Computação e Programação

(LEAmb, LEMat, LQ, MEBiol, MEQ)

Departamento de Matemática, IST

Exame 1 – 12 de Janeiro 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 ocorrências do mínimo do vector.

Resolução

```
function R=f(V)
R=0;
M=+inf;
I=1;
while (I<=length(V))
  if(V(I)<M)
    M=V(I);
    R=1;
  elseif (V(I)==M)
    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 ocorrências do mínimo da linha k.

Resolução

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

```
        R(I)=R(I)+1;
    end
    J=J+1;
  end
I=I+1;
end
```

Grupo II [2,0+2,0]

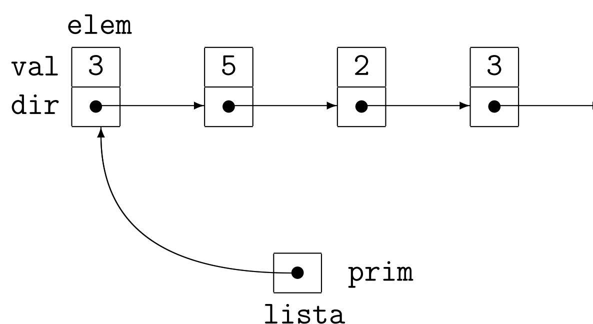
Considere o tipo de dados abstracto *lista de inteiros*.

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 :: prim
end type lista

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

Com esta implementação, pretende-se que uma lista seja representada por uma cadeia de nós. Por exemplo, a lista $[3, 5, 2, 3]$ é representada pela cadeia de nós seguinte:



As operações disponibilizadas pelo módulo são as seguintes:

- `nil()`: função que devolve a lista vazia;
- `primeiro(w,n)`: subrotina que recebe no parâmetro de entrada/saída `w` uma lista de inteiros e devolve no parâmetro de saída `n` o primeiro elemento da lista, caso esta não seja vazia;
- `ultimo(w,n)`: subrotina que recebe no parâmetro de entrada/saída `w` uma lista de inteiros e devolve no parâmetro de saída `n` o último elemento da lista, caso esta não seja vazia;
- `apaga_prim(w)`: subrotina que recebe no parâmetro de entrada/saída `w` uma lista de inteiros e lhe retira o primeiro elemento, caso a lista não seja vazia;
- `apaga_ult(w)`: subrotina que recebe no parâmetro de entrada/saída `w` uma lista de inteiros e lhe retira o último elemento, caso a lista não seja vazia;
- `acrescenta(w,n)`: subrotina que recebe no parâmetro de entrada/saída `w` uma lista de inteiros e no parâmetro de entrada `n` um inteiro e acrescenta `n` ao início da lista `w`;

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

Resolução

```
function nil () result(w)
type(lista) :: w

w%prim=>null()
end function nil

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

if (associated(w%prim)) then
   n=w%prim%val
end if
end subroutine primeiro

subroutine ultimo(w,n)
type(lista), intent(inout) :: w
integer, intent(out) :: n
type(elem), pointer :: a

if (associated(w%prim)) then
   a=>w%prim
   do
      if (.not.associated(a%dir)) then
         n=a%val
         exit
      end if
      a=>a%dir
   end do
end if
end subroutine ultimo

subroutine apaga_prim(w)
type(lista), intent(inout) :: w
type(elem), pointer :: a

if (associated(w%prim)) then
   a=>w%prim
   w%prim=>w%prim%dir
   deallocate(a)
end if
end subroutine apaga_prim

subroutine apaga_ult(w)
```

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

call apagaAux(w%prim)
end subroutine apaga_ult

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

if (associated(p)) then
   if (associated(p%dir)) then
      call apagaAux(p%dir)
   else
      deallocate(p)
   end if
end if
end subroutine apagaAux

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

allocate(a)
a%val=n
a%dir=>w%prim
w%prim=>a
end subroutine acrescenta

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

if (associated(w%prim)) then
   b=.false.
else
   b=.true.
end if
end function vazia
```

b) Desenvolva em F, recorrendo ao módulo acima, uma subrotina `nocc(w,n)` que receba no parâmetro de entrada/saída `w` uma lista de inteiros e devolva no parâmetro de saída `n` o número de ocorrências do máximo de `w`. Se `w` for a lista vazia, a subrotina deverá devolver em `n` o valor `0`.

Resolução

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

```
integer :: x, max

if (.not.vazia(w)) then
  call primeiro(w,max)
  call apaga_prim(w)
  n=1
  do
    if (vazia(w)) then
      exit
    end if
    call primeiro(w,x)
    if (x>max) then
      n=1
      max=x
    elseif (x==max) then
      n=n+1
    end if
    call apaga_prim(w)
  end do
else
  n=0
end if
end subroutine nocc
```

Grupo III [2,0+1,0]

a) Defina recursivamente uma função `comb` que receba como argumentos dois naturais (inteiros não negativos) `n` e `k` tais que `n` $\geq$ `k` e devolva $\binom{n}{k}$. Recorde que estes números satisfazem a relação

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

para $n, k > 0$.

Resolução:

```
recursive function comb(n,k) result(r)
integer, intent(in) :: n,k
integer :: r

if ((n==0).or.(k==0).or.(n==k)) then
  r=1
else
  r=comb(n-1,k-1)+comb(n-1,k)
end if
end function comb
```

b) Seja f uma função contínua num intervalo $[a, b]$ satisfazendo $f(a)f(b) < 0$. Então f tem uma raíz 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 raíz.

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 raíz 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
```