Ficha 6

Programação Imperativa

Buffers

Um `buffer` é um tipo abstracto de dados para armazenar items e que tem duas operações fundamentais:

- **acrescentar** um dado elemento
- **retirar** o próximo elemento

As várias variantes de buffers vão diferir fundamentalmente na definiçõa daquilo que é *o próximo elemento a ser retirado*.
Além disso é costume estarem ainda disponíveis operações para

- inicializar um buffer como vazio
- testar se o buffer está vazio
- saber qual o próximo elemento que será retirado

Neste conjunto de exercícios iremos estudar duas variantes de buffers:

Stack (ou pilha) que são buffers em que o próximo elemento a ser retirado é o último que foi acrescentado (*Last In First Out*). As operações de acrescentar/retirar são normalmente referidas como `push`/`pop`.

Queue (ou fila) que são buffers em que o próximo elemento a ser retirado é o que foi acrescentado há mais tempo (*First In First Out*). As operações de acrescentar/retirar são normalmente referidas como `enqueue`/`dequeue`.

Use, se achar necessário, o projecto `https://codeboard.io/projects/155512` para responder aos problemas propostos.

1. Considere o seguinte tipo para representar *stacks* de números inteiros.

   ```
   struct staticStack {
     int sp;
     int values [Max];
   } STACK, *SStack;
   ```

   Defina as seguintes funções sobre este tipo:

   (a) `void SinitStack (SStack s)` que inicializa uma stack (passa a representar uma stack vazia)

   (b) `int SisEmpty (SStack s)` que testa se uma stack é vazia

   (c) `int Spush (SStack s, int x)` que acrescenta `x` ao topo de `s`; a função deve retornar `0` se a operação fôr feita com sucesso (i.e., se a stack ainda não estiver cheia) e `1` se a operação não fôr possível (i.e., se a stack estiver cheia).

   (d) `int Spop (SStack s, int *x)` que remove de uma stack o elemento que está no topo. A função deverá colocar no endereço `x` o elemento removido. A função deverá retornar `0` se a operação for possível (i.e. a stack não está vazia) e `1` em caso de erro (stack vazia).

(e) `int Stop (SStack s, int *x)` que coloca no endereço `x` o elemento que está no topo da stack (sem modificar a stack). A função deverá retornar `0` se a operação for possível (i.e. a stack não está vazia) e `1` em caso de erro (stack vazia).

2. Considere o seguinte tipo para representar *queues* de números inteiros.

```
struct staticQueue {
  int front;
  int length;
  int values [Max];
} QUEUE, *SQueue;
```

Defina as seguintes funções sobre este tipo:

(a) `void SinitQueue (SQueue q)` que inicializa uma queue (passa a representar uma queue vazia)

(b) `int SisEmptyQ (SQueue q)` que testa se uma queue é vazia

(c) `int Senqueue (SQueue q, int x)` que acrescenta `x` ao fim de `q`; a função deve retornar `0` se a operação fôr feita com sucesso (i.e., se a queue ainda não estiver cheia) e `1` se a operação não fôr possível (i.e., se a queue estiver cheia).

(d) `int Sdequeue (SQueue q, int *x)` que remove de uma queue o elemento que está no início. A função deverá colocar no endereço `x` o elemento removido. A função deverá retornar `0` se a operação for possível (i.e. a queue não está vazia) e `1` em caso de erro (queue vazia).

(e) `int Sfront (SQueue q, int *x)` que coloca no endereço `x` o elemento que está no início da queue (sem modificar a queue). A função deverá retornar `0` se a operação for possível (i.e. a queue não está vazia) e `1` em caso de erro (queue vazia).

3. Na representação de stacks e queues sugeridas nas alíneas anteriores o array de valores tem um tamanho fixo (definido pela constante `MAX`). Uma consequência dessa definição é o facto de as funções de inserção (`push` e `enqueue`) poderem não ser executadas por se ter excedido a capacidade da estruturas.

Uma definição alternativa consiste em não ter um array com tamanho fixo e sempre que seja preciso mais espaço, realocar o array para um de tamanho superior (normalmente duplica-se o tamanho do array).

Considere então as seguintes definições alternativas e adapte as funções definidas atrás para esta nova representação.

Use as funções `malloc` e `free` cujo tipo está definido em `stdlib.h`.

```
typedef struct dinStack {
  int size; // guarda o tamanho do array values
  int sp;
  int *values;
} *DStack;

typedef struct dinQueue {
  int size; // guarda o tamanho do array values
  int front;
  int length;
  int *values;
} *DQueue;
```