Computação e Programação

(LEAmb, LEMat, LQ, MEBiol, MEQ)

Departamento de Matemática, IST

Exame 2 – 2 de Fevereiro de 2007

Resolução

Grupo I [1,5+1,5]

a) Defina em Matlab uma função `f` que quando recebe como argumento um vector de números inteiros percorre esse vector posição a posição recorrendo a um ciclo `while` e devolve o número de 7s que ocorrem no vector depois do último 3.

Resolução. (duas versões possíveis)

```
function R=f(V)
I=1;
R=0;
while I<=length(V)
   if (V(I)==3)
      R=0;
   elseif (V(I)==7)
      R=R+1;
   end
   I=I+1;
end
```

```
function R=f(V)
I=length(V);
R=0;
T=1;
while (I>=1 & T)
   if (V(I)==3)
      T=0;
   elseif (V(I)==7)
      R=R+1;
   end
   I=I-1;
end
```

b) Defina em Matlab uma função `g` que quando recebe como argumento uma matriz de números inteiros percorre essa matriz posição a posição recorrendo a dois ciclos `while` encaixados e devolve um vector com tantas posições quantas as linhas da matriz contendo na posição k o número de 7s que ocorrem na linha k depois do último 3.

Resolução. (duas versões possíveis)

```
function R=g(M)
[L,C]=size(M);
I=1;
R=zeros(1,L);
while I<=L
   J=1;
   while J<=C
      if (M(I,J)==3)
         R(I)=0;
      elseif (M(I,J)==7)
         R(I)=R(I)+1;
      end
      J=J+1;
   end
   I=I+1;
end
```

```
function R=g(M)
[L,C]=size(M);
I=1;
R=zeros(1,L);
while I<=L
   J=C;
   T=1;
   while (J>=1 & T)
      if (M(I,J)==3)
         T=0;
      elseif (M(I,J)==7)
         R(I)=R(I)+1;
      end
      J=J-1;
   end
   I=I+1;
end
```

Grupo II [2,0+2,0]

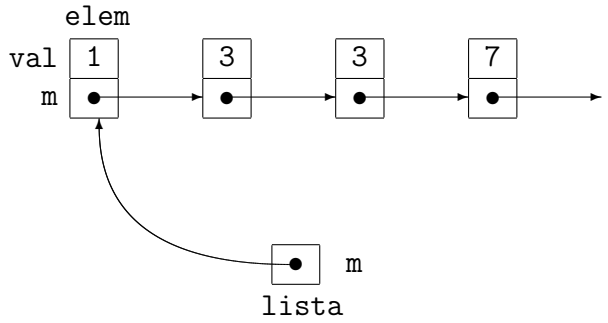
Uma *lista ordenada de inteiros* é uma lista de inteiros cujos elementos estão ordenados de forma crescente.

a) Desenvolva em F um módulo que disponibilize as operações a seguir descritas sobre este tipo de dados, escolhendo a implementação seguinte para o tipo `lista`.

```
type, public :: lista
  private
  type(elem), pointer :: m
end type lista

type, private :: elem
  integer :: val
  type(elem), pointer :: m
end type elem
```

Com esta implementação, pretende-se que uma lista ordenada de inteiros seja representada por uma cadeia de nós. Por exemplo, a lista ordenada $[1, 3, 3, 7]$ é representada pela cadeia de nós seguinte:



As operações disponibilizadas pelo módulo são as seguintes:

- `nil()`: função que devolve a lista vazia;
- `minimo(w,n)`: subrotina que recebe no parâmetro de entrada/saída `w` uma lista ordenada de inteiros e devolve no parâmetro de saída `n` o menor elemento da lista, caso esta não seja vazia;
- `maximo(w,n)`: subrotina que recebe no parâmetro de entrada/saída `w` uma lista ordenada de inteiros e devolve no parâmetro de saída `n` o maior elemento da lista, caso esta não seja vazia;
- `apaga_min(w)`: subrotina que recebe no parâmetro de entrada/saída `w` uma lista ordenada de inteiros e lhe retira o menor elemento, caso a lista não seja vazia;
- `apaga_max(w)`: subrotina que recebe no parâmetro de entrada/saída `w` uma lista ordenada de inteiros e lhe retira o maior elemento, caso a lista não seja vazia;
- `acrescenta(w,n)`: subrotina que recebe no parâmetro de entrada/saída `w` uma lista ordenada de inteiros e no parâmetro de entrada `n` um inteiro e acrescenta (ordenadamente) `n` a `w`;

- `vazia(w)`: função que recebe no argumento `w` uma lista ordenada de inteiros e devolve `.true.` se a lista estiver vazia e `.false.` caso contrário.

Nota: as subrotinas `apaga_min` e `apaga_max` apenas apagam um elemento; caso existam várias cópias do mínimo ou máximo da lista apenas uma é apagada.

