Parte II

Fortran

Capítulo 5

Programação imperativa sobre vectores

1. Defina a função `pertence` que dado um número n e um vector v de números devolve verdadeiro se n pertence a v e falso caso contrário.
2. Defina a função `ocorrencias` que dado um número n e um vector v conta quantas vezes n ocorre em v.
3. Defina a função `pos_max` que dado um vector v devolve a soma dos índices das posições onde o máximo de v ocorre.
4. Defina a função `split` que dado um vector v e um número n devolve um triplo $[a, b, c]$ tal que:
   - a é o número de elementos de v maiores que n,
   - b é o número de elementos de v iguais a n,
   - c é o número de elementos de v menores que n.
5. Defina a função `f` que dado um vector v devolve um par $[x, y]$ tal que
   - x é 1 se os elementos do vector estiverem ordenados por ordem crescente (em sentido lato) e é 0 caso contrário;
   - y é o número de vezes que um elemento é sucedido de outro estritamente maior.
6. Defina a função `f` que dado um vector v devolve um par $[x, y]$ tal que
   - x é 1 se o vector for constante (isto é, se todos os elementos forem iguais) e é 0 caso contrário;
   - y é o número de vezes que um elemento é sucedido de outro diferente.
7. Defina a função `f` que dado um vector v devolve um par $[x, y]$ tal que
   - x é a diferença entre o máximo e o mínimo do vector;
   - y é o número de vezes que o máximo ocorre na lista.
8. Defina a função `f` que dado um vector v devolve um par $[x, y]$ tal que
   - x é a soma das posições onde ocorre o mínimo do vector;
   - y é a última posição em que o mínimo do vector ocorre.
9. Defina a função `f` que dado um vector v devolve um par $[x, y]$ tal que

- x é o menor número primo que ocorre no vector;
- y é o número de números primos que ocorrem no vector.

No caso de não ocorrerem números primos no vector, x deve ser 0.

10. Defina a função `f` que dado um vector v devolve um par $[x, y]$ tal que

    - x é 1 se todos os elementos do vector forem pares e 0 caso contrário;;
    - y é o menor número par.

    No caso de não ocorrerem números pares no vector, y deve ser 0.

11. Defina a função `f` que dado um vector v devolve um par $[x, y]$ tal que

    - x é 1 se no vector ocorrerem mais números pares do que ímpares e 0 caso contrário;
    - y é a diferença entre o número de números pares e o número de números ímpares.

12. Defina a função `car_primos` que dado um vector v devolve um vector de igual comprimento com 1 nas posições onde v contém um número primo e 0 nas restantes.

13. Defina a função `f` que dado um vector v devolve um vector de igual comprimento tal que cada elemento do argumento é substituído:

    - pelo índice da posição que ocupa, caso esse elemento seja um número par;
    - por 0 caso contrário.

14. Defina a função `f` que dado um vector v devolve um vector de igual comprimento tal que cada elemento do argumento é substituído por:

    - 2 se esse elemento for maior do que o elemento na posição anterior;
    - 1 se esse elemento for igual ao elemento na posição anterior;
    - 0 caso contrário.

    O elemento na primeira posição, que não tem elemento na posição anterior, deve ser substituído por 0.

15. Defina a função `f` que dado um vector v devolve um vector de igual comprimento que contém em cada posição o número de números ímpares encontrados até essa posição (inclusive) no vector dado

16. Defina a função `f` que dado um vector v devolve um vector de igual comprimento tal que cada elemento do argumento é substituído:

    - pela soma do elemento com a posição que ocupa, caso o elemento ocorra no vector entre dois números pares;
    - pela sua posição, caso contrário.

17. Defina a função `f` que dado um vector v devolve um vector de igual comprimento com as posições trocadas duas a duas, isto é, os elementos da primeira e da segunda posição trocados entre si, os elementos da terceira e quarta posições trocados entre si, e assim sucessivamente. No caso de o vector ter comprimento ímpar a última posição não se altera.

18. Defina a função `f` que dado um vector v devolve um vector de igual comprimento tal que cada elemento do argumento é substituído:

    - pelo elemento que o antecede, caso o elemento anterior seja inferior à posição que ocupa;
    - por 0 caso contrário, caso contrário.

