



Universidade Federal de Alagoas - UFAL

Instituto de computação - IC

Disciplina: Compiladores

Professor Alcino Dall Igna Junior

# Especificação da Linguagem ENL

## Compiladores 2019.2

Thiago Emmanuel G. Rodrigues

Erivaldo Lourenço Mariano

<b>Introdução</b>	<b>3</b>
<b>Estrutura geral do programa</b>	<b>3</b>
<b>Identificador</b>	<b>3</b>
Tipos de dados	4
Inteiro	4
Ponto flutuante	5
Cadeia de caractere	6
Booleano	6
Arranjo unidimensional	7
<b>Coerção</b>	<b>7</b>
<b>Casting</b>	<b>7</b>
Atribuição	8
<b>Concatenação</b>	<b>8</b>
<b>Comentários</b>	<b>8</b>
<b>Funções</b>	<b>8</b>
<b>Inicialização</b>	<b>9</b>
<b>Operadores</b>	<b>9</b>
Operadores aritméticos	9
Operadores de comparação	10
Operadores lógico	10
Precedência e associatividade	10
Instruções	11
<b>Impressão</b>	<b>12</b>
<b>Leitura</b>	<b>12</b>
<b>Programas:</b>	<b>13</b>

Ola mundo:	13
Fibonacci	13
Shell Sort	13

## Introdução

A linguagem ENL foi baseada na linguagem C, a linguagem é de tipagem estática, possui coerção, case sensitive, desenvolvida com a ideia de ser fácil e rápida.

## Estrutura geral do programa

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

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

- Todo comando em ENL tem por obrigação terminar com ‘;’.
- As variáveis podem ser declaradas dentro da função *begin*, de outras funções, dentro das estruturas de seleção e de interação.
- A linguagem ENL só admite escopo local.
- As funções na linguagem ENL só podem ser escritas antes da função *begin*, e se iniciam com a palavra reservada ***function***

## Identificador

A linguagem ENL é case sensitive e possui algumas regras:

- Iniciam-se com uma letra
- os caracteres podem ser letras, números.

- O tamanho é de até 32 caracteres.
- Não pode ter espaços em branco entre as palavras.
- Não podem ser palavras reservadas.

## Palavras reservadas

'if' , 'else', 'while', 'for', 'in', 'to', 'step', 'do', 'begin', 'function',  
 'return', 'void', 'int', 'float', 'char', 'cchar', 'vector', 'bool', 'char', 'cchar',  
 'vector', 'bool', 'true', 'false', 'break'.

## 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;

Operações suportadas:

Tipo	Operadores aritméticos	Operadores de comparação	Operadores lógicos	Concatenação
int	Soma Subtração Multiplicação Divisão Módulo	Igual Diferente Menor igual Maior igual Menor que		Se concatenados a um caractere ou cadeia de caracteres

	unário negativo	Maior que		(formam um cchar)
--	-----------------	-----------	--	-------------------

O tipo int suporta constantes literais para números inteiros entre -2147483648 e 2147483647

### Ponto flutuante

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

**float** variavel;

Tipo	Operadores aritméticos	Operadores de comparação	Operadores lógicos	Concatenação
float	Soma Subtração Multiplicação Divisão Módulo unário negativo	Igual Diferente Menor igual Maior igual Menor que Maior que		Se concatenados a um caractere ou cadeia de caracteres (Formam um cchar)

O tipo float suporta constantes literais, para reais entre (aproximadamente)  $10^{-38}$  e  $10^{38}$ .

### Caractere

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

**char** caractere;

Tipo	Operadores aritméticos	Operadores de comparação	Operadores lógicos	Concatenação
char		Igual		Podem ser

		Diferente Menor igual Maior igual Menor que Maior que		concatenados aos tipos int, float, char, cchar, bool(Formam um cchar)
--	--	---	--	---

O tipo char não suporta constantes literais.

### Cadeia de caractere

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

**cchar** cadeia[número inteiro];

Todos os elementos do cchar são do tipo char. As operações que podem ser feitas com variáveis do tipo cchar são:

Tipo	Operadores aritméticos	Operadores de comparação a outro cchar	Operadores lógicos	Concatenação
cchar		Igual Diferente Menor igual Maior igual Menor que Maior que		Podem ser concatenados aos tipos int, float, char, cchar, bool(Formam um cchar)

O tipo cchar não suporta constantes literais.

### Booleano

A palavra **boolean** define uma variável que tem como valores *true* ou *false*.

Sendo a declaração da seguinte forma:

**bool** variavel;

Tipo	Operadores aritméticos	Operadores de comparação	Operadores lógicos	Concatenação
bool		Igual Diferente	AND OR NOT	Se concatenados a um caractere ou cadeia de caracteres

O tipo booleanos só aceita duas constantes literais: 'true' ou 'false'.

### 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:

O tamanho e acesso ao vetor é feito usando exclusivamente inteiros.

**vector** tipo variavel[tamanho\_int];

**vector** int exemplo[10];

Tipo	Operadores aritméticos	Operadores de comparação	Operadores lógicos	Concatenação
vector	Os elementos do vetores vão suportar as operações suportadas pelo tipo do vetor.	Os elementos do vetores vão suportar as operações suportadas pelo tipo do vetor	Os elementos do vetores vão suportar as operações suportadas pelo tipo do vetor	Os elementos do vetores vão suportar as operações suportadas pelo tipo do vetor

## Coerção

A ENL faz conversão automática entre valores do tipo float e int. Caso uma variável do tipo float seja atribuída a uma variável do tipo inteiro, o valor guardado será apenas a parte inteira do número

# Casting

A ENL aceita casting nos seguintes formatos:

- Número do tipo inteiro para cchar
- Número do tipo float para cchar
- Caractere do tipo char para cchar

# Atribuição

A atribuição é feita usando o comando '='. Funciona de modo tal que o valor da esquerda do comando = , recebe o valor da direita.

