Aula 2

Objectivos: Definir e implementar classes em C++
Compreender as vantagens do encapsulamento da informação
Utilizar a alocação e a libertação dinâmica de memória

Exercício 1

Pretende-se criar um programa de gestão de uma biblioteca. O programa deve implementar uma base de dados rudimentar que permita adicionar, remover, pesquisar e visualizar os registos relativos aos documentos existentes. Para simplificar, assuma que a biblioteca possui apenas livros.

1.1 Construa a classe `CBook`, para representar um livro, com os atributos adequados para armazenar a seguinte informação:

- Título
- Autor
- Ano de edição

A classe deve ter, além de um construtor adequado, métodos para alterar e aceder à informação armazenada em cada um dos atributos. A sua estrutura pode ser a seguinte:

```
#define MAX_TITLE_LENGTH  40
#define MAX_AUTHOR_LENGTH 50

class CBook
{
public:
     CBook(const char* pTitle, const char* pAuthor, unsigned int year);
     ~CBook();

public:
     void SetTitle(const char* pTitle);
     void SetAuthor(const char* pAuthor);
     void SetYear(unsigned int year);

     const char* GetTitle();
     const char* GetAuthor();
     unsigned int GetYear();

     void Print();

private:
     char m_title[MAX_TITLE_LENGTH + 1];
     char m_author[MAX_AUTHOR_LENGTH + 1];
     unsigned int m_year;
};
```

Numa primeira abordagem assuma que as *strings* armazenadas nos campos `m_title` e `m_author` têm um tamanho máximo predefinido e igual a `MAX_TITLE_LENGTH` e `MAX_AUTHOR_LENGTH` respectivamente.

1.2 Escreva um programa (função `main`) que exemplifique o uso da classe `CBook`, isto é, crie várias instâncias, aceda e modifique os atributos através dos métodos implementados e escreva no ecrã a informação relativa a vários livros usando o método `Print`.

1.3 Em vez de assumir um tamanho máximo predefinido para os campos `m_title` e `m_author`, altere a definição da classe `CBook` de forma a alocar dinamicamente a memória estritamente necessária para armazenar as respectivas *strings*. A memória deve ser alocada no construtor da classe e libertada no seu destrutor. Além disso, quando se alterar os campos `m_title` e `m_author` deve-se, se necessário, proceder ao reajustamento do tamanho da memória.

Exercício 2

Tendo criado a classe `CBook`, que representa um livro, o próximo passo é criar uma classe que representa a base de dados da biblioteca e que vai funcionar como contentor (recipiente) de objectos do tipo `CBook`.

2.1 Para tal, construa a classe `CLibrary` que permita armazenar vários livros. Para simplificar, assuma para já que existe um número máximo de livros definido pela constante `NUM_MAX_BOOKS`. A estrutura da classe pode ser a seguinte:

```
#define NUM_MAX_BOOKS     100

class CLibrary
{
public:
     CLibrary();
     ~CLibrary();

public:
     int CreateBook(const char* pTitle, const char* pAuthor, unsigned int year);
     int PrintBook(int code);
     void PrintAllBooks();

private:
     unsigned int m_numBooks;
     CBook* m_pBooks[NUM_MAX_BOOKS];
};
```

A classe `CLibrary` possui dois atributos: `m_numBooks` e `m_pBooks`. O primeiro armazena o número de livros da biblioteca. O segundo é um *array* de ponteiros para objectos do tipo `CBook`. Inicialmente, todos os ponteiros do *array* devem ser iniciados a `NULL`. O método `CreateBook` deve:

- procurar o primeiro elemento do *array* igual a `NULL` e fixar o respectivo índice;
- criar um objecto da classe `CBook` baseado nos parâmetros de entrada e colocar o seu endereço no elemento do *array* determinado no ponto anterior;
- incrementar a variável `m_numBooks`;

- devolver o código do livro, que corresponde ao índice utilizado do *array*, ou -1 em caso de erro.