19. Um vector u diz-se prefixo de um vector v se o comprimento de u for menor ou igual do que o comprimento de v e $u(i) = v(i)$, para todo o $i = 1, \ldots, length(u)$. Defina a função `prefixo` que recebe como argumento dois vectores u e v e devolve verdadeiro se u é prefixo de v e falso caso contrário.

20. Diz-se que um vector tem um *elemento maioritário* se mais de metade dos seus elementos forem iguais. Defina uma função `maioritario` que dado um vector de números inteiros (positivos) e devolve o elemento maioritário do vector, caso exista. No caso do vector não ter elemento maioritário, a função deverá devolver -1.

21. O *reconhecimento de padrões* consiste em verificar se uma determinada sequência padrão `p` ocorre numa outra sequência `s`. Considere que, quer a sequência padrão `p`, quer a sequência `s`, são vectores de inteiros. Pretende-se verificar se os elementos do vector `p` ocorrem no vector `s`, pela mesma ordem e consecutivamente. Apresentam-se em seguida alguns exemplos.

    - O padrão $[1, 2, 3]$ ocorre na sequência $[2, \mathbf{1}, \mathbf{2}, \mathbf{3}, 4, 5]$;
    - O padrão $[4, 3, 4]$ não ocorre na sequência $[2, 1, 4, 2, 3, 4, 5]$;
    - O padrão $[5, 2, 7]$ ocorre na sequência $[\mathbf{5}, \mathbf{2}, \mathbf{7}]$;
    - O padrão $[9, 4]$ ocorre na sequência $[3, 2, 4, 2, \mathbf{9}, \mathbf{4}]$;
    - O padrão $[7, 4, 3]$ ocorre na sequência $[\mathbf{7}, \mathbf{4}, \mathbf{3}, 7, 2, 7]$.

    Defina uma função `reconhece` que recebe como argumento dois vectores `p` e `s` de números inteiros e devolve 1 se o padrão `p` ocorre na sequência `s`, e devolve 0 caso contrário.

Capítulo 6

Programação imperativa sobre matrizes

1. Defina a função `media_matriz` que recebe como argumento uma matriz e devolve a média dos seus elementos.

2. Defina a função `matriz_igual` que recebe como argumento duas matrizes de igual dimensão e devolve verdadeiro se ambas as matrizes são iguais elemento a elemento e falso caso contrário.

3. Defina a função `linha_par` que recebe como argumento um vector de inteiros e devolve verdadeiro se a soma dos elementos é par e falso caso contrário.

   Generalize esta função de modo a receber como argumento uma matriz e a devolver um vector e comprimento igual ao número de linhas da matriz, que contém na i-ésima posição o valor 1 se a linha i satisfizer `linha_par` e 0 caso contrário.

4. Defina a função `l_primos` que recebe como argumento uma matriz de números inteiros e devolve o número de linhas em que ocorre, pelo menos, um número primo. Pode utilizar, no máximo, dois ciclos encaixados.

5. Defina a função `col_min` que recebe como argumento uma matriz de números inteiros e devolve o número de colunas em que ocorre o mínimo da matriz. Pode utilizar, no máximo, dois ciclos encaixados.

6. Defina a função `f` que recebe como argumento uma matriz e devolve um par $[x, y]$ tal que:

   - x é o número de linhas constituídas exclusivamente por números pares;
   - y é o índice da primeira dessas linhas.

   Pode utilizar, no máximo, dois ciclos encaixados. Caso não existam linhas nestas condições, x deve ser 1 e y deve ser 0.

7. Defina a função `f` que recebe como argumento uma matriz quadrada de números inteiros e devolve um par $[x, y]$ tal que:

   - x é a diferença entre a soma dos elementos da diagonal principal da matriz e a soma dos elementos da diagonal oposta;
   - y é a diferença entre a soma dos elementos da triangular superior e a soma dos elementos da triangular inferior.

   Pode utilizar, no máximo, dois ciclos encaixados. Os elementos das triangulares não incluem os elementos da diagonal principal.

8. Defina a função `f` que recebe como argumento uma matriz quadrada de números inteiros e devolve um par $[x, y]$ tal que:

   - x é o número de números pares que se encontram acima das duas diagonais;
   - y é o número de números pares que se encontram abaixo das duas diagonais.

   Pode utilizar, no máximo, dois ciclos encaixados.

