Ficha 7

Programação Imperativa

Listas Ligadas

1. Considere a seguinte definição de um tipo para representar listas ligadas de inteiros.

```
typedef struct slist {
    int valor;
    struct slist * prox;
} Nodo, *LInt;
```

   (a) Apresente uma sequência de instruções que coloque na variável `a` do tipo `LInt`, uma lista com 3 elementos: 10, 5 e 15 (por esta ordem).

   (b) Apresente definições (preferencialmente não recursivas) das seguintes funções sobre listas ligadas:

      i. `LInt cons (LInt l, int x)` que acrescenta um elemento no inicio da lista.

      ii. `LInt tail (LInt l)` que remove o primeiro elemento de uma lista não vazia (libertando o correspondente espaço).

      iii. `LInt init (LInt l)` que remove o último elemento de uma lista não vazia (libertando o correspondente espaço).

      iv. `LInt snoc (LInt l, int x)` que acrescenta um elemento no fim da lista.

      v. `LInt concat (LInt a, LInt b)` que acrescenta a lista `b` a `a`, retornando o início da lista resultante).

2. Para gerir a informação sobre os alunos inscritos a uma dada disciplina, é necessário armazenar os seguintes dados:

   - Nome do aluno (string com no máximo 60 caracteres)
   - Número do aluno
   - Nota

   (a) Defina os tipos `Aluno` e `Turma`. Para o efeito considere que a informação referente aos alunos de uma turma é armazenada numa lista ligada de alunos.

   (b) Defina uma função `int acrescentaAluno (Turma *t, Aluno a)` que acrescenta a informação de um dado aluno a uma turma. A função deverá retornar `0` se a operação for feita com sucesso.

   (c) Defina uma função `Aluno *procura (Turma t, int numero)` que procura o aluno com um dado número na turma. A função deve retornar `NULL` se a informação desse aluno não existir; caso exista deve retornar o endereço onde essa informação se encontra.

   (d) Defina uma função que determine quantos alunos obtiveram aproveitamento à disciplina (nota final maior ou igual a 10).

3. O tipo `LInt` definido acima pode ser usado para implementar stacks de inteiros (a inserção faz-se no início da lista).

```
typedef LInt Stack;
```

Apresente definições das funções ao lado (já referidas em fichas anteriores)

- `void initStack (Stack *s)`
- `int isEmptyS (Stack *s)`
- `int push (Stack *s, int x)`
- `int pop (Stack *s, int *x)`
- `int top (Stack *s, int *x)`

4. Podemos ainda usar listas ligadas para implementar queues. De forma a garantir a eficiência das várias operações guardam-se os elementos ligados do primeiro para o último e guarda-se também o endereço do último elemento armazenado. A inserção faz-se no fim da lista e a remoção no início.

```
typedef struct queue {
     LInt front, last;
} Queue;
```

Apresente definições das funções (já referidas em fichas anteriores) ao lado.

- `void initQueue (Queue *q)`
- `int isEmptyQ (Queue *q)`
- `int enqueue (Queue *q, int x)`
- `int dequeue (Queue *q, int *x)`
- `int front (Queue *q, int *x)`

5. Uma variante de listas ligadas, conhecida como listas duplamente ligadas, consiste em guardar em cada nodo, além do endereço do próximo, o endereço do anterior.
Apresente definições das seguintes funções sobre listas duplamente ligadas.

```
typedef struct dlist *DLInt;
typedef struct dlist {
    int valor;
    DLInt ant, prox;
} NodoD;
```

- `void inicio (DLInt *l)` que coloca em *l o nodo mais à esquerda da lista.
- `void fim (DLInt *l)` que coloca em *l o elemento mais à direita da lista.
- `void concat (DLInt *a, DLInt b)` que concatena as duas listas.
- `LInt toLInt (DLInt l)` que constrói uma lista ligada com os mesmos elementos (e pela mesma ordem) de uma lista duplamente ligada.
- `DLInt fromLInt (LInt l)` que constrói uma lista duplamente ligada com os mesmos elementos (e pela mesma ordem) de uma lista ligada.