# Folha 1 - Revisão: Escrita de algoritmos em pseudo-código e verificação de correção

# Descrição da linguagem

**Constantes:** inteiros, números reais, caracteres e sequências de caracteres constantes. Usaremos a notação `'A'`, `'B'`, `'9'`, ..., para representar o código dos caracteres `A`, `B`, `9`, ... e `"AB9"` para representar a sequência de caracteres `AB9`.

**Variáveis:** representadas por sequências de letras ou dígitos que começam por uma letra. Podem ser *simples* – por exemplo, $maior$, $Aux1$, $y$, ...; *dimensionadas* – por exemplo *arrays* unidimensionais e bidimensionais (abstrações de vetores e matrizes), ou estruturas mais complexas.

Salvo indicação em contrário, convencionamos que a primeira posição de um vetor $x$ é referida como $x[0]$, a segunda $x[1]$, .... Se $x$ for um vetor, $i$ uma variável simples, $x[i]$ refere a posição de índice $i$ do vetor $x$. Se $mat$ for uma matriz, $i$ e $j$ variáveis simples, $mat[i, j]$ refere o elemento que está na linha $i$ e na coluna $j$. Como anteriormente, convencionamos que a primeira linha (respectivamente coluna) é a linha $0$ (respectivamente coluna $0$).

**Expressões Aritméticas:** definidas à custa de constantes e/ou variáveis, usando operadores binários $+$, $-$, $/$ e $*$ (soma, diferença, quociente e produto) e operador unário $-$ (sinal $-$). Poderá ainda ser usado $\%$ para designar o resto da divisão inteira.

**Condições:** – definidas à custa de expressões, operadores relacionais $=$, $\neq$, $<$, $>$, $\geq$, e $\leq$, e operadores lógicos $\neg$ (negação), $\wedge$ (conjunção) e $\vee$ (disjunção).

**Instruções:** – consultar a tabela.

**Programas (ou algoritmos)** – sequências de instruções.

Por vezes, na indicação de um bloco de instruções, omitiremos as chavetas mas passamos a considerar que a **indentação é relevante**. Assim, por exemplo, os dois excertos serão equivalentes.

```
Enquanto (m[i] ≠ 0) fazer {
  m[j] ← m[i];
  j ← j + 1;
  i ← i + 1;
}
```

```
Enquanto (m[i] ≠ 0) fazer
  m[j] ← m[i];
  j ← j + 1;
  i ← i + 1;
```

Para representar **funções**, usamos a notação *nome(Arg1,Arg2,...,ArgN)* e consideramos que as variáveis simples são passadas por valor e as restantes por referência. Usaremos "**retorna** *expressão;*", sem aspas, para as instruções de retorno.