9. Defina a função `f` que recebe como argumento uma matriz quadrada de números inteiros e devolve 1 se o número de números pares que acima das diagonais for igual ao número de números impares que ocorre abaixo das diagonais. Pode utilizar, no máximo, dois ciclos encaixados.

10. Defina a função `f` que recebe como argumento uma matriz quadrada de números inteiros e devolve o maior elemento que ocorre entre as duas diagonais. Pode utilizar, no máximo, dois ciclos encaixados. Caso não existam elementos nestas condições, a função deve devolver `0`.

11. Defina a função `f` que recebe como argumento uma matriz de números inteiros e devolve um vector de comprimento igual ao número de linhas que contém na posição k o valor:

    - 2 se na linha k existirem mais números pares do que ímpares;
    - 1 se na linha k existirem mais números ímpares do que pares;
    - 0 se na linha k existirem tantos números pares como ímpares.

12. Defina a função `f` que recebe como argumento uma matriz de números inteiros e devolve um vector de comprimento igual ao número de linhas que contém na posição k a diferença entre o maior número ímpar e o menor número ímpar que ocorrem na linha k. Se não ocorrerem números ímpares considera-se o valor 0.

13. Defina a função `f` que recebe como argumento uma matriz de números inteiros e devolve um vector de comprimento igual ao número de colunas que contém na posição k a soma dos elementos pares da coluna k que ocorrem em linhas ímpares.

14. Defina a função `f` que recebe como argumento uma matriz de números inteiros e devolve um vector de comprimento igual ao número de colunas que contém na posição k o maior elemento da coluna k.

15. Defina a função `f` que recebe como argumento uma matriz de números inteiros e devolve um vector de comprimento igual ao número de linhas que contém na posição k o índice a primeira coluna onde ocorre o maior elemento da linha k.

Capítulo 7

Programação recursiva

Os exercícios desta secção devem ser resolvidos recorrendo a *funções definidas por recursão*.

1. Defina a função `soma_natR` que recebe como argumento um número inteiro positivo n e devolve a soma de todos os naturais até n.

2. Defina a função `divR` que recebe como argumento dois números inteiros positivos m e n e devolve o resultado da divisão inteira de m por n.

3. Defina a função `prim_algR` que recebe como argumento um número inteiro positivo n e devolve o primeiro algarismo na representação de n.

4. Defina a função `num_perfR` que recebe como argumento um número inteiro positivo e que devolve verdadeiro se esse número for um número perfeito e falso caso contrário. Se necessário, recorde a noção de *número perfeito* no Exercício 6 do Capítulo 1.

5. Defina a função `mdcR` que recebe como argumento dois números naturais e devolve o seu máximo divisor comum, calculado através do *algoritmo de Euclides*.

6. Considere a função $f : \mathbb{N} \to \mathbb{N}$ tal que

$$f(x) = \begin{cases} x/2 & \text{se } x \text{ for um número par} \\ 3x+1 & \text{caso contrário} \end{cases}$$

   Defina a função `num_itR` que recebe como argumento um número natural n e devolve o número de vezes que f tem que se aplicada (recursivamente) a n até se atingir o número 1, i.e., devolve o número k tal que

$$\underbrace{f(f(\ldots f(n)))}_{k-\text{vezes}} = 1.$$

7. Defina a função `comb` que recebe como argumentos dois naturais `m` e `q`, com `m` $\geq$ `q`, tal que `comb(m,q)`$= \begin{pmatrix} m \\ q \end{pmatrix}$, as combinações de `m`, `q` a `q`. Recorde que as combinações satisfazem a relação

$$\begin{pmatrix} m \\ q \end{pmatrix} = \begin{pmatrix} m-1 \\ q-1 \end{pmatrix} + \begin{pmatrix} m-1 \\ q \end{pmatrix}$$

   para $0 < q < m$.

8. Defina a função `pertenceR` que recebe como argumento um número n e um vector v e devolve verdadeiro se n pertence a v e falso caso contrário.

9. Defina a função `ocorrenciasR` que dado um número n e um vector v conta quantas vezes n ocorre v.

10. Defina a função `maxR` que recebe como argumento um vector v e devolve o máximo de v.

11. Defina a função `pos_maxR` que recebe como argumento um vector v e devolve a soma dos índices das posições onde o máximo de v ocorre.