Quando `m_numBooks` for igual a `NUM_MAX_BOOKS` a base de dados encontra-se cheia, pelo que não se podem acrescentar mais livros.
A função `PrintBook` deve devolver 0 se for bem sucedida ou -1 em caso de erro.

2.2 Escreva um programa que permita, repetida e interactivamente, adicionar novas entradas à base de dados e visualizar o seu conteúdo (sugestão: utilize um menu com as seguintes opções:

```
C - Criar livro
V - Visualizar livro
L - Listar todos os livros
X - Sair do programa
```

2.3 Escreva o método `DeleteBook` que permita apagar um livro da biblioteca baseado no seu código. Este método deve remover o livro da biblioteca, destruir o objecto e devolver 0 se for bem sucedido ou -1 em caso contrário.

```
class CLibrary
{
      // ...
      int DeleteBook(int code);
      // ...
};
```

2.4 Altere a função `main` de forma a usar este método.

Exercício 3

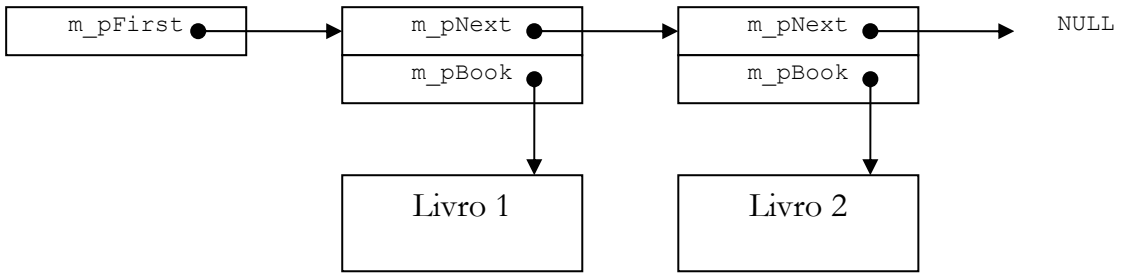
Substitua a implementação da classe `CLibrary` baseada num *array*, por uma lista ligada. Desta forma elimina-se a limitação do número máximo de livros.
Para implementar a lista ligada poderá criar-se uma estrutura de dados auxiliar que representa um nodo da lista. Esta estrutura deverá ter dois campos: um ponteiro para um livro e um ponteiro para o nodo seguinte da lista.

```
struct SListNode
{
      CBook* m_pBook;
      SListNode* m_pNext;
};
```

A classe `CLibrary` deve ter ainda um ponteiro adicional para o primeiro nodo da lista (que inicialmente deve ser colocado a `NULL`, uma vez que não existem livros).

```
SListNode* m_pFirst;
```

O ponteiro `m_pNext` do último nodo possui o valor `NULL`, uma vez que não existe nenhum nodo a seguir a ele.



No exercício anterior o código do livro correspondia ao índice do *array* onde era guardado o ponteiro para o respectivo objecto. No entanto, agora não faz sentido falar em índices do *array*. Assim, pense numa solução para implementar o código do livro.
A função `main` de teste deste exercício deverá ser a mesma do exercício anterior, uma vez que o interface da classe `CLibrary` não foi alterado.

Exercício 4

Acrescente à classe `CLibrary` do exercício 2 facilidades de pesquisa e ordenação baseadas nos seguintes métodos:

```
class CLibrary
{
      // ...
      void Search(int code);
      void Search(const char* pTitle);

      void SortByYear();
      void SortByAuthor();
      // ...
};
```

Os métodos de pesquisa (`Search`) devem mostrar no ecrã todos os livros que satisfazem o critério de procura ou apresentar uma mensagem de erro no caso de não terem sido encontrados quaisquer registos.

Os métodos `SortByYear` e `SortByAuthor` ordenam a biblioteca por ordem decrescente de ano e por ordem alfabética de autor, respectivamente.

Adicione estas novas facilidades à função `main` do exercício anterior.

Exercício 5

Repita o exercício anterior para a implementação baseada na lista ligada do exercício 3.