| **Sintaxe** | **Semântica** |
|---|---|
| variável ← expressão; | Avaliar a *expressão* e colocar o seu valor na *variável*.<br>$x \leftarrow 3 * 2$; \| equivale $x \leftarrow 6$;<br>$maior \leftarrow y$; \| copia valor em $y$ para $maior$<br>$y \leftarrow'$ a$' - '$ A$'$; \| coloca $97 - 65 = 32$ em $y$ |
| **Se** (condição) **então**<br>{ instruções } | Avalia a *condição*. Se for verdade, executa o bloco de *instruções*. Senão, passa à instrução seguinte.<br>Se ($m[i] \neq 0$) então {<br>$m[j] \leftarrow m[i]$;<br>$j \leftarrow j + 1$; } |
| **Se** (condição) **então**<br>{ instruções 1 }<br>**senão** { instruções 2 } | Avalia a *condição*. Se for verdade, executa o bloco de *instruções 1*. Senão, executa *instruções 2*.<br>Se ($m[i] \neq 0$) então {<br>$m[j] \leftarrow m[i]$;<br>} senão $m[j] \leftarrow 2$; |
| **Enquanto** (condição) **fazer**<br>{ instruções } | Avalia a *condição*. Se for verdade executa *instruções*. Depois, volta a testar a *condição*. Se ainda for verdadeira, volta a executar as *instruções*, e procede analogamente até a *condição* ser falsa.<br>Enquanto ($m[i] \neq 0$) fazer {<br>$m[j] \leftarrow m[i]$;<br>$j \leftarrow j + 1$;<br>$i \leftarrow i + 1$; } |
| **Repita**<br>{ instruções }<br>**até** (condição); | Executa *instruções*. Depois testa a *condição*. Se for falsa, volta a executar as *instruções*. Volta a testar,..., até a *condição* ser satisfeita.<br>Repita {<br>$n \leftarrow n + 1$;<br>$i \leftarrow i + 1$;<br>} até ($m[i] = 0$); |
| **Para** $var \leftarrow inicio$ **até** $fim$ **fazer**<br>{ instruções }<br><br>**Para** $var \leftarrow inicio$ **até** $fim$ **com passo** $k$ **fazer**<br>{ instruções } | Equivale a:<br>$var \leftarrow inicio$;<br>Enquanto ($var \leq fim$) fazer {<br>instruções<br>$var \leftarrow var + 1$;<br>}<br><br>No segundo caso, a atualização do valor da variável é feita por $var \leftarrow var + k$; |
| **ler(**variável**)**; | lê (input) valor e coloca na *variável*.<br>ler($N$) \| $N$ fica com o valor dado pelo utilizador |
| **escrever(**expressão**)**;<br>**escrever(**string**)**; | escreve (output) o valor da *expressão*<br>escreve a sequência de caracteres.<br>escrever($Aqui$); \| escreve valor de $Aqui$<br>escrever($m[i]$); \| escreve valor de $m[i]$<br>escrever("*Aqui*="); \| escreve (a sequência) *Aqui*= |
| **parar** | terminar a execução. |
| $/ \star \ldots \star /$ | $/ \star$ anotar comentarios $\star /$ \| instrução não executável |

# Alguns exemplos e exercícios

Apresentamos a seguir alguns problemas e algoritmos para sua resolução. No fim, deixamos alguns problemas.

**Exercício:** Em cada caso, justifique a **correção do algoritmo** apresentado, isto é, analise o estado das variáveis ao longo da execução e justifique que o algoritmo resolve corretamente o problema enunciado. Para isso, caracterize o estado das variáveis antes e depois da execução de cada instrução, de acordo com os valores possíveis para as *instâncias* do problema. No caso dos ciclos, caracterize o estado das variáveis à entrada do ciclo, em cada uma das iterações e à saída do ciclo.

**1.** Imprimir o máximo de dois inteiros a indicar pelo utilizador.

ler$(x)$; ler$(y)$;
Se $(x > y)$ então escrever$(x)$;
senão escrever$(y)$;

**2.** Imprimir o máximo de 1000 inteiros a indicar pelo utilizador.

$ler(maximo)$;
$contagem \leftarrow 1$;
Repita {
  ler$(valor)$;
  Se $(valor > maximo)$ então $maximo \leftarrow valor$;
  $contagem \leftarrow contagem + 1$;
} até $(contagem = 1000)$;
escrever$(maximo)$;

**3.** Imprimir o máximo dos inteiros dados pelo utilizador, não considerando primeiro dos inteiros dados, o qual representa o número de valores que serão dados a seguir e é maior ou igual a 1.

ler$(num)$;
ler$(maximo)$;
$contagem \leftarrow 1$;
Enquanto $(contagem < num)$ fazer
  ler$(valor)$;
  Se $(valor > maximo)$ então $maximo \leftarrow valor$;
  $contagem \leftarrow contagem + 1$;
escrever$(maximo)$;

**4.** Sendo lidos pelo menos dois valores da entrada padrão, determinar quantos são iguais ao primeiro valor lido (o qual não conta). Assumir que $-1$, indica que a sequência terminou.