12. Defina a função `splitR` que dado um vector v e um número n devolve um triplo $[a, b, c]$ tal que:

    - a é o número de elementos de v maiores que n,
    - b é o número de elementos de v iguais a n,
    - c é o número de elementos de v menores que n.

13. Um vector u diz-se prefixo de um vector v se o comprimento de u for menor ou igual do que o comprimento de v e $u(i) = v(i)$, para todo o $i = 1, \ldots, length(u)$. Defina a função `prefixoR` que recebe como argumento dois vectores u e v e devolve verdadeiro se u é prefixo de v e falso caso contrário.

14. Recorde o *algoritmo de pesquisa binária* para pesquisar um número num vector ordenado. Defina uma função `pesquisa_bin` que recebe como argumento um vector v de números ordenado (por ordem crescente) e um elemento x e que devolve verdadeiro se x pertence a v e falso caso contrário.

15. Defina a função `media_matrizR` que recebe como argumento uma matriz e devolve a média dos seus elementos.

16. Defina a função `max_matrizR` que recebe como argumento uma matriz a e devolve o máximo de a.

Capítulo 8

Tipos de dados abstractos sobre vectores

1. Recorde o módulo `mbstacks` que disponibiliza o tipo de dados *pilha de inteiros*, a seguir definido:

```
type, public :: stack
    private
    integer, dimension(10) :: array
    integer :: index
end type stack
```

   e as operações:

   - `new()`: função sem argumentos que devolve a pilha (`stack`) vazia;
   - `push(n,s)`: subrotina que recebe no parâmetro de entrada `n` (de tipo `integer`) um número inteiro e no parâmetro de entrada/saída `s` (de tipo `stack`) uma pilha e sobrepõe `n` em `s`;
   - `pop(s)`: subrotina que recebe no parâmetro de entrada/saída `s` (de tipo `stack`) uma pilha lhe apaga o elemento no topo, caso a pilha não esteja vazia;
   - `top(s)`: função que recebe no parâmetro `s` (de tipo `stack`) uma pilha e devolve topo de `s`, caso a pilha não esteja vazia;
   - `empty(s)`: função que recebe no parâmetro `s` (de tipo `stack`) uma pilha e devolve `.true.` se a pilha estiver vazia e `.false.` caso contrário;

   (a) Recorrendo ao módulo `mbstacks` desenvolva as seguintes operações sobre o tipo de dados `stack`:
      - `noc(x,s)`: devolve o número de ocorrências do número `x` na pilha `s`;
      - `inverte(s)`: devolve a pilha `s` invertida;
      - `prefixo(s1,s2)`: devolve `.true.` se a pilha `s1` for prefixo da pilha `s2`. Uma pilha `s1` diz-se *prefixo* de `s2` se `s2` é igual a `s1` ou `s2` pode ser obtida de `s1` sobrepondo-lhe alguns elementos.

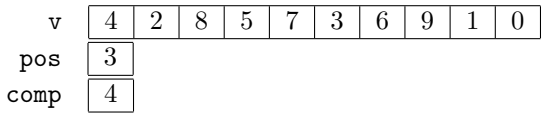
   (b) Enriqueça o módulo `mbstacks` com funções ou subrotinas que implementem as operações anteriores.

2. Considere o tipo de dados *fila de espera* de inteiros, escolhendo a seguinte representação para o tipo de dados `fila`:

```
type, public :: fila
private
```

```
    integer, dimension(:),allocatable :: vector
    integer :: pos, comp
end type fila
```

Com esta representação pretende-se que uma *fila de espera* seja representada da seguinte forma: o campo `vector` é utilizado para guardar (por ordem de chegada) os elementos na fila de espera; o campo `pos` marca a primeira posição ocupada do vector (um número entre 1 e o comprimento do vector, que é fixado no momento de criação da fila); o campo `comp` indica o número de posições do vector que estão ocupadas. Considere-se a configuração descrita na figura. Neste caso, o primeiro elemento da fila de espera encontra-se na posição 3, ou seja, é o número 8, e a fila tem 4 elementos, o que significa que o último elemento da fila de espera é o número 3. Note-se que os valores nas restantes posições são irrelevantes, não fazendo parte do conteúdo da fila. O próximo elemento a chegar à fila de espera irá ocupar a primeira posição livre que, neste caso, é a posição 7.



