Aula prática 5

A aula prática 5 tem como objetivo a aplicação dos conhecimentos relativos a pilhas e filas.

1 - Uma possível aplicação de pilhas é a conversão entre bases numéricas. Pretende-se a implementação de um programa que faça a conversão entre números representados no sistema de numeração decimal e números representados no sistema de numeração binário, e vice-versa.

Para isso, deverá utilizar a implementação de uma pilha de inteiros, uma biblioteca de conversão entre bases numéricas, e também as aplicações de conversão, disponibilizadas na pasta **`ex1`** do ficheiro **`P05.zip`**, disponibilizado no Moodle.

a) Analise a estrutura de dados representada no ficheiro **`pilha.h`**, assim como o restante código fonte fornecido. No ficheiro **`conversor.c`**, implemente a função de conversão de números representados no sistema de numeração decimal para números representados no sistema de numeração binário (**`dec2bin`**).

A função deverá colocar na pilha `p` os bits que compõem a representação no sistema de numeração binário do número `n`, começando pelo bit menos significativo (*least significant bit - LSB*), que ficará no fundo da pilha, até atingir o bit mais significativo *(most significant bit - MSB)*, que deverá ficar no topo da pilha.

Protótipo:	`int dec2bin(pilha *p, int n);`
Argumentos:	**`pilha *p`** apontador para a pilha **`int n`** inteiro (decimal) a converter (binário)
Retorno:	Retorna 1 se convertido com sucesso, 0 caso contrário (por exemplo, caso o número `n` seja negativo).

Teste a função que desenvolveu com o programa **`dec2bin`**.

Exemplos

```
Introduza o número decimal: 42
Representação binária em 6 bits: 1 0 1 0 1 0

Introduza o número decimal: 2022
Representação binária em 11 bits: 1 1 1 1 1 1 0 0 1 1 0
```

b) No ficheiro **conversor.c**, implemente a função de conversão de números representados no sistema de numeração binário para números representados no sistema de numeração decimal (**`bin2dec`**).

Assuma que na pilha consta o número em binário a converter (não negativo), onde o bit mais significativo *(most significant bit - MSB)* está localizado no topo da pilha. A função deverá retornar o número inteiro na sua representação decimal em caso de sucesso ou -1 caso contrário.

Protótipo:	`int bin2dec(pilha *p);`
Argumentos:	**`pilha *p`** apontador para a pilha
Retorno:	Número (em decimal), se converteu com sucesso, -1 caso contrario

Teste a função que desenvolveu com o programa **`bin2dec`**.

Exemplos

```
Introduza o número binário: 1010
Representação decimal: 10

Introduza o número binário: 10110111
Representação decimal: 183
```

2 – O armazém de um porto marítimo guarda contentores à espera de serem carregados em navios que os irão transportar aos seus destinos.

Os contentores que chegam para armazenamento são empilhados por ordem de chegada, o mais próximo possível do local de saída. Quando uma pilha de contentores atingiu a sua capacidade máxima e um novo contentor é recebido, deve iniciar-se uma nova pilha de contentores, colocada atrás (isto é, mais longe da saída) das pilhas existentes até então.

Quando os contentores são retirados para os navios, começa-se por retirar o contentor mais acima da pilha mais próxima da saída. Considere que quando a pilha da frente fica vazia, um tapete rolante automático aproxima as pilhas restantes da saída (isto é, caso existam contentores no armazém, a primeira posição da fila está sempre ocupada).

Considere que:

- Cada **contentor** tem informação sobre o destino e o valor total da sua mercadoria.
- A capacidade de um **armazém** é definida pela altura de cada pilha, que limita o número de contentores que podem ser colocados uns sobre os outros, bem como pelo comprimento da fila de pilhas de contentores (ver Figura 1).

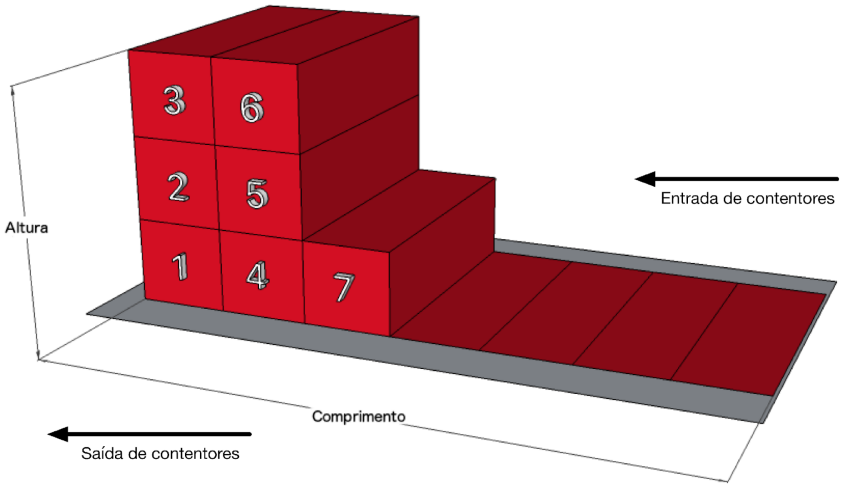

Figura 1 - Esquema de um armazém, contentores numerados por ordem de chegada

Fornecem-se os ficheiros ".c" e ".h" com as estruturas definidas e código parcialmente implementado:

- `contentor` – estrutura do contentor e código associado (para completar na aula);
- `armazem` – estrutura do armazém e código associado (para completar na aula);
- `pilha_contentores` – estrutura e código para uma pilha de contentores (não modificar);
- `fila_contentores` – estrutura e código para uma fila de pilhas de contentores (não modificar);
- `teste_armazem` – ficheiro de teste para o código a desenvolver nesta aula (não modificar);

Note que as filas e pilhas usadas na aplicação de gestão do armazém são implementadas com base em listas ligadas.

a) Analise a estrutura de dados *`contentor`* no ficheiro **`contentor.h`**. Implemente, no ficheiro **`contentor.c`** as funções *`contentor_novo()`* e *`contentor_apaga()`*, que devem utilizar gestão dinâmica de memória para, respetivamente, criar e apagar um contentor.

Protótipo:	`contentor* contentor_novo(char* dest, float val);`
Argumentos:	**`dest`** destino do contentor **`val`** valor das mercadorias no contentor
Retorno:	apontador para o contentor criado (NULL em caso de erro)

Protótipo:	`void contentor_apaga(contentor* contr);`
Argumentos:	**`contr`** apontador para o contentor a apagar
Retorno:	

b) Analise a estrutura de dados *armazem* no ficheiro **armazem.h**. Repare em particular que a um armazém está associada uma fila, cujos elementos são pilhas de contentores.

Implemente, no ficheiro **armazem.c**, a função que cria um armazém novo com o comprimento e altura definidos pelos parâmetros da função. Note que a fila de contentores deverá ser criada nesta altura, ainda que inicialmente fique vazia.

Protótipo:	`armazem* armazem_novo(int comprimento, int altura);`
Argumentos:	**comprimento** comprimento máximo da fila de pilhas de contentores **altura** altura máxima de cada pilha de contentores
Retorno:	apontador para o armazém criado (NULL em caso de erro)

c) Implemente, no ficheiro **armazem.c**, uma função que testa se um determinado armazém está vazio.

Protótipo:	`int armazem_vazio(armazem* armz);`
Argumentos:	**armz** apontador para o armazém a testar
Retorno:	1 se o armazém não contiver contentores (ou se armz for NULL), 0 no caso contrário

d) Implemente, no ficheiro **armazem.c**, uma função que testa se um determinado armazém está cheio. Note que para um armazém estar cheio, tanto a fila de contentores como a última pilha da fila devem estar na sua ocupação máxima (comprimento da fila e altura da última pilha).

Protótipo:	`int armazem_cheio(armazem* armz);`
Argumentos:	**armz** apontador para o armazém a testar
Retorno:	1 se o armazém estiver cheio, 0 no caso contrário (ou se armz for NULL)

e) Implemente, no ficheiro **armazem.c**, a função que realiza os procedimentos associados à chegada de um contentor ao armazém. Em concreto, esta função deve colocar o contentor recém-chegado no topo da pilha que se encontra na cauda da fila (no caso dessa pilha estar cheia, isto é, conter tantos contentores como a altura máxima permitida, a função é responsável por criar uma nova pilha e inseri-la na fila). Se o armazém estiver cheio e já não se puder colocar mais contentores, a função deve retornar 0, assinalando assim a falha. Caso contrário deve retornar 1.

Protótipo:	`int armazenar_contentor(armazem* armz, contentor* contr);`

Argumentos:	**armz** apontador para o armazém que recebe o contentor **contr** Apontador para o contentor recebido
Retorno:	1 se contentor foi armazenado, 0 se erro ou armazém cheio

f) Implemente, no ficheiro **armazem.c**, a função que realiza os procedimentos associados à saída do armazém de um contentor. O primeiro contentor a sair é aquele que se encontra no topo da pilha da cabeça da fila. Note também que quando se retira o contentor da base de uma pilha, deve-se eliminar essa pilha. A função deve retornar um apontador para o contentor retirado. Quando não há contentores no armazém para serem retirados, a função deve retornar NULL.

Protótipo:	`contentor* expedir_contentor(armazem* armz);`
Argumentos:	**armz** apontador para o armazém de onde vai ser retirado um contentor
Retorno:	apontador para o contentor retirado ou NULL se armazém vazio