Alguns exercícios em pseudocódigo

Enunciado do problema: Escreva um programa em pseudocódigo que determine as raízes de um polinômio do segundo grau dado por

$$a x^2 + b x + c$$

usando o algoritmo de Báskara. Caso $b^2 - 4ac < 0$, o código deverá emitir um aviso desta ocorrência, caso não o código deve imprimir os valores das raízes.

PROGRAMA Baskara

REAIS a, b ,c, delta, x1, x2

LEIA a, b, c

delta ← b * b – 4.0 * a * c

SE (delta >= 0.0) ENTÃO
x1 ← (-b + sqrt(delta))/(2.0 * a)
x2 ← (-b - sqrt(delta))/(2.0 * a)

IMPRIMA "As raízes do polinômio de coeficientes", a, b, c, " são ", x1, x2, x3

SENÃO
IMPRIMA "delta negativo. Não ha raízes reais"
FIM SE

PARE
FIM

Observe que o código é muito direto. Repare ainda que usamos uma função muito comum nas mais variadas linguagens de programação, o **sqrt**. Trata-se de uma função bem otimizada que calcula a raiz quadrada de um número positivo.

A única questão de destaque está que se põe uma crítica da informação gerada pelo valor **delta** para que o algoritmo, ao ser implementado, não pare de forma imprevista, pois se o **delta** fosse negativo, o cálculo das raízes não seria possível, já que as variáveis foram definidas como pertencentes aos reais. Repare também que o programa não segue a sequência do enunciado pois o teste não é feito para o delta negativo mas foi testado se o delta é positivo. No entanto, funcionalmente não há diferença e o código executa o solicitado.

Mas observem que o código pode parar fora do controle do programador. Se a variável **a** for nula, o pseudocódigo gerará uma indefinição no momento do cálculo das raizes. Assim, vamos reescrever o pseudocódigo de forma que isto não aconteça.

```
PROGRAMA Baskara2

REAIS a, b ,c, delta, x1, x2

LEIA a, b, c

SE (a ≠ 0.0) ENTÃO
    delta ←b * b – 4.0 * a * c

    SE (delta >= 0.0) ENTÃO
        x1←(-b + sqrt(delta))/(2.0 * a)
        x2←(-b - sqrt(delta))/(2.0 * a)

        IMPRIMA “As raízes do polinômio de coeficientes”, a, b, c, “ são “, x1, x2, x3

    SENÃO
    IMPRIMA “delta negativo. Não ha raízes reais”
    FIM SE
SENÃO
    x1←- c/b
    IMPRIMA “O polinômio é do primeiro grau e tem como raiz”, x1
FIM SE
PARE
FIM
```

Enunciado: Dados a, b e c, valores reais positivos e não nulos. Estas valores correspondem aos lados de um triângulo. Determine se o triângulo é equilátero, isósceles ou retângulo.

Descrição prévia do problema:
Teremos que examinar algumas definições:

Um triângulo equilátero é o que tem três lados iguais, enquanto um triângulo isósceles é definido como um triângulo com pelo menos dois lados iguais. Temos por estas definições que o triângulo equilátero é um caso especial de triângulo isósceles.

Temos que um triângulo é retângulo quando a soma do quadrado dos catetos é igual ao quadrado da hipotenusa, ou seja,

$$h^2 = c_1^2 + c_2^2$$

onde aqui h é a hipotenusa (o maior dos lados) e c_1, c_2 são os catetos. Observe que nada impede que os catetos sejam iguais. Portanto, um triângulo isósceles também pode ser retângulo.

O resultado desta análise é que um triângulo pode ter vários aspectos. Observe que podemos dividir nossa tarefa:

Escrevamos um segmento de pseudocódigo para pensarmos: Pretendemos detectar se os lados do triângulo são de um triângulo isósceles.

```
SE (a = b) OU (b = c) OU (a = c) ENTÃO
    IMPRIMA “Triângulo isósceles”
FIM SE
```

Observe que isto dá conta do triângulo isósceles pois o uso do **OU** lógico estabelece que basta que uma destas igualdades seja verdadeira para haver a impressão.

Para o caso do triângulo equilátero fica fácil de perceber que o segmento de pseudocódigo anterior pode ser inspiração para nós pois agora exigiremos que todos os lados seja iguais, ou seja,

```
SE (a = b) E (b = c) E (a = c) ENTÃO
    IMPRIMA “Triângulo equilátero”
FIM SE
```

Repare que este o escrito acima também detecta o triângulo equilátero pois o uso do **E** lógico obriga que a expressão só é verdadeira se todas comparações o são.