Resolução.

```
function nil() result(w)
type(lista) :: w
w%m => null()
end function nil

subroutine minimo(w,n)
type(lista), intent(inout) :: w
integer, intent(out) :: n

if (associated(w%m)) then
    n=w%m%val
end if
end subroutine minimo

subroutine maximo(w,n)
type(lista), intent(inout) :: w
integer, intent(out) :: n
type(elem), pointer :: aux

if (associated(w%m)) then
    aux => w%m
    do
        if (.not.associated(aux%m)) then
            n=aux%val
            exit
        end if
        aux => aux%m
    end do
end if
end subroutine maximo

subroutine apaga_min(w)
type(lista), intent(inout) :: w
type(elem), pointer :: aux

if (associated(w%m)) then
    aux => w%m
    w%m => w%m%m
    deallocate(aux)
end if
end subroutine apaga_min
```

```
subroutine apaga_max(w)
type(lista), intent(inout) :: w

if (associated(w%m)) then
    call apaga_aux(w%m)
end if
end subroutine apaga_max

recursive subroutine apaga_aux(p)
type(elem), pointer :: p

if (.not.associated(p%m)) then
    deallocate(p)
    p => null()
else
    call apaga_aux(p%m)
end if
end subroutine apaga_aux

subroutine acrescenta(w,n)
type(lista), intent(inout) :: w
integer, intent(in) :: n

call acrescenta_aux(w%m,n)
end subroutine acrescenta

recursive subroutine acrescenta_aux(p,n)
type(elem), pointer :: p, aux
integer, intent(in) :: n

if (.not.associated(p) .or. (p%val > n)) then
    allocate(aux)
    aux%val = n
    aux%m => p
    p => aux
else
    call acrescenta_aux(p%m,n)
end if
end subroutine acrescenta_aux

function vazia(w) result(b)
type(lista), intent(in) :: w
logical :: b

b = .not.associated(w%m)
end function vazia
```

b) A *mediana* de uma lista ordenada é o elemento na posição do meio da lista, caso esta tenha um número ímpar de elementos, ou a média dos dois elementos no meio da lista, caso contrário. Desenvolva em F, recorrendo ao módulo acima, uma subrotina `med(w,n)` que recebe no parâmetro de entrada/saída `w` uma lista ordenada de inteiros e devolve no parâmetro de saída `n` a mediana de `w`.

Resolução.

```
subroutine med(w,n)
type(lista), intent(inout) :: w
integer, intent(out) :: n
integer :: x,y

if (.not.vazia(w)) then
    do
        call minimo(w,x)
        call apaga_min(w)
        if (vazia(w)) then
            n=x
            exit
        end if
        call maximo(w,y)
        call apaga_max(w)
        if (vazia(w)) then
            n=(x+y)/2.0
            exit
        end if
    end do
end if

end subroutine med
```

Grupo III [2,0+1,0]

a) Considere a seguinte função `f` definida em F.

```
function f(x,y) result(n)
integer, intent(in) :: x, y
integer :: n

n=0
do
   if (y**(n+1)>x) then
     exit
   end if
   n=n+1
end do
end function f
```

Descreva a funcionalidade da função `f`. Defina uma função recursiva `g` com a mesma funcionalidade de `f`.

Resolução. A função `f` recebe como argumentos dois inteiros `x` e `y` e devolve o maior natural `n` tal que `y**n<=x`, ou seja, o logaritmo (inteiro) de `x` na base `y`.

Uma função recursiva com a mesma funcionalidade é por exemplo a seguinte.

```
recursive function g(x,y) result(n)
integer, intent(in) :: x, y
integer :: n

if (y>x) then
   n=0
else
   n=1+g(x/y,y)
end if
end function g
```

b) Seja f uma função contínua num intervalo $[a, b]$ satisfazendo $f(a)f(b) < 0$. Então f tem uma raiz no intervalo $[a, b]$, isto é, existe um valor x com $a < x < b$ tal que $f(x) = 0$.

Um dos métodos utilizados para encontrar valores aproximados de uma dessas raízes é o método da bissecção, que a seguir se descreve.

Em primeiro lugar definem-se sucessões $\{a_n\}$ e $\{b_n\}$ da seguinte forma:

1. $a_0 = a$ e $b_0 = b$;
2. dados a_n e b_n:
   (a) toma-se $m = (a_n + b_n)/2$;
   (b) se $f(a_n)f(m) < 0$, então $a_{n+1} = a_n$ e $b_{n+1} = m$;

(c) caso contrário, $a_{n+1} = m$ e $b_{n+1} = b_n$;

O método consiste em fixar um valor e para o erro pretendido para a aproximação e calcular valores sucessivos de a_n ou b_n até que $b_n - a_n < e$. Quando esta condição se verifica, a_n (ou b_n) é a aproximação pretendida para a raiz.

Implemente o método da bissecção em F através de uma função `bissec` que receba como argumentos os reais `a`, `b` (extremos do intervalo) e `e` (erro máximo da aproximação) e a função real de variável real `f` e devolva a aproximação encontrada para a raiz da equação `f(x)`=0.

Resolução.

```
function bissec(a,b,e,f) result(r)
real, intent(in) :: a,b,e
interface
  function f(x) result(y)
  real, intent(in) :: x
  real :: y
  end function f
end interface
real :: r,x,y,d

x=a
y=b
d=(b-a)/2
r=x+d
do
  if (d<e) then
    exit
  else
    if (f(x)*f(r)<0) then
      y=r
    else
      x=r
    end if
    d=d/2
    r=x+d
  end if
end do
end function bissec
```