As operações sobre o tipo de dados fila de espera são as seguir descritas:

- `nova`: operação que recebe o comprimento máximo da fila de espera, que devolve a `fila` vazia com esse comprimento;
- `entra(x,f)`: operação que que acrescenta o elemento `x` ao fim da `fila f`, se houver lugar;
- `sai(f)`: operação que retira o primeiro elemento da `fila f`, caso esta não esteja vazia;
- `primeiro(f)`: operação que devolve o primeiro elemento da `fila f`, caso esta não esteja vazia;
- `vazia(f)`: operação que indica se a `fila f` está ou não vazia;
- `cheia(f)`: operação que indica se a `fila f` está ou não cheia;
- `comprimento(f)`: operação que devolve o comprimento da `fila f`.

(a) Desenvolva em *Fortran* um módulo de disponibilize o tipo de dados *fila de espera*, descrito acima.

(b) Recorrendo ao módulo anterior, desenvolva uma implementação para as seguintes operações:

- `retiraN(n,f)`: retira os primeiros `n` elementos da fila `f` (caso não existam `n` elementos, retira os elementos todos deixando a fila vazia);
- `noc(x,f)`: devolve o número de vezes o número `x` ocorre na fila `f`;
- `junta(f1,f2)`: acrescenta os elementos da fila `f2` ao fim da fila `f1`, mantendo a ordem e deixando a fila `f2` vazia, se houver espaço em `f1`;
- `distribui(f,f1,f2)`: distribui alternadamente os elementos da fila `f` pelas filas `f1` e `f2`, deixando a fila `f` vazia, se houver espaço em `f1` e `f2`;
- `prefixo(f1,f2)`: devolve `.true.` no caso de `f1` ser um prefixo de `f2` e `.false.` no caso contrário;
- `subfila(f1,f2)`: devolve `.true.` no caso de `f1` ser uma *subfila* de `f2` e `.false.` no caso contrário. Uma fila `f1` diz-se *subfila* de `f2` se `f1` é igual a `f2` ou `f2` pode ser obtida de `f1` acrescentando-lhe alguns elementos no início e outros no fim.

3. Uma *pilha navegável* é uma pilha com as operações usuais, mas com uma componente adicional: um *cursor*. Este cursor permite percorrer os elementos da pilha sem contudo alterar o seu conteúdo. O cursor pode ser deslocado para cima e para baixo e, adicionalmente, pode consultar-se o conteúdo da

pilha na posição apontada pelo cursor. Sempre que se adiciona o remove um elemento da pilha, o cursor volta ao topo da pilha. Pretende-se implementar o tipo de dados `xstack` das pilhas navegáveis escolhendo a seguinte representação para o tipo `xstack`:

```
type, public :: xstack
private
    integer, dimension(:), allocatable :: vector
    integer :: topo, curs
end type xstack
```

As operações sobre o tipo de dados pilha navegável são as seguir descritas:

- `new`: operação que recebe a profundidade máxima da pilha, que devolve a pilha vazia com essa profundidade;
- `push(x,s)`: operação que que acrescenta o elemento `x` ao topo da pilha `s`, se houver lugar;
- `pop(s)`: operação que retira o elemento no topo da pilha `s`, caso esta não esteja vazia;
- `top(s)`: operação que devolve o elemento no topo da pilha `s`, caso esta não esteja vazia;
- `empty(s)`: operação que indica se a ilha `s` está ou não vazia;
- `full(s)`: operação que indica se a pilha `s` está ou não cheia;
- `up(s)`: operação que desloca o cursor da pilha `s` uma posição para cima, caso tal seja possível;
- `down(s)`: operação que desloca o cursor da pilha `s` uma posição para baixo, caso tal seja possível;
- `curr(s)`: operação que devolve o elemento dea pilha `s` apontado pelo cursor, caso tal a pilha não esteja vazia;
- `at_top(s)`: operação que devolve `.true.` caso o cursor esteja no topo da pilha `s` e `.false.` caso contrário;
- `at_bot(s)`: operação que devolve `.true.` caso o cursor esteja no fundo da pilha `s` e `.false.` caso contrário.

(a) Desenvolva em *Fortran* um módulo de disponibilize o tipo de dados *pilha navegável*, descrito acima.