Resta-nos o caso do triângulo retângulo. Vamos continuar insistindo no que a gente já viu. Não nos importa qual é a hipotenusa, nos interessa saber apenas se a equação que estabelece a relação entre os lados é cumprida. Veja, então, que o teste abaixo poderia dar conta desta situação

```
SE (a * a = b * b + c * c) OU (b * b = a * a + c * c) OU (c * c = a * a + b * b) ENTÃO
    IMPRIMA “Triângulo retângulo”
FIM SE
```

Certamente não é um segmento de pseudocódigo inteligente! Você não faria a resolução do problema deste jeito. No entanto, funciona de acordo com as especificações do problema. Assim,um pseudocódigo funcional, mas não muito esperto, seria o que se segue

PROGRAMA Triangulos

REAIS a, b ,c

LEIA a, b, c

SE (a = b) OU (b = c) OU (a = c) ENTÃO
    IMPRIMA “Triângulo isósceles”
FIM SE

SE (a = b) E (b = c) E (a = c) ENTÃO
    IMPRIMA “Triângulo equilátero”
FIM SE

SE (a * a = b * b + c * c) OU (b * b = a * a + c * c) OU (c * c = a * a + b * b) ENTÃO
    IMPRIMA “Triângulo retângulo”
FIM SE

PARE
FIM

A falta de esperteza deste código (que, de quebra, é deselegante) está no grande número de comparações e operações feitas. Funciona mas ...

Observe que este código poderá gerar mensagens múltiplas, ora dizendo que o triângulo é isósceles e equilátero, ora dizendo que o triângulo é isósceles e retângulo. Também poderá emitir uma mensagem dizendo que o triângulo é puramente isósceles ou puramente retângulo mas ninguém esperará que surja a mensagem que o triângulo seja equilátero e retângulo. Nada disto é um erro.

Fica uma questão que não foi pedida mas que pode surgir no uso deste pseudocódigo: os valores dados podem ser de um triângulo que não é de nenhuma destas categorias, ou seja, um triângulo escaleno não retângulo. Tente incluir esta opção no pseudocódigo acima. Mas ainda pode haver algo pior: Para podermos construir um triângulo a medida de um dos lados deve ser maior ou igual a soma das medidas dos outros dois lados. Observe que o enunciado diz que os lados são de um triângulo mas não custa nada evitar um engano e deixar o usuário da futura implementação sem saber o que aconteceu. Neste caso, deveríamos fazer uma crítica dos valores de entrada. Vamos modificar o código nada esperto acima:

```
PROGRAMA Triangulos

REAIS a, b ,c

LEIA a, b, c

SE (a +b >= c) OU (b + c >= a) OU (c >= a + c) ENTÃO

    SE (a = b) OU (b = c) OU (a = c) ENTÃO
        IMPRIMA “Triângulo isósceles”
    FIM SE

    SE (a = b) E (b = c) E (a = c) ENTÃO
        IMPRIMA “Triângulo equilátero”
    FIM SE

    SE (a * a = b * b + c * c) OU (b * b = a * a + c * c) OU (c * c = a * a + b * b) ENTÃO
       IMPRIMA “Triângulo retângulo”
    FIM SE

SENÃO
    IMPRIMA “Os valores dados não formam um triângulo”
FIM SE

PARE
FIM
```

Observe que o código continua nada esperto mas funciona. O interessante neste pseudocódigo está no uso sistemático de uma mesma estrutura para resolver um problema, o uso dos operadores lógicos e (espero) uma compreensão fácil. Tirando isto, ele é abaixo de qualquer crítica.

Enunciado do problema: a e **b** são dois inteiros, sendo que **a < b**. Fazer o somatório dos valores compreendidos entre **a** e **b** inclusive. Os valore de **a** e **b** serão dados no corpo do código.

PROGRAMA soma

INTEIROS a, b, s, i
a ← 1
b ← 100

s ← a
i ← a

ENQUANTO (i < b) FAÇA
  i ← i + 1
  s ← s + i
FIM ENQUANTO

IMPRIMA "A soma de ", a, " ate ", b, " = ", s
PARE
FIM

Explanando sobre o pseudocódigo:

Observe que foi criada uma variável auxiliar **i** que é inicializada com o valor de **a**. Funcionalmente, você não necessitaria desta variável para dar o resultado correto. No entanto, sem o uso da variável auxiliar, o valor de **a** seria alterado e no momento da impressão o valores ao final o valor de **a** seria igual a **b** e a mensagem de saída seria sem sentido.

Enunciado do problema: Calcule o fatorial de **n**. O valor de **n** será dado no corpo do código.

