Aula 11

Objectivo: Processamento de excepções.

Exercício 1

1. Adapte o código do *array* dinâmico de inteiros relativo ao exercício 2 da aula 6 e disponibilizado no *site* da disciplina de forma a que as situações de erro detectadas pelos construtores e métodos da classe `CIntArray` passem a ser comunicadas aos seus utilizadores usando excepções.
1.1 Pretende-se que as excepções geradas pela classe `CIntArray` sejam divididas em dois tipos: o primeiro relativo a falhas na obtenção de memória quando se pretende efectuar uma alocação ou realocação e, o segundo quando é feito o acesso a um índice do *array* fora da sua gama válida. Para tal, crie uma hierarquia de classes tal que a classe `CExcept` represente uma qualquer excepção, sendo por isso a classe base, e as classes `CMemoryExcept` e `CRangeExcept`, classes derivadas da anterior que representam excepções para os dois casos de erro apresentados acima, respectivamente. Tendo em consideração o código de interface das três classes apresentado a seguir, implemente cada uma delas.
O código de interface pretendido para a classe `CExcept` é o seguinte:

```
class CExcept
{
public:
    CExcept();
    virtual ~CExcept();

public:
    virtual void PrintMessage(ostream& os) = 0;
};
```

A classe `CExcept` é uma classe abstracta pois possui um método virtual puro. Consequentemente, não podem ser instanciados quaisquer objectos desta classe.
O código de interface pretendido para a classe `CMemoryExcept` é o seguinte:

```
#include "Except.h"

class CMemoryExcept : public CExcept
{
public:
    CMemoryExcept(unsigned int reqSize);
    CMemoryExcept(const CMemoryExcept& memoryExcept);
    virtual ~CMemoryExcept();

public:
    virtual void PrintMessage(ostream& os);

protected:
    unsigned int m_reqSize;
};
```

O primeiro construtor da classe `CMemoryExcept` deve copiar o valor de `reqSize` para o seu atributo `m_reqSize`. Este atributo representa o espaço em bytes de memória que se pretendia reservar e cuja alocação falhou.

O código de interface pretendido para a classe `CRangeExcept` é o seguinte:

```
#include "Except.h"

class CRangeExcept : public CExcept
{
public:
    CRangeExcept(unsigned int usedIndex, unsigned int maxIndex);
    CRangeExcept(const CRangeExcept& rangeExcept);
    virtual ~CRangeExcept();

public:
    virtual void PrintMessage(ostream& os);

protected:
    unsigned int m_usedIndex;
    unsigned int m_maxIndex;
};
```

O primeiro construtor da classe `CRangeExcept` deve copiar os valores de `usedIndex` e `maxIndex` para os atributos `m_usedIndex` e `m_maxIndex`, respectivamente. O primeiro representa o índice que se pretende aceder e o segundo representa o índice máximo do *array* que corresponde a `m_usedSize-1` da classe `CIntArray`. O método `PrintMessage` deve ser usado para enviar para o ecrã uma mensagem de erro apropriada para ambos os casos.

1.2 Altere o seu *array* dinâmico de acordo com o seguinte código de interface:

```
#include "MemoryExcept.h"
#include "RangeExcept.h"

class CIntArray
{
public:
  //Construtor usado para criar um array vazio
  CIntArray(unsigned int allocStep = 5);

  //Construtor usado para criar um array igual ao objecto "array"
  CIntArray(const CIntArray& array) throw (CMemoryExcept);

  //Destrutor do array
  virtual ~CIntArray();

  //Acrescenta o elemento "data" no fim do array
  void Append(int data) throw (CMemoryExcept);

  //Insere o elemento "data" no índice "index" do array
  void Insert(unsigned int index, int data) throw (CMemoryExcept, CRangeExcept);

  //Apaga e devolve em "data" o elemento que se encontra no índice "index" do array
  void Remove(unsigned int index, int& data) throw (CRangeExcept);

  //Apaga sem devolver o elemento que se encontra no índice "index" do array
  void Delete(unsigned int index) throw (CRangeExcept);

  //Apaga do array todas as ocorrências do elemento "data"
  void DeleteAllOccurences(int data);

  //Modifica para "data" o elemento que se encontra no índice "index" do array
  void Set(unsigned int index, int data) throw (CRangeExcept);

  //Devolve em "data" o elemento que se encontra no índice "index" do array
  void Get(unsigned int index, int& data) throw (CRangeExcept);

  //Esvazia o array, isto é, apaga todos os elementos
```

```
  void Empty();

  //Verifica se o array está vazio
  bool IsEmpty() const;

  //Devolve o número de elementos armazenados no array
  unsigned int GetSize(void) const;

  //Devolve o índice do primeiro elemento do array com o valor "data"
  int Find(int data) const;

  //Operador usado para atribuir ao objecto "*this" o objecto "array"
  CIntArray& operator=(const CIntArray& array) throw (CMemoryExcept);

  //Devolve o elemento armazenado no índice "index" do array
  int& operator[](unsigned int index) const throw (CRangeExcept);

  //Escreve o array na stream de saída "os"
  friend ostream& operator<<(ostream& os, const CIntArray& array);

protected:
  //Método auxiliar usado para fazer crescer, se necessário,
  //"m_allocStep" elementos ao bloco de memória onde é armazenado o array
  void GrowArrayIfNeeded() throw (CMemoryExcept);

  //Método auxiliar para libertar, se possível, "m_allocStep" elementos
  //não usados do bloco de memória onde é armazenado o array
  void ReduceArrayIfPossible();

  //Método auxiliar para copiar o objecto "array" para o objecto "*this"
  void Copy(const CIntArray& array) throw (CMemoryExcept);

protected:
  //Ponteiro para o bloco de memória onde é armazenado o array
  int* m_pArray;

  //Nº de elementos armazenados e libertados de cada vez que o array é realocado
  const unsigned int m_allocStep;

  //Tamanho usado do array
  unsigned int m_usedSize;

  //Tamanho do bloco de armazenamento do array
  unsigned int m_allocatedSize;
};
```

Exercício 2

Implemente uma classe chamada `CObjArray` para armazenar objectos de qualquer tipo num *array* dinâmico. Para tal, baseando-se na classe `CIntArray`, transforme-a numa classe *template*, mantendo a utilização de excepções.

Exercício 3

Modifique a implementação da classe `CStr` do teste prático 2 de forma a gerar excepções sempre que for preciso notificar o utilizador da classe, de erros ocorridos nos seus construtores, métodos e operadores. Implemente completamente a classe (incluindo os métodos que não foram pedidos no enunciado). Para simplificar utilize as classes implementadas no exercício 1 deste guião. Adapte também o programa principal fornecido de forma a detectar todas as excepções ocorridas dentro da classe `CStr`.