(b) Recorrendo ao módulo anterior, desenvolva uma implementação para as seguintes operações:

- `noc(x,s)`: devolve o número de ocorrências do número `x` na pilha navegável `s`;
- `prefixo(s1,s2)`: devolve `.true.` se a pilha navegável `s1` for prefixo da pilha navegável `s2`. Uma pilha navegável `s1` diz-se *prefixo* de `s2` se `s2` é igual a `s1` ou `s2` pode ser obtida de `s1` sobrepondo-lhe alguns elementos. Esta operação não deve alterar o conteúdo das pilhas `s1` e `s2`, podendo no entanto alterar as posições dos cursores.

Capítulo 9

Tipos de dados abstractos sobre apontadores

1. Recorde o módulo `mustacks` que disponibiliza o tipo de dados *pilha de inteiros*, a seguir definido:

```
type, public :: stack
    private
    type(node), pointer :: top
    integer :: dep
end type stack

type, private :: node
    integer :: val
    type(node), pointer :: next
end type node
```

   e as operações:

   - `new()`: função sem argumentos que devolve a pilha (`stack`) vazia;
   - `push(n,s)`: subrotina que recebe no parâmetro de entrada `n` (de tipo `integer`) um número inteiro e no parâmetro de entrada/saída `s` (de tipo `stack`) uma pilha e sobrepõe `n` em `s`;
   - `pop(s)`: subrotina que recebe no parâmetro de entrada/saída `s` (de tipo `stack`) uma pilha lhe apaga o elemento no topo, caso a pilha não esteja vazia;
   - `top(s)`: função que recebe no parâmetro `s` (de tipo `stack`) uma pilha e devolve topo de `s`, caso a pilha não esteja vazia;
   - `empty(s)`: função que recebe no parâmetro `s` (de tipo `stack`) uma pilha e devolve `.true.` se a pilha estiver vazia e `.false.` caso contrário;

   (a) Recorrendo ao módulo `mustacks` desenvolva as seguintes operações sobre o tipo de dados `stack`:
   - `noc(x,s)`: devolve o número de ocorrências do número `x` na pilha `s`;
   - `inverte(s)`: devolve a pilha `s` invertida;
   - `prefixo(s1,s2)`: devolve `.true.` se a pilha `s1` for prefixo da pilha `s2`. Uma pilha `s1` diz-se *prefixo* de `s2` se `s2` é igual a `s1` ou `s2` pode ser obtida de `s1` sobrepondo-lhe alguns elementos.

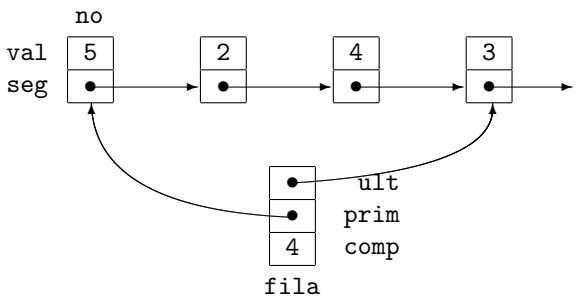
   (b) Enriqueça o módulo `mustacks` com funções ou subrotinas que implementem as operações anteriores.

2. Considere o tipo de dados *fila de espera* de inteiros, escolhendo a seguinte representação para o tipo de dados `fila`:

```
type, public :: fila
    private
    type(no), pointer :: prim,ult
    integer :: comp
end type fila

type, private :: no
    integer :: val
    type(no), pointer :: seg
end type no
```

Com esta implementação pretende-se que uma *fila de espera* com 4 elementos, contento 5 na primeira posição, seguido de 2, 4 e 3, seja representada da seguinte forma



As operações sobre o tipo de dados fila de espera são as seguir descritas:

- `nova`: operação que devolve a `fila` vazia;
- `entra(x,f)`: operação que que acrescenta o elemento `x` ao fim da `fila f`;
- `sai(f)`: operação que retira o primeiro elemento da `fila f`, caso esta não esteja vazia;
- `primeiro(f)`: operação que devolve o primeiro elemento da `fila f`, caso esta não esteja vazia;
- `vazia(f)`: operação que indica se a `fila f` está ou não vazia;
- `comprimento(f)`: operação que devolve o comprimento da `fila f`.

(a) Desenvolva em *Fortran* um módulo de disponibilize o tipo de dados *fila de espera*, descrito acima.

