

char caractere;

Cadeia de caractere

A palavra **cchar** define uma cadeia de caracteres. Uma cadeira de caractere é uma sequência de caracteres. A declaração é da seguinte forma:

cchar cadeia[tamanho];

Todos os elementos do cchar sao do tipo char. As operações que podem ser feita com variáveis do tipo cchar sao: concatenação.

Arranjo unidimensional

vector - Define um arranjo de uma dimensão. Esse arranjo é uma coleção de objetos que contém obrigatoriamente, os tipos primitivos, e são declarados da seguinte forma:

vect tipo variavel[tamanho];

vect int exemplo[10];

Operadores

Dividimos os operadores em vários tipos, verificados na tabela abaixo:

Nome do operador	Símbolo
Operadores aritméticos	
Soma Subtração Multiplicação Divisão Módulo	+ - * / %

A associatividade para operadores aritméticos será da esquerda para direita. A ordem de precedência segue abaixo:

- Multiplicação, Divisão, Módulo
- Adição e Subtração

Nome do operador	Símbolo
Operadores de comparação	
Igual Diferente Menor igual Maior igual	== != <= >=

Menor que Maior que	< >

A associatividade para operadores de comparação será da esquerda para direita. A ordem de precedência segue abaixo:

- **< , > , <= , >=**
- **== !=**

Nome do operador	Símbolo
Operadores lógico	
AND OR NOT	&& || !

A associatividade para operadores lógicos será da esquerda para direita. A ordem de precedência segue abaixo:

- !
- && ||

Atribuição

A atribuição é feita usando o operador '**=**', como mostra na tabela de operadores. Funciona de modo tal que o valor da esquerda do operador = , recebe o valor da direita.

EX:

```
a = b+1;
```

Neste caso a variável recebe o resultado da expressão b+1. A atribuição é associativa a direita, de modo que no caso **a=b=c**, feito primeiro a operação **b=c** e o resultado é atribuído a **a**.

Instruções

Estrutura condicional de uma e duas vias:

```
if  ( expressão lógica ){
        instruções
}
else{
}
```

Estrutura iterativa com controle lógico:

```
while ( expressão lógica ){
        instruções
}
```

Estrutura interativa controlada por contador:

```
for ( inicio, fim, ritmo) {
        instruções
}
```

Funções

As funções são declaradas com a palavra reservada *`function`* *, e a palavra reservada* *`return`* *indica o retorno da função.*

Exemplo de declaração de função:

```
function tipo_de_retorno identificador( tipo parâmetro1, tipo parâmetro2, … , tipo parâmetroN ){

     return tipo_retorno;
}
```

Exemplos

Alo mundo:

```
int begin()
{
	put('Ola mundo');

}
```

Fibonacci

```
variable
{
	int num, sum, cont, a, b;
}

function void fibonacci(int n){
	num = n;
	sum = 0;
	a = 1;
	b = 1;
	put(sum);
	for(i = 1; i < num; i = i+1)
	{
		b = a;
		a = sum;
		sum = a + b;
		put(sum);
	}
}

int begin()
{
	fibonacci(5);
}
```

Shell Sort

```
function void shellSort(vect vet, int tam){
     variable {
          int i, j, valor;
          int gap = 1;
     }
     while(gap<tam){
          gap = 3*gap+1;
     }
     while (gap>1){
          gap = gap/3;
          for(i=gap; i<tam; i=i+1){
               valor = vet[i];
               j = i - gap;
               while(j >= 0 && valor < vet[j]){
                    vet[j+gap] = vet[j];
                    j = j - gap;
               }
          vet[j + gap] = valor;
          }
     }
}


int begin()
{
     variable{
          int vetor[50], tam = 50;
     }
     shellsort(vetor, tam);
}
```

Especificação dos tokens

A linguagem de Programação que será utilizada para a implementação dos analisadores léxicos e sintáticos será o **Python**.