PROGRAMA fatorial

INTEIROS n, fat, i
fat ← 1
n ← 5

i ← 1

ENQUANTO (i <= n) FAÇA
  fat ← fat * i
  i ← i + i
FIM ENQUANTO

IMPRIMA "O fatorial de ", n, " = ", fat
PARE
FIM

Enunciado do problema: Calcule o n-ésimo termo da **Sequência de Fibonacci** como ilustrada abaixo:

1, 1, 2, 3, 5, 8, 13, ...

Esta sequência é obtida partindo dos dois primeiros números (que são **1** e **1**) que são somados dado origem ao novo número da sequência que é **2**. Some este termo da sequência com o anterior (**1**) e obtemos o próximo número da sequência que é o **3**. Novamente somamos este número obtido com o anterior (**2**) e obtemos o próximo termo na sequência que é o **5**. De outra forma podemos escrever o crescimento da sequência da seguinte forma

1, 1

1, 1, 2

1, 1, 2, 3

1, 1, 2, 3, 5

1, 1, 2, 3, 5, 8

1, 1, 2, 3, 5, 8, 13.

Aqui paramos no número **13** que é o sétimo termo da **Sequência de Fibonacci.**

```
PROGRAMA Fibonacci

INTEIROS a, b, n, cont, s
fat ← 6
a ← 1
b ← 1

cont ← 2

ENQUANTO (cont <n) FAÇA
   s ← a + b
   a ← b
   b ← s
   cont ← cont + 1
FIM ENQUANTO

IMPRIMA “O ”, n, “-ésimo termo da Sequência de Fibonacci = “, b
PARE
FIM
```

Enunciado do problema: Crie um pseudocódigo que dado um inteiro não negativo seja devolvido quantos algarismos este inteiro tem.

Aqui exercitaremos duas abordagens de resolução.

i) Usaremos como procedimento divisões por dez feitos sobre o número de entrada. Observe que um número em base dez, por exemplo, 327, é de fato

$$3\times10^2+2\times10^1+7\times10^0 \quad .$$

Podemos determinar o número de algarismos, assim como quais são os algarismos se usarmos a divisão do número dado por dez sucessivamente. Mas o nosso objetivo aqui é o mais simples, determinar o número de algarismos.

Obs: lembremos que aqui falamos de divisão de inteiro por inteiro com resultado inteiro. Assim 5/3 é igual a 1 e 3/5 é igual a 0.

Vamos ao nosso exemplo. Se dividirmos 327 por dez teremos como resultado 32, dividindo novamente por dez teremos como resultado 3. Após uma nova divisão teremos o resultado 0. Observe que fizemos 3 divisões por zero e este valor é precisamente o número de algarismos. Isto nos inspira a fazermos um laço de repetição que pare somente se a divisão que fazemos é igual a zero tendo ainda uma variável que conte o número de divisões.

```
PROGRAMA algarismos

INTEIROS numero, nalgarismos, aux
numero ← 327

aux ← numero

nalgarismos ← 1

ENQUANTO (numero ≠ 0) FAÇA
   aux ← aux/10
   nalgarismos ← nalgarismos + 1
FIM ENQUANTO

IMPRIMA  nalgarismos, "é o número de algarismos de  ", numero
PARE
FIM
```

Repare que inicializamos a variável **nalgarismos** com 1 pois pelo menos um algarismo nosso número terá. Outro ponto importante é que criamos uma variável auxiliar **aux** que contém o valor do número original. O motivo disto é que no nosso processamento destruimos o número pelas sucessivas divisões. Para que a impressão ocorra de maneira correta **neste pseudocódigo** é necessário preservar o valor original. Nada me impedi de reescrever este código de forma que não necessitaríamos desta variável axiliar: basta imprimir o valor do número antes de começarmos as divisões.

ii) No lugar de dividir o número original, faremos uma comparação do mesmo com sucessivas potências de dez. Voltemos ao nosso exemplo, ou seja, o número 327. Se compararmos este número com dez elevado a potência zero, nosso número é claramente maior. Se comparamos com dez elevado a potência 1 também, assim com se compararmos com potência 2. No entanto ao compararmos com dez elevado à potência 3, nosso número será menor. Um pseudocódigo possível é

```
PROGRAMA algarismos2

INTEIROS numero, nalgarismos, n
numero ← 327

n ← 1

nalgarismos ← 1

ENQUANTO (numero > n * 10) FAÇA
   nalgarismos ← nalgarismos + 1
FIM ENQUANTO

IMPRIMA  nalgarismos, “é o número de algarismos de  “, numero
PARE
FIM
```

Aqui criamos uma variável auxiliar **n** para fazer as multiplicações sucessivas para nossas comparações. Este algoritmo se diferencia do outro por ser não destrutivo em relação ao valor da variável de entrada.