Universidade Federal de Alagoas - UFAL
Instituto de Computação - IC
Curso de Ciência da Computação

Professor Alcino Dall'Igna Júnior

Especificação da Linguagem

Thalyssa de Almeida Monteiro

Maceió, 03 de dezembro de 2022

Sumário

1 Introdução

A linguagem de programação **Neon** tem como base as linguagens C e C++. Não admite escopo global e não possui conversão implícita de tipos (coerção), é estaticamente tipada e a compatibilidade de tipos é realizada por nome. Não é orientada à objetos e é sensível ao caso (*case sensitive*).

Os programas escritos em **Neon** devem possuir a extensão ".nbl".

O nome da linguagem deriva de uma música que a autora descobriu recentemente.

2 Estrutura Geral

Um programa escrito em Neon deve possuir:

- A palavra reservada de início do programa (`Begin`) seguido de chaves ({ }) e o restante do código deve ficar entre essas chaves.
  Exemplo:

```
Begin {
    ...
    code
    ...
}
```

- Uma função deve apresentar a palavra reservada `Function`, seu tipo de retorno, um nome e seus parâmetros dentro de parênteses e separados por vírgula, com cada parâmetro apresentando primeiro seu tipo e depois seu identificador. Por fim, colocam-se também chaves ({ }) e o código da função entre estas chaves, juntamente com a palavra reservada de retorno (`Return`).
  Exemplo:

```
Function ReturnType functionName(TypeA paramA, TypeB paramB, ...)
{
    ...
    code
    ...
}
```

- Exemplo de código completo:

```
Begin {
    Function Int myFunction(Int var1, Int var2) {
        Int result = var1 + var2;
        Return result;
    }

    Int num1, num2, res;
    Get(num1);
    Get(num2);

    res = myFunction(num1, num2);
    Show(res);
}
```

3 Tipos de dados e Nomes

3.1 Variáveis globais/locais

As variáveis devem ser declaradas dentro das funções em que serão utilizadas.

3.2 Palavras reservadas

Lista de palavras reservadas: **`Int`**, **`Float`**, **`Char`**, **`String`**, **`Bool`**, **`Begin`**, **`End`**, **`If`**, **`Else`**, **`While`**, **`From`**, **`To`**, **`Increase`**, **`Get`**, **`Show`**, **`Return`**, **`Function`**, **`True`**, **`False`**, **`Array`**, **`And`**, **`Or`**.

3.3 Identificadores

Não há limite de tamanho para os identificadores e todos devem seguir as seguintes regras:

- Todas as variáveis devem iniciar com letra minúscula e funções devem ser iniciadas com letra maiúscula;
- A partir do segundo caractere podem ser usadas letras maiúsculas ou minúsculas;
- Não é permitido o uso de espaços;
- Não é permitido o uso de palavras reservadas.

3.4 Comentários

Os comentários são apenas por linha, onde cada comentário deve iniciar com o caractere: "@".

Exemplo:

```
@ This is a comment
```

3.5 Declaração de variáveis

A declaração de variáveis deve apresentar o tipo da variável e seu nome (vide o item 3.3). Uma atribuição de valor imediata é opcional.

Exemplo:

```
<type> identifierA;
<type> identifierB = value;
```

Os tipos disponíveis para utilização são do item 3.6 ao item 3.11.

3.6 Inteiro

Representa um número inteiro de 64 bits, deve ser identificado pela palavra reservada `Int` e seus literais são da forma `[0-9]+`

Exemplo:

```
Int integer;
Int myInt = 16;
Int myOtherInt = 5500;
```

3.7 Ponto Flutuante

Representa um número real de 64 bits, deve ser identificado pela palavra reservada `Float` e seus literais são da forma `[0-9]+\.[0-9]*`

Exemplo:

```
Float floatingPoint;
Float myFloat = 7.64;
```

3.8 Caracteres

Cada caractere representa um símbolo da tabela ASCII, estes sendo denotados por um valor entre 0 e 127, armazenados em 1 byte.

