Ficha de Trabalho – 4

Objectivos: Objectos dinâmicos. Matrizes dinâmicas Classes com construtores por cópia, operador de atribuição e destrutor

Exercício 1:

Analise o seguinte código. Trata-se de uma classe que não serve para nada em particular a não ser apresentar uma mensagem sempre que é criada explicitamente ou destruída. Passe o seu código para o computador e inclua as bibliotecas necessárias.

```
class MSG
{
        public:
        MSG(char * p) {       cout << "ola " << p << "\n";    }
        ~MSG() {       cout << "Adeus\n";    }
};
```

a) Na função main crie dois objectos a e b de forma dinâmica mas sem usar malloc. Os objectos são criados independentemente um do outro. Repare nas mensagens que aparecem no ecrã: a dos construtores aparecem, mas as dos destrutores não. Será que os objectos não são destruídos? Mas se o programa terminou, tudo o que pertence ao programa é libertado, objectos incluídos. O que se estará a passar? Explique.

b) Acrescente o código à função main para apagar os objectos e sem usar free. Experimente apagar os objectos por uma ordem (a e depois b) e depois por outra (b e depois a). Repare que tem total liberdade para escolher a ordem pela qual prefere apagar os objectos. Confirme pelas mensagens que agora os destrutores estão a ser invocados.

c) Acrescente ao seu programa uma função void func() e mova para dentro dela a criação dos dois objectos. Repare que a vida dos objectos deixa de ser controlada pela duração da função onde os criou. Neste caso os objectos são criados na função func e apagados já depois de essa função ter terminado. Confirme com as mensagens dos construtores e destrutores, inserindo também mensagens nas funções func e main para o ajudar a identificar os pontos por onde a execução vai passando.

d) Agora crie uma matriz dinâmica de 3 objectos. Repare que a sintaxe não lhe permite a criação da matriz por causa do construtor. Modifique o parâmetro do construtor da classe MSG de forma a que possa ser invocado sem nenhum parâmetro e confirme que agora já é possível a criação da matriz dinâmica. Execute o programa e repare nas mensagens. O primeiro elemento da matriz a ser criado é o que está na posição com índice 0, e o último é o que está na posição com índice 2.

e) Acrescente o código necessário para destruir a matriz dinâmica. Corra novamente o programa e confirme pela ordem das mensagens que os objectos da matriz dinâmica estão a ser destruídos.

Exercício 2:

Pretende-se construir uma classe que represente imagens num ecrã. As imagens são formadas por pontos de cor. Assim uma imagem não é mais do que matriz de pontos organizada em linhas e colunas. Cada ponto é uma associação de três cores: vermelho, verde e azul (cores primárias), sendo que a combinação destas três cores em diversas intensidades resulta em todas as cores que se conseguem ver. Já existe o seguinte código para a classe de pontos de cor (classe PixelRGB).

```
class PixelRGB
{
        char r,g,b;        // RED GREEN BLUE, valores de 0 a 255
        public:
                PixelRGB(char cr = 0, int cg = 0, int cb = 0) : r(cr), g(cg), b(cb) {}
                int getR() const { return r; }
                int getG() const { return g; }
                int getB() const { return b; }
                void setR(int c) { r = c; }
                void setG(int c) { g = c; }
                void setB(int c) { b = c; }
};
```

Usando a classe PixelRGB, escreva a classe Imagem para representar imagens rectangulares constituídas por pontos de cor. A classe Imagem deve ter as seguintes características:

- Armazena um número indeterminado de pontos de cor representados por objectos de PixelRGB. A classe armazena internamente os pontos de cor (objectos de PixelRGB).
- A construção de objectos de Imagem pressupõe a indicação do número de linhas e de colunas. A Imagem fica logo com número-de-linhas x número-de-colunas pontos de cor. Consegue prever qual será a cor inicial que esses pontos de cor vão ter?
- Deve ser possível obter/chegar ao ponto de cor que se encontra numa determinada posição linha e coluna. Faça uma função para este propósito devolvendo o ponto de cor:
  a) Por referência para objectos PixelRGB.
  b) Por ponteiro para objectos PixelRGB.
  c) De forma a que as seguintes expressões funcionem:

```
int main()
{
        Imagem img(30,40);
        PixelRGB aux = img[15][16]); // obtém ponto de cor em 15,16
        PixelRGB temp(30,40,50);
        img[13][15] = temp;        // coloca pixel de cor diferente
        return 0;
}
```

Não esqueça que a classe deve ser robusta para todas as operações habituais em que os seus objectos possam participar (passagem de objectos por cópia, por referência, por ponteiro, atribuição, inicialização, destruição, etc.).