Linguagens e Ambientes de Programação (2018/2019)

Prática 7

Estruturas ligadas em C. Exercícios de 37 a 38.

- 37 - Copie o módulo fechado `LinkedList` da aula teórica 13, estude-o, e acrescente-lhe as seguintes operações, todas elas para programar sem usar recursividade mas usando iteração. A complexidade temporal das quatro primeiras funções deve ser linear (ou seja, não use a função `ListPutAtEnd` das aulas teóricas, que gera soluções quadráticas).

Nestas funções vai sentir necessidade de usar todas as técnicas estudadas na aula teórica, nomeadamente a *técnica do apontador para o último nó* (que permite ir fazendo crescer a lista no final), e a *técnica do apontador atrasado* (que simplifica o código das funções que precisam de adicionar ou eliminar nós no interior da lista). [Saiba também que estas duas técnicas não são suficientes para resolver todos os problemas: problemas novos podem requerer a invenção de novas técnicas.]

Agora uma recomendação importante: na programação destas funções, escreva primeiro o ciclo interno e só depois pense na inicialização do ciclo e na finalização do ciclo.

- `List listClone(List l)` ; // Cria uma cópia duma lista, em que todos os nós são novos.
 - `List listAppend(List l1, List l2)` ; // No final de l1, acrescenta uma cópia da lista l2 (chamando `listClone`, claro)
 - `List listRev(List l)` ; // Inverte uma lista, duplicando todos os nós da lista original. [Repare que fica mais simples do que a função `listClone`, pois é mais simples ir acrescentando na cabeça do que no final.]
 - `List listRevInPlace(List l)` ; // Inverte uma lista, aproveitando os nós da lista original. Só se alteram os apontadores `next`
 - `List listUniq(List l)` ; // Elimina duma lista, todos os nós com valores repetidos. Mantenha a ordem dos elementos que ficam. No caso dos elementos repetidos, pode manter o nó que desejar. Use a operação `free`. Neste problema, espera-se que a complexidade da solução seja quadrática.
-
-

- 38 - A biblioteca standard do C oferece diversas operações úteis e interessantes: uma operação para aceder ao relógio do sistema **time**; um gerador de números aleatórios com operações **srand** e **rand**; uma operação de ordenação genérica **qsort**.

- a) Use as *man pages* do Linux para obter informação sobre estas operações;
 - `man 3 time`
 - `man 3 rand`
 - `man 3 qsort`
- b) Escreva um pequeno programa que faça o seguinte: definir um vetor de inteiros com 30 componentes; inicializar o gerador de números aleatórios usando o tempo do sistema; preencher o

vetor com valores inteiros aleatórios; ordenar crescentemente o vetor usando a função de biblioteca **qsort**; mostrar na consola o vetor ordenado. É já oferecido este ponto de partida:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#define N_ELEMS    30

void fill(int vect[], int n) {
    // FAZER
}

void sort(int vect[], int n) {
    // FAZER
}

void show(int vect[], int n) {
    int i;
    printf("-----\n");
    for( i = 0 ; i < n ; i++ )
        printf("%12d\n", vect[i]);
    printf("-----\n");
}

int main() {
    int vect[N_ELEMS];
    fill(vect, N_ELEMS);
    show(vect, N_ELEMS);
    sort(vect, N_ELEMS);
    show(vect, N_ELEMS);
    return 0 ;
}
```

Se tiver dúvidas sobre a passagem de vetores para funções, leia na teórica 12 a secção "Passagem de parâmetros para funções".

-
- 39 - Inspire-se no módulo fechado `LinkedList` das aula teórica 13 para programar em C um novo módulo fechado de árvores binárias de inteiros chamado `BinTree`. Implemente nele as funções `newTreeNode`, `treeSize`, mais as principais funções da aula prática 3

As operações que geram árvores novas em OCaml, devem alterar diretamente as árvores passadas por argumento (ou seja, devem ser operações "destrutivas"). Como se trata de árvores, faz sentido usar recursividade em muitas das situações.