EX:

```
void Begin(){  
    int a;  
    a = 1;  
}
```

# Concatenação

A linguagem ENL suporta concatenação de caracteres através do operador ++, o novo tipo formado é do tipo cchar.

# Comentários

Pode se fazer comentários usando o operador # no início e final do comentário.

Não é possível fazer comentário dentro de outro comentário.

# Funções

As funções são definidas antes da função begin(), com a palavra reservada *function*, e após o tipo do retorno, o identificador da variável.



Dentro de uma função devem ser indicados o tipo e o identificador do parâmetro e podem ser int, float, char, cchar, boolean, vector além que a função pode receber outra função como parâmetro.

Exemplo de declaração de função:

```
function tipo_de_retorno identificador( tipo parametro1, tipo parametro2,  
... , tipo parametroN ){  
  
    return valor_ou_alguma_variável;  
}
```

## Inicialização

As variáveis podem ser inicializadas no momento da criação por um valor ou expressão. Caso não sejam inicializadas, seus valores default são os que seguem:

Tipo	Valor default
int	0
float	0.0
bool	false
array	inicializado com o valor padrão de seu tipo definido
char, string	null

## Operadores

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

Operadores aritméticos	Símbolo
Soma Subtração Multiplicação Divisão Módulo unário negativo	+ - * / % -

Operadores de comparação	Símbolo
Igual Diferente Menor igual Maior igual Menor que Maior que	== != <= >= < >

Operadores lógico	Símbolo
AND OR NOT	&&    !

#### 4.1. Precedência e associatividade

A tabela abaixo mostra a precedência e associatividades dos operadores, a mesma está ordenada do operador de maior precedência para o de menor precedência.

Operadores	Associatividade
------------	-----------------

~	Da direita para a esquerda
* /	Da esquerda para a direita
%	Da esquerda para a direita
+ -	Da esquerda para a direita
< > <= >=	Da esquerda para a direita
== !=	Da esquerda para a direita
!	Da direita para a esquerda
&&	Da esquerda para a direita

## Instruções

Estrutura condicional de uma via:

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

Estrutura condicional de duas vias:

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

Estrutura iterativa com controle lógico:

```
while ( expressão booleana ){
```

```
        instruções
    }
```

Estrutura iterativa controlada por contador:

```
for (i in 1 to n step inteiro do)
{
    instruções
}
```

Se o passo for 1 pode ser omitido da expressão.

ex:

```
for (i in 1 to n do)
{
    instruções
}
```

A palavra reservada 'break' pode ser usada para sair da estrutura iterativa.

## Impressão

A função put recebe uma string de formatação e imprime algo na tela, também pode receber caracteres e imprimi-los na tela.

%d para inteiros

%f para float

%c para char

%s para cchar

%b para bool

para imprimir os elementos do vector é necessário usar uma estrutura de iteração ou de controle lógico especificando o tipo de seu elemento

put('string de formatação', variáveis ou expressões a imprimir separados por vírgula)

ex:

```
put("%d", 10);
```

A instrução acima imprime 10 na tela;

## Leitura

A função read recebe argumentos e lê algo digitado pelo usuário.

%d para inteiro

%f para float

%c para char

%s para cchar

%b para bool

para ler os elementos do vector é necessário usar uma estrutura de iteração ou de controle lógico especificando o tipo de seus elementos

read('tipos a serem lidos separados por vírgula', variáveis que vão armazenar os valores lidos separados por vírgula)

ex:

```
void begin()
{
    int i;
    read('%d', i);
}
```

o programa lê um inteiro digitado pelo usuário e armazena na variável i.

## Programas:

Ola mundo:

```
void begin()
{
    put('%s', 'Ola mundo');
}
```

### Fibonacci

```
function void fibonacci(int num)
{
    int i, j, t, k;
    j = 1;
    k = 1;
    i = 0;

    if (num <= 0)
    {
        put('%s', 'Não é possível fazer uma sequencia fibonacci');
    }

    else
    {
```

```

        put('%d', 1);

        while(k <= num)
        {
            t = i + j;
            i = j;
            j = t;

            if(j <= num)
            {
                put(' ', %d', j);
            }
            k = k + 1;
        }
    }

}

void main()
{
    int num;
    read('%d', num);
    fibonacci(num);
}

```

## Shell Sort

```

function void shellSort(vector<int> vet, int tamanho)
{
    int i, j, value;
    int h = 1;

    while(h < tamanho)
    {
        h = 3 * h + 1;
    }

    while (h > 0)
    {
        for (i in h to tamanho do)
        {

```

```

        value = vet[i];
        j = i;
        while (j > h-1 && value <= vet[j - h])
        {
            vet[j] = vet [j - h];
            j = j - h;
        }
        vet[j] = value;
    }
    h = h/3;
}
return vet;
}

```

```

void begin()
{

    vector<int> vetor[1000];
    int tam;

    read("%d", tam);

    for(i in 1 to tam do){
        read("%d" , vetor[i]);
    }

    for(i in 1 to tam do){
        put("%d", vetor[i]);
    }

    vetor = shellsort(vetor, tam);

    for(i in 1 to tam do){
        put("%d", vetor[i]);
    }

}

```

