Folha 1

Escrita de algoritmos em pseudo-código. Análise e demonstração de correção para programas simples. Estatísticas de execução (pior caso e melhor caso)

Descrição da linguagem (pseudo-código)

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, ...; *indexadas* – por exemplo *arrays* unidimensionais e bidimensionais (abstrações de vetores e matrizes), ou estruturas mais complexas. Assumimos que os valores iniciais são **desconhecidos**.

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), **excepto se for dito algo em contrário**.

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 $\leftarrow$ expressão;	Avaliar a *expressão* e colocar o seu valor na *variável*. $x \leftarrow 3 * 2;$ | equivale $x \leftarrow 6;$ $maior \leftarrow y;$ | copia valor em y para $maior$ $y \leftarrow'$ a$' -' $ A$';$ | coloca $97 - 65 = 32$ em y
Se (condição) **então** { instruções }	Avalia a *condição*. Se for verdade, executa o bloco de *instruções*. Senão, passa à instrução seguinte. Se $(m[i] \neq 0)$ então { $m[j] \leftarrow m[i];$ $j \leftarrow j + 1;$ }
Se (condição) **então** { instruções 1 } **senão** { instruções 2 }	Avalia a *condição*. Se for verdade, executa o bloco de *instruções 1*. Senão, executa *instruções 2*. Se $(m[i] \neq 0)$ então { $m[j] \leftarrow m[i];$ } senão $m[j] \leftarrow 2;$
Enquanto (condição) **fazer** { 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. Enquanto $(m[i] \neq 0)$ fazer { $m[j] \leftarrow m[i];$ $j \leftarrow j + 1;$ $i \leftarrow i + 1;$ }
Repita { instruções } **até** (condição);	Executa *instruções*. Depois testa a *condição*. Se for falsa, volta a executar as *instruções*. Volta a testar,..., até a *condição* ser satisfeita. Repita { $n \leftarrow n + 1;$ $i \leftarrow i + 1;$ } até $(m[i] = 0);$
Repita { instruções } **enquanto** (condição);	Executa *instruções*. Depois testa a *condição*. Se for verdade, volta a executar as *instruções*. Volta a testar,..., até a *condição* ser falsa.

Para $var \leftarrow inicio$ **até** fim **fazer**
{ instruções }
 	
Para $var \leftarrow inicio$ **até** fim **com passo** k **fazer**
{ instruções }
	Equivale a:
	$var \leftarrow inicio;$
	Enquanto $(var \leq fim)$ fazer {
	instruções
	$var \leftarrow var + 1;$
	}
	
	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*. ler(N) | N fica com o valor dado pelo utilizador
escrever(expressão); **escrever**(string);	escreve (output) o valor da *expressão* escreve a sequência de caracteres. escrever($Aqui$); | escreve valor de $Aqui$ escrever($m[i]$); | escreve valor de $m[i]$ escrever("*Aqui*="); | escreve (a sequência) *Aqui*=
parar	terminar a execução.
$/ \star \ldots \star /$	$/ \star$ anotar bloco de comentarios $\star /$ | instrução não executável
$//$	$//$ anotar linha de comentário | instrução não executável

Exercícios

1. Pretendemos um programa para imprimir o máximo de uma sequência de inteiros, $x_1, \ldots, x_n$, dada pelo utilizador, o qual começa por indicar o valor de n, que é sempre maior ou igual a 1.

```
ler(n);
ler(maximo);
contagem ← 1;
Enquanto (contagem < n) fazer
    ler(valor);
    Se (valor > maximo) então
        maximo ← valor;
    contagem ← contagem + 1;
escrever(maximo);
```

a) Seja $v_0, v_1, v_2, \ldots, v_k$, com $k \geq 1$ fixo, a sequência de valores que o utilizador deu até ao momento em que a condição de paragem do ciclo vai ser testada pela k-ésima vez. Nesse instante, o que pode afirmar sobre o estado das variáveis n, $valor$, $maximo$ e $contagem$?

b) Prove a propriedade que indicou em **1.a)**, por **indução matemática** sobre k. Para isso, justifique que a condição que enunciou na alínea anterior é **(i)** verdadeira se $k = 1$ e que **(ii)** se for verdade para um certo k então é verdade para $k+1$ (sendo $v_0, v_1, v_2, \ldots, v_k, v_{k+1}$ a sequência que foi dada até ao $(k+1)$-ésimo teste da condição de paragem do ciclo).

c) Complete a frase: *O ciclo pára quando* [ ]. *Portanto, tinham sido lidos* [ ] *além de* v_0. *De acordo com a propriedadade demonstrada, à saída do ciclo, o valor de* $maximo$ *é* [ ]. *Logo, o algoritmo escreve o valor correto.*

d) **Para n fixo**, indique o número de vezes que se executa cada um dos testes ou das instruções indicadas, no **melhor caso** e no **pior caso**. Defina-o como função de n e caracterize as instâncias que conduzem ao pior caso (em que o número é máximo) e ao melhor caso (em que o número é mínimo), se houver diferença.