- Especificação dos Tokens por Categorias

Os tokens foram divididos em 6 categorias conforme abaixo:

tokens simples

begin
variable
function
return
void
;
(
)
{
}
,
"
'

atrib

=

oprr

<
>
==
!=
>=
<=

opra

+
-

oprm

*
/
%

oprl

&&

||
!

Tokens com argumentos

int
char
ptf
bool
id
cteN
if
else
while
for
string
let
dig
charEsp

Estrutura Geral do Programa

As palavras com iniciais **maiusculas** são NÃO-TERMINAIS e as palavras com iniciais **minúsculas** são TERMINAIS.

INICIO => VAR* | FUNC* | BEGIN <EOF>
VAR => 'variable' '{' TIPO IDENT ('=' Ea | Ta | Fa | El)? (‘,’ IDENT)*}';’
FUNC => 'function' TIPO IDEMT '(' PARAM ')' '{' CORPO ' 'return' TIPO}'
BEGIN => 'void' 'begin' '(' ')' '{' CORPO '}'
IDENT => 'let' ('let' | 'dig' | 'charEsp')*
TIPO => 'int' | 'char' | 'ptf' | CCHAR | VECTOR | 'bool'
PARAM => (TIPO IDENT (',' TIPO IDENT)*)?
CORPO => VAR* (Ea | Ta | Fa | El)* EstCond* EstItr*
Ea => TaEar
Ear => oprlTaEar
Ear => & (palavra vazia)
Ta => FaTar
Tar => oprmFaTar
Tar => &
El => TlElr

Elr => oprlTlElr | OprrElElr
Elr => &
Fa => 'id' | 'cteN' | '"Ea"'
EstCond => 'if' '(' (El)+ ')' '{' CORPO ('else' '{' CORPO '}')* '}'
EstItr => 'while' '(' (El)+ ')' '{' CORPO '}' | 'for' '(' VAR '=' 'cteInt' ';' VAR 'oprr' 'cteInt' ';' 'Ea' | (VAR '+' '1') ')' '{' CORPO '}'
CCHAR => "" 'cchar' "" ';'
VECTOR => 'vect' TIPO '[' cteInt ']' ';'
CHAMADA_FUNC => IDENT '(' PARAM_CHAMADA ')' ';'
PARAM_CHAMADA => (IDENT (',' IDENT)*)?

Expressões Regulares

let = ['a'-'z' | 'A'-'Z']
char = ['a'-'z' | '0'-'9']
charEsp = ["" | '!' | '@' | '#' | '$' | '%' | '&' | '*' | '(' | ')' | '_' | '-' | '{' | '}' | '[' | ']']
dig = ['0'-'9']
digI = ['1'-'9']
digO = ['0'-'7']
digH = ['0'-'9' ('a'-'f' | 'A'-'F')
id = ('let' | 'charEsp')('let' | 'dig' | 'charEsp')*
cteInt = (digI dig* | '0' digO* | '0' ('z' | 'x') digH)
ctePtf = (((dig* '.' dig+ | dig+ '.')(('e' | 'E')('+' | '-')?dig+) | dig+ ('e' | 'E')('+' | '-')? dig+)('f' | 'l' | 'F' | 'L')?
bool = ((dig (oprr | oprl) dig) | (VAR (oprr | oprl) VAR) | (dig (oprr | oprl) VAR) | (VAR (oprr | oprl) dig))
cchar = ('let' ('dig' | 'let' | 'charEsp')* | 'dig' ('dig' | 'let' | 'charEsp')* | 'charEsp' ('dig' | 'let' | charEsp)*)

Terminais

atr = '='
id = ("let" | "charEsp") ("let" | "dig" | "charEsp")*
oprr = '==' | '!=' | '>' | '<' | '>=' | '<='
opra = '+' | '-'
oprm = '*' | '/' | '%'
oprl = '&&' | '||' | '!'
cteN = cteInt | cteFlt