Aula 10

Objectivo: Generalização de classes para estruturas de dados comuns usando *templates*

Exercício 1

Altere a resolução da aula prática 5 – "Construção de classes para estruturas de dados comuns" – de forma a usar *templates*. Use como ponto de partida o código fonte disponível no site da disciplina. A manipulação da informação armazenada na lista ligada e na *stack* deixa de ser feita através de ponteiros do tipo `void*` para objectos com `dataSize` *bytes*, passando a ser realizada através de ponteiros e/ou referências para objectos de um tipo de dados genérico T, sendo T o parâmetro do *template* da classe. O objectivo é poder declarar, no programa principal, a variável que representa a *stack* de números reais da seguinte maneira:

```
CStack<double> stack;
```

A declaração da classe `CLinkedList` passa a ser a seguinte:

```
template <class T> class CLinkedList
{
protected:
	struct SNode
	{
		T m_data;
		SNode* m_pNext;
	};

public:
	CLinkedList();
	virtual ~CLinkedList();

public:
	int InsertHead(const T& data);
	int RemoveHead(T& data);

	void Empty();

	bool IsEmpty() const;

	unsigned int Count() const;

	const T* GetHead();
	const T* GetNext();

protected:
	unsigned int m_count;
	SNode* m_pHead;
	SNode* m_pCurrent;
};
```

Nota muito importante: Enquanto numa classe ordinária a implementação dos seus métodos deve em geral ser feita num ficheiro *.cpp, no caso de uma classe *template*, para que o compilador possa criar uma implementação concreta da classe (substituir os parâmetros do *template* por valores concretos), a sua implementação deve ser feita no ficheiro *.h, logo a seguir à definição do interface da classe.

Exercício 2

Construa uma classe chamada `CSortedLinkedList` que representa uma lista ligada, mas em que, ao contrário da anterior, a inserção dos elementos é feita de forma a manter a lista ordenada. Para tal, cada nodo da lista deve possuir um campo adicional do tipo inteiro que representa uma chave. Os nodos da lista devem estar sempre ordenados, da cabeça para a cauda, por ordem crescente da chave. Tal como no exercício anterior, a implementação da classe deve ser baseada em *templates* de forma a que o campo de dados da lista possa armazenar informação de qualquer tipo. O ficheiro de implementação deverá chamar-se `SortedLinkedList.h`. O interface pretendido para a classe `CSortedLinkedList` é o seguinte:

```
template <class T> class CSortedLinkedList
{
protected:
	struct SNode
	{
		int m_key;		// Chave associada ao nodo
		T m_data;		// Campo de dados do nodo
		SNode* m_pNext;		// Ponteiro para o nodo seguinte
	};

public:
	// Constroi uma lista vazia
	CSortedLinkedList();

	// Destroi a lista apagando todos os seus elementos
	virtual ~CSortedLinkedList();

public:
	// Insere na lista o elemento "data" com a chave "key"
	int Insert(int key, const T& data);

	// Remove da lista e devolve em "data" o elemento com a chave "key"
	int Remove(int key, T& data);

	// Remove da lista e devolve em "data" o elemento que se encontra à cabeça
	// da lista assim como a respectiva chave em "key"
	int RemoveHead(int& key, T& data);

	// Esvazia a lista
	void Empty();

	// Verifica se a lista está vazia
	bool IsEmpty() const;

	// Devolve o número de elementos da lista
	unsigned int Count() const;

	// Devolve em "data" o elemento que se encontra à cabeça da lista assim como
	// a respectiva chave em "key"
	int GetHead(int& key, T& data);

	// Devolve em "data" o próximo elemento da lista assim como a respectiva
	// chave em "key" - Consulte o guião da lição 5 para mais informações
	int GetNext(int& key, T& data);

	// Devolve em "data" o elemento com a chave "key"
	int Get(int key, T& data) const;

protected:
	unsigned int m_count;	// Número de elementos armazenados na lista
	SNode* m_pHead;		// Ponteiro para o primeiro nodo da lista
```

```
        SNode* m_pCurrent;          // Ponteiro para o último nodo cujo elemento de
                                    // dados foi devolvido pelo método GetNext()
};
```

Por conveniência defina os seguintes códigos de erro devolvidos pelos métodos da classe `CSortedLinkedList`:

```
        #define OK                  0
        #define OUT_OF_MEMORY       -1
        #define LIST_EMPTY          -2
        #define END_OF_LIST         -3
        #define INVALID_KEY         -4
```

Exercício 3

Utilize a classe desenvolvida no exercício anterior para alterar a resolução do teste prático 1 disponível no site da disciplina. O objectivo é usar a classe `CSortedLinkedList` de forma a implementar a base de dados de empregados com base numa estrutura de dados dinâmica baseada numa lista ligada em vez da solução semi-estática disponibilizada. Desta forma é removida a limitação do número máximo de empregados.
Como apenas é necessário alterar a forma como são guardados os ponteiros para os objectos que armazenam informação sobre os diversos empregados da base de dados, os únicos ficheiros que é necessário alterar são os que definem a base de dados, ou seja, o `BaseDados.h` e o `BaseDados.cpp`. Os restantes ficheiros, assim como o programa principal devem permanecer inalterados. É também necessário incluir no projecto o ficheiro que define o interface e a implementação da classe `CsortedLinkedList` (`SortedLinkedList.h`). O interface da classe `CBaseDados` passa a ser o seguinte:

```
#include "Docente.h"
#include "Funcionario.h"
#include "SortedLinkedList.h"

class CBaseDados
{
public:
        CBaseDados();
        virtual ~CBaseDados();

public:
        int RegistarDocente(const char* nome, const char* telefone,
                            float salario, EPosicao posicao);
        int RegistarFuncionario(const char* nome, const char* telefone,
                                float salario, const char* servico);

        int ModificarTelefone(unsigned int numero, const char* telefone);

        int ApagarEmpregado(unsigned int numero);

        int VisualizarEmpregado(unsigned int numero);
        void ListarTodosEmpregados();

private:
        unsigned int m_proximoNumero;
        CSortedLinkedList<CEmpregado*> m_listaEmpregados;
};
```

O atributo `m_listaEmpregados` é uma lista ligada usada para armazenar os ponteiros para cada um dos empregados, substituindo o *array* de ponteiros para `CEmpregado` usado na solução semi-estática.