- $contagem < n$
- $ler(valor)$
- $escrever(maximo)$
- $valor > maximo$
- $maximo \leftarrow valor$

e) Modifique o algoritmo para contar também quantas vezes o valor máximo da sequência ocorre. Use uma variável $cmax$ como contador.

f) Na continuação da alínea anterior, explique porque é que $cmax$ está correto no início do ciclo "Enquanto", se mantém correto ao longo desse ciclo, e está correto no final do ciclo.

2. Para uma sequência de inteiros dada pelo utilizador, pretendemos determinar quantos valores são iguais ao primeiro valor lido (o qual não conta). Assuma que serão lidos pelo menos dois inteiros. O valor -1, indica que a sequência terminou (e supomos que já não faz parte da sequência).

a) Prove formalmente a correção do programa seguinte.

```
resposta ← 0;
ler(primeiro);
ler(x);
Enquanto (x ≠ −1) fazer {
  Se (x = primeiro) então
    resposta ← resposta + 1;
  ler(x);
}
escrever(resposta);
```

b) Se tiverem sido lidos n inteiros, com $n \geq 2$, até $x \neq -1$ falhar, quantas vezes se executou: (1) a instrução $ler(primeiro)$; (2) o teste $x \neq -1$; (3) o teste $x = primeiro$; (4) a atribuição $resposta \leftarrow resposta + 1$, no pior caso e no melhor caso? Caraterize as instâncias que conduzem ao melhor caso e ao pior caso.

3. Pretendemos um programa que leia uma sequência de inteiros da entrada padrão e imprima o número de inteiros dados antes de zero (que é o terminador da sequência). Considere os algoritmos seguintes.

```
//Algoritmo A
c ← 0;
Enquanto (x ≠ 0) fazer
    ler(x);
    c ← c + 1;
escrever(c);
```

```
//Algoritmo B
c ← 0;
ler(x);
Enquanto (x ≠ 0) fazer
    c ← c + 1;
    ler(x);
escrever(c);
```

```
//Algoritmo C
c ← −1;
Enquanto (x ≠ 0) fazer
    ler(x);
    c ← c + 1;
escrever(c);
```

```
//Algoritmo D
c ← −1;
Repita
    ler(x);
    c ← c + 1;
até (x = 0);
escrever(c);
```

a) Designe por a_i o i-ésimo valor lido, se necessário. Para Algoritmo B, caraterize o estado das variáveis c e x no momento em que se vai testar a condição de paragem do ciclo pela k-ésima vez, com $k \geq 1$, relacionando-o com a sequência de valores lida até esse instante. Prove essa propriedade (por indução matemática) e explique como pode ser usada para concluir que o algoritmo B resolve corretamente o problema.

b) Averigue a correção dos algoritmos A, C e D para resolução do problema. Justifique a resposta (indicando casos em que o programa falha ou provando que produz o resultado correto para todos os casos).

4. Escreva em pseudocódigo um algoritmo para imprimir a soma dos dois últimos valores de uma sequência de valores dada pelo utilizador, sendo lidos pelo menos dois valores além do valor -1. Esse valor indica que a sequência terminou. Demonstre formalmente a correção do algoritmo.

5. Pretendemos uma função COMPACTAR(x, n) para compactar uma sequência de n inteiros dada num vetor x, retirando as repetições de elementos que ocorram consecutivamente. Admita que os elementos do *array* são indexados a partir de 0. À medida que for analisando o *array* deve **alterar os valores para construir o resultado final**. A função retorna o número de elementos da versão compactada.

Se $n = 11$ e o conteúdo do *array* for | 7 | 7 | 7 | 5 | 5 | 4 | 8 | 3 | 3 | 4 | 4 | deve retornar 6 e ficar com | 7 | 5 | 4 | 8 | 3 | 4 | 8 | 3 | 3 | 4 | 4 | em memória (nessa região).

a) Escreva a função em pseudocódigo. **b)** Demonstre formalmente a sua correção.