Ficha 8

Programação Imperativa

Buffers implementados em lista

Use, se achar necessário, o projecto https://codeboard.io/projects/240896 para responder aos problemas propostos.

Relembre o tipo LInt usado para representar listas ligadas de inteiros.

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

LInt newLInt (int x, LInt xs) {
    LInt r = malloc (sizeof(struct slist));
    if (r!=NULL) {
        r->valor = x; r->prox = xs;
    }
    return r;
}
```

Este tipo pode ser usado na implementação de Stacks e Queues.

• No caso das stacks, os elementos são inseridos e removidos do início da lista.

```
typedef LInt Stack;
```

• No caso das queues, e de forma a tornar as operações de adição/remoção de um elemento eficientes (e com uma eficácia que não depende do tamanho da queue), é costume usar dois endereços: um para cada uma das extremidades da lista. A inserção é feita no final da lista e a remoção do início.

```
typedef struct {
  LInt inicio,fim;
} Queue;
```

- Uma alternativa a esta última abordagem é implementar as queues com uma lista circular, guardando o endereço do último elemento da lista (que, como a lista é circular, aponta para o primeiro).
- 1. Apresente definições das funções habituais sobre Stacks:

```
• void initStack (Stack *s)
```

- int SisEmpty (Stack s)
- int push (Stack *s, int x)

```
int pop (Stack *s, int *x)int top (Stack s, int *x)
```

2. Apresente definições das funções habituais sobre Queues:

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

3. Apresente definições alternativas destas funções usando uma lista circular tal como referido acima.

```
typedef LInt QueueC;
```

Uma outra instância de buffers consiste em generalizar o conceito de queue, mas permitindo que as operações de inserção e remoção de um elemento possas ser feitas em ambas as extermidades da fila. Esta estrutura é muitas vezes referidas como $double\ ended\ queue$ e abreviada como $(deque^1)$.

Uma implementação deste tipo que permite obter definições eficientes destas funções, consiste em, generalizando a implementação de queues com listas ligadas, usar *listas duplamente ligadas*, i.e., listas em que cada célula contêm ainda o endereço da célula anterior.

```
typedef struct dlist {
      int valor;
      struct dlist *ant, *prox;
} *DList;
typedef struct {
      DList back, front;
} Deque;
```

- 4. Apresente definições das funções habituais sobre Deques. Implemente ainda uma função não standard deste tipo de dados que corresponde a remover o maior dos elementos armazenados (popMax),
 - void initDeque (Deque *q)
 int DisEmpty (Deque q)
 int pushBack (Deque *q, int x)
 int pushFront (Deque *q, int x)
 int popBack (Deque *q, int *x)
 int popFront (Deque *q, int *x)
 int popMax (Deque *q, int *x)
 int back (Deque q, int *x)
 int front (Deque q, int *x)
- 5. Tal como acima, em vez de se usarem dois endereços de uma lista, pode-se usar uma lista (duplamente ligada) circular e então, basta usar o endereço da última célula (que por sua vez, por se tratar de uma lista circular, contem o endereço da primeira).

Apresente definições alternativas destas funções usando esta solução.

 $^{^1}$ https://en.wikipedia.org/wiki/Double-ended_queue