

Universidade Federal de Alagoas  
Instituto de Computação  
*Ciência da Computação*

---

Nova - Especificação v2

Rubens Pessoa

23 de outubro de 2016



# Sumário

<b>Sumário</b>	<b>i</b>
<b>1 Introdução</b>	<b>1</b>
<b>2 Conjunto de Tipo de Dados</b>	<b>1</b>
2.1 Identificador . . . . .	1
2.2 Comentário . . . . .	1
2.3 Inteiro . . . . .	1
2.4 Ponto flutuante . . . . .	1
2.5 Caracteres e cadeias de caracteres . . . . .	2
2.6 Boolean . . . . .	2
2.7 Vetores unidimensionais . . . . .	2
2.8 Operações suportadas . . . . .	3
2.9 Coerção . . . . .	3
<b>3 Conjuntos de Operadores</b>	<b>3</b>
3.1 Atribuição . . . . .	3
3.2 Aritméticos . . . . .	3
3.3 Relacionais . . . . .	4
3.4 Lógicos . . . . .	4
3.5 Concatenação de cadeias de caracteres . . . . .	4
<b>4 Precedência e Associatividade</b>	<b>4</b>
<b>5 Instruções</b>	<b>5</b>
5.1 Estrutura condicional de uma e duas vias . . . . .	5
5.2 Estrutura iterativa com controle lógico . . . . .	5
5.3 Estrutura iterativa controlada por contador . . . . .	6
5.4 Entrada e saída . . . . .	6
5.5 Funções . . . . .	6
<b>6 Exemplos de Códigos</b>	<b>7</b>
6.1 Alô mundo . . . . .	7
6.2 Fibonacci . . . . .	7
6.3 Shell sort . . . . .	8



# 1 Introdução

A Nova consiste em uma linguagem de propósito geral, estruturada, imperativa e fortemente tipada. Inspirada na linguagem de programação C. Cada instrução deverá terminar com um ponto e vírgula. O escopo de suas variáveis será local e será possível a criação de funções afim de melhorar a legibilidade e escritabilidade.

## 2 Conjunto de Tipo de Dados

### 2.1 Identificador

Seu identificador segue a seguinte forma:

- Inicia-se, obrigatoriamente, com uma letra maiúscula ou minúscula.
- Os demais caracteres podem ser letras, números ou underline.
- Seu tamanho é ilimitado.
- É vedada a utilização de espaços em branco.

### 2.2 Comentário

Os comentários serão indicados com o seguinte caractere “#”. Desta forma, o que se escrever na linha após estes caracteres será descartado.

### 2.3 Inteiro

O tipo inteiro é declarado pela palavra reservada “int” seguido do identificador da variável. Por ser uma linguagem fortemente tipada, não há coerção entre tipos. Ela denota um número inteiro e seus literais são declarado por qualquer número. As operações para este tipo estão em 3.2. Exemplo:

---

```
1  int inteiro = 1;
```

---

### 2.4 Ponto flutuante

O tipo ponto flutuante é declarado pela palavra reservada “float” seguido do identificador da variável.

Um literal do tipo ponto flutuante é dado por qualquer número real separado por um “.” de suas casas decimais. As operações para este tipo estão em 3.2.

Exemplo:

---

```
1 float real = 1.2;
```

---

## 2.5 Caracteres e cadeias de caracteres

O tipo string é declarado pela palavra reservada “string” seguido do identificador da variável. Da mesma forma serão tratados as unidades de caracteres. Seus literais são um conjunto de caracteres de tamanho mínimo 0 e tamanho máximo ilimitado e são delimitados por aspas duplas. As operações para este tipo estão em 3.5.

Exemplo:

---

```
1 string str = ‘‘Hello World!’’;  
2 string character = ‘‘A’’;
```

---

## 2.6 Boolean

O tipo boolean é declarado usando a palavra reservada “bool” seguido do identificador da variável, os únicos dois possíveis valores para a variável são “True” e “False”. As operações para este tipo estão em 3.3 e 3.4.

Exemplo:

---

```
1 bool flag = True;
```

---

## 2.7 Vetores unidimensionais

Os vetores unidimensionais serão tratados como listas, as quais serão declaradas informando o tipo que será armazenado na lista com acesso aleatório, terá um identificador e um “::” separando o identificador do tamanho máximo da lista que consistirá de um literal ou variável do tipo inteiro.

Exemplo:

---

```
1 int list::3;  
2 string words::10;
```

---

Tabela 1: Operações Suportadas

Tipo	Operações Suportadas
int	atribuição, aritméticos e relacionais
float	atribuição, aritméticos e relacionais
string	atribuição, relacionais e concatenação
boolean	atribuição, relacionais e lógicos

## 2.8 Operações suportadas

Todos os tipos suportam as operações descritas na tabela 1 com valores correspondentes ao seu respectivo tipo.

## 2.9 Coerção

A NOVA não admite coerção implícita de tipos. Para converter de qualquer tipo para string explicitamente, basta usar aspas ao redor do valor ou identificador.

# 3 Conjuntos de Operadores

## 3.1 Atribuição

A atribuição é feita pelo operador “=”, onde do lado esquerdo é o ID da variável e do lado direito é o valor a ser guardado. Os dois lados devem possuir o mesmo tipo, pois a linguagem não permite coerção. Exemplo:

---

```
1  int variable = 10;
```

---

## 3.2 Aritméticos

- “+”: Soma dos dois operandos.
- “-”: Diferença dos dois operandos.
- “\*”: Multiplicação dos dois operandos.
- “/”: Divisão dos dois operandos.
- “%”: Resto da divisão dos dois operandos.