Deve ser identificado pela palavra reservada `Char` e ao ser atribuído à uma variável, deve vir entre aspas duplas (”). Para denotar as próprias aspas, deve-se usar o caractere de escape contrabarra (\), o mesmo vale para o próprio caractere de escape.

Exemplo:

```
Char a = "a";
Char b = "\"";
```

3.9 Cadeia de Caracteres

Representam uma sequência de caracteres e deve ser identificado pela palavra reservada `String`.

Ao ser atribuída uma string à uma variável, deve-se utilizar aspas duplas (”). Para denotar as próprias aspas, a mesma regra do item 3.8 se aplica.

Exemplo:

```
String str;
String myString = "spam";
String otherString = "e \" ggs";
```

3.10 Arranjo Unidimensional

Define-se utilizando o tipo dos valores que estarão contidos no vetor juntamente com o identificador e logo após o tamanho, ou seja, a quantidade de elementos dentro do vetor, entre colchetes ([ ]).

Exemplo:

```
<type> indetifier[size];
<type> indetifier[size] = {<values>};
```

Para o segundo caso, onde os valores já são passados no momento da inicialização do vetor, o tamanho do mesmo é opcional.

Cada elemento do vetor pode ser acessado utilizando

```
identifier[index];
```

onde o `index` varia de `0` até `size - 1`.

Existem duas possibilidade para o caso de a quantidade de elementos em um vetor ser diferente da quantidade declarada:

1. Quantidade menor que a declarada:
   O vetor é preenchido com os valores padrão do tipo declarado.

2. Quantidade maior que a declarada:
   Erro.

3.11 Booleano

Representam dois valores apenas: verdadeiro e falso, estes denotados por `True` e `False`, e devem ser identificados utilizando a palavra reservada `Bool`.

Exemplo:

```
Bool boolean;
Bool myBool = False;
```

3.12 Valores Padrão

Os valores padrão são atribuídos às variáveis no momento de sua criação, caso um valor não seja atribuído à elas pelo programador.

Tipo	Valor
`Int`	`0`
`Float`	`0.0`
`Char`	” ”
`String`	” ”
`Bool`	`False`

3.13 Operações Aceitas por Tipo

Tipo	Operações
`Int`	Atribuição, Aritmética, Relacional
`Float`	Atribuição, Aritmética, Relacional
`Char`	Atribuição, Relacional
`String`	Atribuição, Relacional
`Bool`	Atribuição, Relacional*, Lógica

* somente operadores ”==” e ”! =” são aceitos.

4 Operadores

4.1 Aritméticos

Operação	Símbolo	Exemplo
Soma	$+$	$a+b$
Subtração	$-$	$a-b$
Multiplicação	$*$	$a*b$
Divisão	$/$	a/b

4.2 Relacionais

Operação	Símbolo	Exemplo
Igual	$==$	$a==b$
Diferente	$!=$	$a!=b$
Maior que	$>$	$a>b$
Menor que	$<$	$a<b$
Maior ou igual	$>=$	$a>=b$
Menor ou igual	$<=$	$a<=b$

4.3 Lógicos

Operação	Símbolo
Conjunção	`And`
Disjunção	`Or`

4.4 Precedência

Quanto menor o valor, maior sua precedência.

Operador	Precedência
$*$, $/$	1
$+$, $-$	2
$==$, $!=$	3
$>$, $<$, $>=$, $<=$	4
`and`	5
`or`	6

5 Instruções

5.1 Atribuição

Define-se pelo operador '=', onde do lado esquerdo fica o identificador que receberá o valor e do lado direito o valor a ser atribuído.

Exemplo:

```
Int myInt = 519;
String myString = "spam and eggs";
```

5.2 Estrutura Condicional

Para uma estrutura simples, utiliza-se a palavra reservada `If` para identificar o inicio do bloco condicional, seguida pela expressão lógica que, ao ser verdadeira, entrará na condição, e então chaves ({ }), onde ficará o código a ser executado.