(b) Recorrendo ao módulo anterior, desenvolva uma implementação para as seguintes operações:

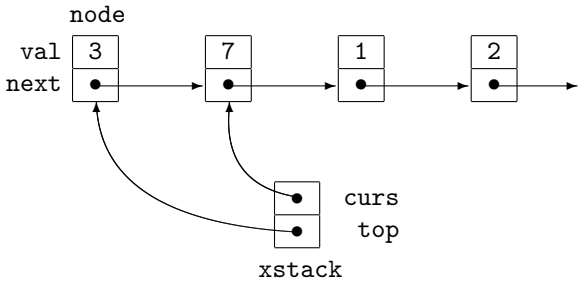
- `retiraN(n,f)`: retira os primeiros `n` elementos da fila `f` (caso não existam `n` elementos, retira os elementos todos deixando a fila vazia);
- `noc(x,f)`: devolve o número de vezes o número `x` ocorre na fila `f`, podendo a fila `f` ficar vazia;
- `junta(f1,f2)`: acrescenta os elementos da fila `f2` ao fim da fila `f1`, mantendo a ordem e deixando a fila `f2` vazia;
- `distribui(f,f1,f2)`: distribui alternadamente os elementos da fila `f` pelas filas `f1` e `f2`, deixando a fila `f` vazia;
- `duplica(f,f1,f2)`: copia o conteúdo da fila `f` para as filas `f1` e `f2`, deixando a fila `f` vazia.

3. Recorde o tipo de dados *pilha navegável*, definido no Capítulo 8. Pretende-se implementar o tipo de dados `xstack` das pilhas navegáveis escolhendo a seguinte representação para o tipo `xstack`:

```
type, public :: xstack
    private
    type(node), pointer :: top, curs
end type xstack

type, private :: node
    integer :: val
    type(no), pointer :: next
end type node
```

Com esta implementação pretende-se que uma *pilha navegável* com 4 elementos, com o elemento 3 no topo, e contendo abaixo os elementos 7, 1 e 2, seja representada da seguinte forma



Na figura anterior, o cursor está, de momento, a apontar para o segundo elemento da pilha, a contar do topo. As operações sobre o tipo de dados *pilha navegável* são as seguir descritas:

- `new`: operação que recebe a profundidade máxima da pilha, que devolve a pilha vazia com essa profundidade;
- `push(x,s)`: operação que que acrescenta o elemento `x` ao topo da pilha `s`, se houver lugar;
- `pop(s)`: operação que retira o elemento no topo da pilha `s`, caso esta não esteja vazia;
- `top(s)`: operação que devolve o elemento no topo da pilha `s`, caso esta não esteja vazia;
- `empty(s)`: operação que indica se a ilha `s` está ou não vazia;
- `up(s)`: operação que desloca o cursor da pilha `s` uma posição para cima, caso tal seja possível;
- `down(s)`: operação que desloca o cursor da pilha `s` uma posição para baixo, caso tal seja possível;
- `curr(s)`: operação que devolve o elemento dea pilha `s` apontado pelo cursor, caso tal a pilha não esteja vazia;
- `at_top(s)`: operação que devolve `.true.` caso o cursor esteja no topo da pilha `s` e `.false.` caso contrário;
- `at_bot(s)`: operação que devolve `.true.` caso o cursor esteja no fundo da pilha `s` e `.false.` caso contrário.

(a) Desenvolva em *Fortran* um módulo de disponibilize o tipo de dados *pilha navegável*, descrito acima.

(b) Recorrendo ao módulo anterior, desenvolva uma implementação para as seguintes operações:

- `noc(x,s)`: devolve o número de ocorrências do número `x` na pilha navegável `s`;

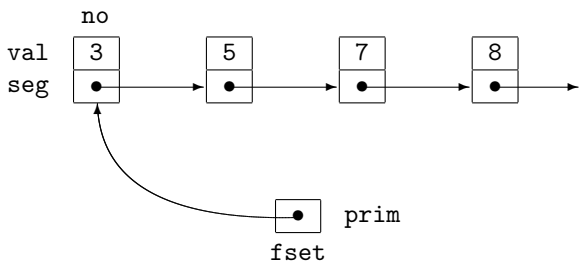
- `prefixo(s1,s2)`: devolve `.true.` se a pilha navegável `s1` for prefixo da pilha navegável `s2`. Uma pilha navegável `s1` diz-se *prefixo* de `s2` se `s2` é igual a `s1` ou `s2` pode ser obtida de `s1` sobrepondo-lhe alguns elementos. Esta operação não deve alterar o conteúdo das pilhas `s1` e `s2`, podendo no entanto alterar as posições dos cursores.