O operador unário negativo é “not”. Este irá negar valores aritméticos dos números, tanto inteiro, quanto ponto flutuante.

### 3.3 Relacionais

- “==”: Igualdade entre dois operandos.
- “!=”: Desigualdade entre dois operandos.
- “<”: Operador ”menor que”.
- “>”: Operador ”maior que”.
- “<=”: Operador ”menor ou igual que”.
- “>=”: Operador ”maior ou igual que”.

Exemplo:

---

```
1  if (a == b) {  
2      # do something  
3  }
```

---

### 3.4 Lógicos

- not: Operador unário que nega uma expressão lógica.
- and: Executa um “and” lógico.
- or: Executa um “or” lógico.

Exemplo:

---

```
1  if (a > b or c == d) {  
2      # do something  
3  }
```

---

### 3.5 Concatenação de cadeias de caracteres

A concatenação de cadeias de caracteres será dada pelo operador binário sobrecarregado “+”.

## 4 Precedência e Associatividade

A Tabela 2 mostrará a precedência e associatividade dos operadores ordenadas decrescentemente de acordo com a precedência.



Operadores	Associatividade
()	Não associativo
not	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
and or	Da esquerda para a direita
=	Não associativo

Tabela 2: Tabela de precedência e associatividade

## 5 Instruções

Como é uma linguagem inspirada em C, temos que seus statements são similares e portanto são terminados sempre na presença de um “;”. Entretanto, há uma diferença: Estruturas condicionais e iterativas também devem terminar com “;”.

### 5.1 Estrutura condicional de uma e duas vias

#### if, if-else

A estrutura condicional “if” será sempre relacionada a uma condição lógica ou à uma variável booleana dentro de parenteses e seu escopo será definido por chaves e ponto-virgula. O algoritmo irá executar o código contido no “if”, se e somente se, a sua condição lógica resultar em True, em caso negativo (False), pode ser criada uma cláusula “else”, que será executada no caso de a condição lógica resultar em False.

### 5.2 Estrutura iterativa com controle lógico

#### while

A estrutura de loop “while” é usado como uma repetição condicional, ou seja, a repetição só irá parar se a sua condição for falsa. Desta forma, o “while” será sempre relacionada a uma condição lógica ou à uma variável booleana dentro de parenteses e seu escopo será definido por chaves e finalizado com um ponto-virgula.

### 5.3 Estrutura iterativa controlada por contador

**for**

O “for” será uma estrutura de repetição que receberá três parâmetros: índice, limite e passo. Ele irá repetir o bloco de código desejado no intervalo [*índice*, *limite*) variando em *passo* até todo intervalo ser passado.

### 5.4 Entrada e saída

Seguem as funções de input e output:

- readIn(identificador)
- printOut(String string)

### 5.5 Funções

A NOVA não suportará sobrecarga de funções. Na declaração da função será necessário que seja definida o tipo de retorno, o nome da função e dentro de parênteses todos os parâmetros e seus respectivos tipos. Seu retorno será feito com a palavra reservada “shoot”. Para chamá-la, utilizaremos o nome da função e, dentro dos parênteses, os valores que utilizaremos como parâmetros.

Exemplo:

---

```
1  int sumInt(int x, int y) {  
2      shoot x + y;  
3  };  
4  
5  void main() {  
6      int a = sumInt(5, 2);  
7  };
```

---

Para Arranjos unidimensionais, a passagem de parâmetros é feita por referência, os outros tipos são por valor.

Para funções há um tipo especial chamado “void”, em que significa que tal função retornará nenhum tipo. Não é possível uma variável ser do tipo “void”.

## 6 Exemplos de Códigos

### 6.1 Alô mundo

---

```
1 void main() {  
2     printOut(" Alo_Mundo!" );  
3 };
```

---

### 6.2 Fibonacci

---

```
1 int fibonacci(int n) {  
2     int f1 = 0;  
3     int f2 = 1;  
4     int fi = 0;  
5  
6     printOut(0);  
7     printOut("", "");  
8     printOut(1);  
9  
10    if (n == 0 or n == 1) {  
11        shoot 1;  
12    };  
13  
14    while( fi < n) {  
15        fi = f1 + f2;  
16        f1 = f2;  
17        f2 = fi;  
18        printOut("", "");  
19        printOut(" fi" );  
20    };  
21  
22    shoot fi;  
23 };  
24  
25 void main() {  
26     int n;  
27     readIn(n);  
28  
29     int fib = fibonacci(n);  
30     # do something with fib
```

31 };

---

### 6.3 Shell sort

---

```
1 void main() {
2     int size;
3     readIn(size);
4
5     int vet::size;
6     for (int i = 0; i < size; i = i + 1) {
7         int x;
8         readIn(x);
9         add(vet, x);
10    };
11
12    int value;
13    int gap = 1;
14    while(gap < size) {
15        gap = 3 * gap + 1;
16    };
17
18    while(gap > 1) {
19        gap = gap / 3;
20        for (int i = gap; i < size; i = i + 1) {
21            value = getValue(vet, i);
22            int j = i - gap;
23
24            while(j >= 0 and value < getValue(vet, j)) {
25                setValue(vet, j + gap, getValue(vet, j));
26                j = j - gap;
27            };
28
29            setValue(vet, j + gap, value);
30        };
31    };
32    };
```

---