Aula Prática 7

Resumo:
- Funções e estruturas de dados recursivas.

Exercício 7.1

Escreva um programa `FilterLines` que processe um ficheiro de texto e imprima primeiro as linhas com menos de 20 carateres, depois as linhas com 20 a 40 carateres e finalmente as linhas mais longas. Para cumprir o objectivo, o programa poderá ler e armazenar as linhas de texto, divididas em três conjuntos distintos, imprimindo-os no fim. Como não se conhece a priori o número de linhas existentes no ficheiro, sugere-se a utilização de três listas para guardar as frases separadamente. Use para esse efeito a classe `LinkedList` do pacote `p2utils` em anexo.

Exercício 7.2

A classe `p2utils.LinkedList` define o tipo *lista ligada* com os métodos essenciais para inserção e remoção de elementos. Analise os métodos `print()` e `contains()`. Repare que ambos recorrem a funções auxiliares privadas e recursivas, que usam um parâmetro extra de tipo `Node`.

Acrescente à classe os seguintes métodos, implementado-os também de forma recursiva.

- `clone()` - devolve uma nova lista com a mesma sequência de elementos.
- `reverse()` - devolve uma nova lista com os mesmos elementos por ordem inversa.
- `get(pos)` - devolve o elemento na posição `pos` da lista, em que `pos` varia entre `0` e `size()-1`.
- `concatenate(lst)` - devolve uma nova lista com os elementos da lista (em que o método é chamado) seguidos dos elementos da lista dada no argumento.
- `remove(e)` - remove da lista a primeira ocorrência do elemento dado no argumento.

Teste a classe com o programa `TestList1`.

Exercício 7.3

No exercício 6.5 fez uma função para listar recursivamente o conteúdo de um diretório.

No programa `ListRec2` pretende-se fazer o mesmo, mas agora recorrendo a uma função `recListFiles` que, em vez de imprimir a lista, deve devolver o resultado numa lista ligada de elementos de tipo `File`.

Exercício 7.4

Considere agora um problema parecido com o do exercício 7.1. Neste caso, o programa deve imprimir primeiro as linhas com menos de 20 carateres, por ordem inversa daquela em que estavam no ficheiro, seguidas das linhas mais longas, pela ordem original. Note que pode resolver o problema com apenas uma lista, se inserir elementos pelas duas extremidades.

Exercício 7.5

O programa `TestDArray` demonstra a utilização de um tipo de dados que funciona como um array, mas sem limitação de capacidade. Complete a classe `p2utils.DynamicArray`, que implementa esse tipo de dados. Considere que:

- A classe é genérica. O parâmetro de tipo é usado para indicar o tipo dos elementos a armazenar no array dinâmico.
- O campo `array` servirá para guardar os elementos introduzidos. O construtor deve criar um array com uma dimensão inicial nula.
- A operação `a.set(n, v)` serve para armazenar o valor `v` na posição `n` do array dinâmico `a` (análogo a `a[n] = v`). O indíce `n` não pode ser negativo, mas pode ser maior ou igual a `a.length()`. Quando isso acontece, o array tem de ser redimensionado para ter um tamanho maior que o índice. O tamanho deverá aumentar sempre para um múltiplo da constante `BLOCK`.
- A operação `v = a.get(n)` deve devolver o valor que tiver sido armazenado na posição `n` de `a` (análogo a `v = a[n]`, se `a` fosse um array nativo). Se nenhum valor tiver sido armazenado nessa posição, deve devolver `null`.
- A operação `v = a.get(n, d)` só difere da anterior por devolver o valor `d` em vez de `null`.

Experimente compilar o programa de teste. Para o executar, tem de fornecer um texto pelo dispositivo de entrada, terminando com a marca de fim-de-ficheiro (`Ctrl+D` em Unix), ou usar redirecionamento de *input*, por exemplo fazendo:
`java -ea TestDArray < /etc/dictionaries-common/words`.