$resposta \leftarrow 0$;
ler$(primeiro)$;
ler$(x)$;
Enquanto $(x \neq -1)$ fazer {
  Se $(x = primeiro)$ então $resposta \leftarrow resposta + 1$;
  ler$(x)$;
}
escrever$(resposta)$;

**5.** Pedir ao utilizador uma sequência de valores que termina por indicação de $-1$, e imprimir o último valor dado (antes de ser dado $-1$). Assumir que o primeiro valor dado nunca é $-1$.

```
ler(ultimo);
ler(outro);
Enquanto (outro ≠ −1) fazer {
  ultimo ← outro;
  ler(outro);
}
escrever(ultimo);
```

**6.** Guardar uma sequência de valores indicada pelo utilizador. Assumir que a sequência fica no vetor $x$. O número de elementos da sequência é o primeiro valor indicado pelo utilizador.

```
ler(n);
pos ← 0;
Enquanto (pos < n) fazer
    ler(dado);
    x[pos] ← dado;
    pos ← pos + 1;
```

**7.** Imprimir o máximo dos $n$ primeiros elementos de um vetor $x$, já em memória. O valor de $n$ é um inteiro não negativo e o programa não escreve nada se $n$ for zero.

```
Se (n ≠ 0) então
    maximo ← x[0];
    i ← 1;
    Enquanto (i < n) fazer
        Se (x[i] > maximo) então  maximo ← x[i];
        i ← i + 1;
    escrever(maximo);
```

**8.** Suponha dada uma sequência $Seq$ de inteiros não negativos. O primeiro elemento é $Seq[0]$, e o valor 0 indica o fim da sequência. Escrever um segmento de programa para:

**a)** Imprimir o valor máximo em $Seq$ (ou zero, se $Seq$ não contiver elementos, i.e., se $Seq[0] = 0$).

```
i ← 0;
maximo ← 0;
Enquanto (Seq[i] ≠ 0) fazer
        Se (Seq[i] > maximo) então
            maximo ← Seq[i];
        i ← i + 1;
escrever(maximo);
```

**b)** Imprimir o índice da primeira ocorrência do máximo de $Seq$, se $Seq[0] \neq 0$.

Se $(Seq[0] \neq 0)$ então
    $posmax \leftarrow 0$;
    $i \leftarrow 1$;
    Enquanto $(Seq[i] \neq 0)$ fazer
        Se $(Seq[i] > Seq[posmax])$ então $posmax \leftarrow i$;
        $i \leftarrow i + 1$;
    escrever$(posmax)$;

Escrever algoritmos *eficientes* (em termos de tempo e espaço de memória necessários à execução) para os problemas seguintes e justificar a sua correção.

**9.** Contar quantas vezes ocorre o máximo numa sequência de $n$ inteiros lida da entrada padrão, sendo $n \geq 1$ já conhecido.

**10.** Dado um vetor $v$ de inteiros não negativos com $n$ elementos, calcular a soma dos elementos da sequência. O valor fica guardado em $soma$ e será zero por defeito.

**11.** Dado um vetor $v$ com $n$ inteiros não negativos guardados nas $n$ primeiras posições, determinar a sequência $s$ de *somas parciais*, a qual pode ser definida por $s[i] = \sum_{j=0}^{i} v[j]$, com $0 \leq i < n$.

**12.** Imprimir a soma dos dois últimos valores de uma sequência de valores dada pelo utilizador, sendo $-1$ o valor que indica que a sequência terminou. Admita que são dados pelo menos dois valores antes do terminador $-1$ (o qual não é relevante para o resultado).

**13.** Verificar se os valores que o utilizador está a indicar estão ordenados por ordem crescente. Convencionar que a sequência termina quando o utilizador der um inteiro que é igual ao conteúdo da variável $final$ (já inicializada). Imprimirá então a resposta `"sim"` ou ”`"nao"`.

**14.** Compactar uma sequência de inteiros dada num vetor $x$, retirando as repetições de elementos que ocorram em posições adjacentes. À medida que for analisando o vetor deve construir o resultado final. Deve considerar os elementos que ocupam as $n$ primeiras posições (sendo $n$ já conhecido). No fim, $n$ terá o número de elementos da versão compactada.