Já em uma estrutura composta, utiliza-se o mesmo da estrutura simples com a adição que, após o fecha chaves (}) que encerra o bloco do `If`, vem a palavra reservada `Else` identificando o bloco que deverá ser executado caso a condição da expressão lógica inserida no primeiro bloco seja falsa. O bloco do `Else` não possui expressão lógica associada diretamente à ele.

Exemplo:

- Simples:

```
If expressao_logica {
    ...
    code
    ...
}
```

- Composta:

```
If expressao_logica {
    ...
    code
    ...
} Else {
    ...
    code
    ...
}
```

5.3 Estrutura Iterativa com Controle por Contador

Utiliza-se a palavra reservada `From` seguida de uma variável que será utilizada como contador já com seu valor inicial atribuído, seguido da palavra reservada `To` e o valor de destino do contador e por fim a palavra reservada `Increase` e o valor que será incrementado no contador. Um bloco de chaves ({ }) identifica o código que deverá ser executado durante as iterações.

Exemplo:

```
Int varInicio = valorInicial;
From varInicio To valorDestino Increase valorIncremento {
    ...
    code
```

```
    ...
}
```

5.4 Estrutura Iterativa com Controle Lógico

Utiliza-se a palavra reservada `While` seguida de uma expressão lógica. Um bloco de chaves ({ }) identifica o código que deverá ser executado durante as iterações.

Exemplo:

```
While expressao_logica {
    ...
    code
    ...
}
```

5.5 Entrada e Saída

- Entrada:

  É realizada através da instrução `Get()` que recebe como argumento as variáveis que irão receber a(s) entrada(s). Os tipos das variáveis que receberão as entradas deve obrigatoriamente ser iguais aos tipos dos valores inseridos.

  Exemplo:

```
<type> identifier;
Get(identifier);
```

- Saída:

  É realizada através da instrução `Show()` que recebe como argumentos:

  1. Uma cadeia de caracteres, que pode conter ou não instruções de formatação;
  2. As variáveis que deverão ser impressas, separadas por vírgula, caso haja instruções de formatação.

A formatação deve ser feita como segue:

Tipo	Formatação
`Int`	%d
`Float`	%*f
`Char`	%c
`String`	%s
`Bool`	%b

* Valor numérico opcional que define a quantidade de casa decimais impressas.

Exemplos:

```
Show("Hello, World!");
Show("%d", intVar);
Show("Spam and %s", "eggs")
Show("%3f", floatVar);
```

6 Programas Exemplo

6.1 Hello, World!

```
Begin {
    Show("Hello, World!");
}
```

6.2 Fibonacci

```
Begin {
    Function Int fibonacci(Int value) {
        Int last = 1;
        Int penultimate = 1;
        Int next;

        If value == 1 {
            Return 1;
        }
        If value == 2 {
            Return 1;
        }

        Int count = 3;
        While count <= value {
            next = last + penultimate;
            penultimate = last;
            last = next;
            count = count + 1;
        }

        Return next;
    }

    Int nthTerm;
    Int result;

    Get(nthTerm);

    result = fibonacci(value);
    Show("%d", result);
}
```

6.3 Shell Sort

```
Begin {
    Function Int shellSort(Array array[], Int n) {
        Int gap = n / 2;
        While gap > 0 {
            Int i = gap;
            From i To (n - 1) Increase 1 {
                Int temp = array[i];
                Int j = i;
```

```
                While j >= gap And array[j - gap] > temp {
                    array[j] = array[j - gap];
                    j = j - gap;
                }
                array[j] = temp;
            }
        }
        Return 0;
    }

    Array Int array[10];
    Int i = 0;

    From i To 9 Increase 1 {
        int input;
        get(input);
        array[i] = input;
    }

    shellSort(array, 10);

    From i To 9 Increase 1 {
        Show("%d ", array[i]);
    }
}
```