Aula 10

Objectivo: Templates

Problema 1

Construa uma classe `ARRAY` que implemente um "array" dinâmico genérico (ou seja cujos elementos podem ser de um qualquer tipo). Esta classe deve utilizar adequadamente "templates" e – tal como acontecia no problema 1 da aula 8 – deve ter os operadores normais na utilização de um array, e deve ser devidamente protegida contra maus usos.

```
template <class T>
class ARRAY
{
public:
  ARRAY(int indice_minimo,int indice_maximo);
  ARRAY(int num_elementos); // índices de 0 a num_elementos-1
  virtual ~ARRAY();

  ARRAY(ARRAY&outro);
  ARRAY &operator =(ARRAY&outro);

  T &operator [](int indice);

  int indice_minimo(void);
  int indice_maximo(void);
  int tamanho(void);

// (...)
};
```

1.1. Teste a classe instanciando-a, por exemplo, com um array de números inteiros.

1.2. Faça procedimentos globais (com "templates") que permitam a ordenação de "arrays" um com o algoritmo de selecção e outro com o de bolha (NOTA: assuma que os tipos de dados ordenáveis têm implementadas os operadores de relação de ordem `operator <` e `operator >`).

1.3. Faça com que a utilização dos procedimentos de ordenação fique independente de um qualquer algoritmo de ordenação (NOTA: utilize ponteiros

para funções, ou então, verifique a possibilidade da utilização de classes para esse efeito).

Problema 2

Construa uma classe abstracta `LISTA` de um qualquer tipo de dados (`T`) com pelo menos os seguintes serviços públicos:

```
adiciona      // adiciona elemento (T) à lista
apaga         // apaga o elemento actual da lista
limpa         // faz com que a lista fique vazia
primeiro      // faz com que o elemento actual seja o primeiro
seguinte      // faz com que o elemento actual seja o seguinte
actual        // devolve o elemento (T) actual da lista
e_fora        // diz se não há elemento actual
e_primeiro    // diz se o elemento actual é o primeiro
e_ultimo      // diz se o elemento actual é o último
procura       // percorre a lista a partir da posição actual
              //  aplicando o operador = a cada elemento
              // se bem sucedida o elemento actual da lista
              //  passará a ser o elemento procurado, caso
              //  contrário ficará fora da lista
```

(NOTA: Esta classe é muito semelhante à definida no problema 1 da aula 5, aparte de não existir nenhuma classe `ELEMENTO_DE_LISTA`.)

2.1. Crie a classe `LISTA_COM_ARRAY`.

2.2. Crie a classe `LISTA_LIGADA`.

2.3. Crie a classe abstracta `LISTA_ORDENADA` (neste tipo de lista os seus elementos são sempre adicionados garantindo uma ordenação crescente ou decrescente).

2.4. Crie as classes `LISTA_ORDENADA_COM_ARRAY` e `LISTA_ORDENADA_LIGADA`.