4. Considere o tipo de dados *conjunto finito* de inteiros, escolhendo a seguinte representação para o tipo de dados `fset`:

```
type, public :: fset
   private
   type(no), pointer :: prim
end type fset

type, private :: no
   integer :: val
   type(no), pointer :: seg
end type no
```

Com esta implementação pretende-se que o *conjunto finito* $\{3, 5, 7, 8\}$ seja representado da seguinte forma



Note que embora não seja obrigatório, recomenda-se que os elementos sejam mantidos ordenados. As operações sobre o tipo de dados *conjunto finito* são as seguir descritas:

- `emptyset()`: função sem parâmetros que devolve o conjunto vazio;
- `insert(x,s)`: subrotina que recebe no parâmetro de entrada `x` (de tipo `integer`) um elemento e no parâmetro de entrada/saída `s` (de tipo `fset`) um conjunto e acrescenta `x` a `s`;
- `remove(x,s)`: subrotina que recebe no parâmetro de entrada `x` (de tipo `integer`) um elemento e no parâmetro de entrada/saída `s` (de tipo `fset`) um conjunto e retira `x` a `s`;
- `emptyQ(s)`: função que recebe no parâmetro `s` de tipo `fset` um conjunto e devolve `.true.` se `s` for o conjunto vazio e `.false.` caso contrário;
- `memberQ(x,s,b)`: subrotina que recebe no parâmetro de entrada `x` (de tipo `integer`) um elemento e no parâmetro de entrada/saída `s` (de tipo `fset`) um conjunto e devolve no parâmetro de saída `b` (de tipo `logical`) o valor `.true.` se `x` for um elemento de `s` e `.false.` caso contrário;
- `union(s,r,u)`: subrotina que recebe nos parâmetros de entrada/saída `s` e `r` (de tipo `fset`) dois conjunto e devolve no parâmetro de saída `u` (de tipo `fset`) a união de `s` e `r`;
- `intersect(s,r,u)`: subrotina que recebe nos parâmetros de entrada/saída `s` e `r` (de tipo `fset`) dois conjunto e devolve no parâmetro de saída `u` (de tipo `fset`) a intersecção de `s` e `r`;

- `diff(s,r,u)`: subrotina que recebe nos parâmetros de entrada/saída `s` e `r` (de tipo `fset`) dois conjunto e devolve no parâmetro de saída `u` (de tipo `fset`) a diferença entre `s` e `r`;, isto é, o conjunto dos elementos que estão em `s` e não estão em `r`;
- `min(s,x)`: subrotina que recebe no parâmetro de entrada/saída `s` (de tipo `fset`) um conjunto e devolve no parâmetro de saída `x` (de tipo `integer`) o menor elemento de `s`, caso `s` não seja vazio;
- `max(s,n)`: subrotina que recebe no parâmetro de entrada/saída `s` (de tipo `fset`) um conjunto e devolve no parâmetro de saída `x` (de tipo `integer`) o maior elemento de `s`, caso `s` não seja vazio;
- `nelem(s,k)`: subrotina que recebe no parâmetro de entrada/saída `s` (de tipo `fset`) um conjunto e devolve no parâmetro de saída `k` (de tipo `integer`) o número de elementos de `s`.

(a) Desenvolva em *Fortran* um módulo de disponibilize o tipo de dados *conjunto finito*, descrito acima.

(b) Recorrendo ao módulo anterior, desenvolva uma implementação para as seguintes operações:
   - `subsetQ(s,r,b)`: subrotina que recebe nos parâmetros de entrada/saída `s` e `r` (de tipo `fset`) dois conjunto e devolve no parâmetro de saída `b` (de tipo `logical`) o valor `.true.` se `s` for subconjunto de `r` e devolve `.false.` caso contrário;
   - `supersetQ(s,r,b)`: subrotina que recebe nos parâmetros de entrada/saída `s` e `r` (de tipo `fset`) dois conjunto e devolve no parâmetro de saída `b` (de tipo `logical`) o valor `.true.` se `s` for sobreconjunto de `r` e devolve `.false.` caso contrário.

(c) Defina as operações `union`, `intersect` e `diff` assumindo que estas não são disponibilizadas pelo módulo anterior.