Programação Orientada a Objetos 2023/2024

Exercícios

Ficha Nº 7

Classes com recursos dinâmicos
Ponteiros raw em situações de composição e de agregação

1. Ponto dinâmico

Considere a classe Ponto cujo código é dado em seguida. Copie o código para o seu projeto e, salvo indicações em contrário, não pode mudar o código.

Ponto.h

```
// includes e outras declarações omitidos
class Ponto {
public:
    Ponto(int cx, int cy);
    ~Ponto();
private:
    int x,y;
};
```

Ponto.cpp

```
// includes e outras declarações omitidos
Ponto::Ponto(int cx, int cy) : x(cx), y(cy) {
   cout << "CONSTR. Ponto com " << x << "," << y << "\n";
}
Ponto::~Ponto(){
   cout << "DESTR. Ponto com " << x << "," << y << "\n";
}
```

a) Escreva uma função *main* onde cria dinamicamente dois objetos **Ponto a** e **b**, com coordenadas (1,2) e (3,4), respetivamente. A última instrução da função *main* deverá ser a impressão no ecrã de uma mensagem "\nfinal\n" e nesse momento todos os objetos deste exercício (ambas as alíneas) deverão já ter sido apagados. Execute a função *main* nos seguintes cenários usando sempre os objetos **a** e **b**, vendo as mensagens que aparecem e tirando as devidas conclusões:

- Apague os objetos no final, primeiro **b**, depois **a**;
- Apague os objetos no final, primeiro **a**, depois **b**;
- Não apague os objetos;
- Apague duas vezes o mesmo objeto **a**, sem tocar no ponteiro para ele;
- Apague duas vezes o mesmo objeto **a** tendo colocado o ponteiro para ele a *nullptr* logo a seguir a o ter apagado na primeira vez.

b) Escreva uma nova versão da função *main* onde cria um *array* de três objetos **Ponto** com coordenadas (1,2), (3,4) e (5,6), respetivamente. Se não conseguir:

(1) Explique ao professor por que é que não consegue.

(2) Proponha e concretize uma solução que envolva alterar a classe **Ponto**. Justifique essa solução ao professor.

(3) Proponha e concretize uma outra solução que não mexa na classe **Ponto**, mas altere o *array*, mais concretamente, o seu conteúdo, tendo sempre obrigatoriamente que envolver objetos **Ponto** de alguma forma, direta ou indiretamente. A sua solução deve garantir que os objetos **Ponto** são criados e também destruídos antes do final da função *main*. Descreva e justifique a solução ao professor.

c) Escreva uma nova versão da função *main* onde agora existe o seguinte requisito: o número de objetos Ponto a criar é desconhecido à partida. Assim, deve criar uma forma de perguntar ao utilizador quantos pontos deseja criar, e de seguida, criar cada um deles, perguntando também ao utilizador quais as coordenadas que deseja para cada um deles. Explique ao professor quais são todos os recursos dinâmicos que são agora envolvidos e garanta que todos são destruídos antes de atingir a mensagem "\nfinal\n".

d) Escreva mais uma função *main* (coloque as anteriores em comentários) que permita perguntar ao utilizador quantas linhas e quantas colunas deseja, e de seguida crie uma matriz bidimensional de objetos Ponto com as dimensões especificadas. Por matriz bidimensional deve entender uma estrutura de dados que pareça uma matriz bidimensional, alocada dinamicamente e sem usar nenhuma coleção da STL. No final, a função deve libertar todos os recursos alocados.

- Quando terminar, adapte o que fez (mantendo o original em comentários) para matriz bidimensional de ponteiros para ponto para ver qual a diferença em termos de sintaxe. Crie os objetos ponto, um por um e no final liberte tudo: matriz e pontos.

No final do exercício deverá ter

- Percebido como se processa a criação e destruição de objetos de forma dinâmica.
- Entendido os operadores new e delete, e percebido por que é que não deve nunca usar malloc nem free.
- Percebido como se processa a criação de matrizes dinâmicas de objetos (1 e 2 dimensões), quais as suas restrições e o uso de new[] e delete[].
- Explorado a alternativa de matrizes dinâmicas de objetos vs matrizes de ponteiros para objetos dinâmicos.
- Percebido todos os aspetos envolvidos no cenário *matriz dinâmica de ponteiros para objetos dinâmicos*.

2. MyString

Assuma que não existe a classe de biblioteca STL *string* mas que se pretende algo semelhante a ela. Neste exercício pretende-se fazer uma classe **MyString** que represente uma cadeia de caracteres de uma forma parecida ao que a classe *string* da STL faz (obviamente, *não pode* usar aqui a classe *string*). Neste exercício **não pode** usar nenhuma coleção STL.

a) Escreva a classe **MyString** com as seguintes características

- Representa uma cadeia de caracteres de tamanho indeterminado, terminada por '\0'. Pode ser vazia, pequena, ou enorme, sendo a única limitação os recursos existentes na máquina.
- Os caracteres de um objeto **MyString** pertencem a esse objeto.
    - Os objetos de **MyString** são inicializados com:
    - Nada -> fica uma cadeia de caracteres vazia.
    - Um ponteiro de caracteres -> o objeto fica com uma cópia desses caracteres.
- É possível obter o tamanho da cadeia de caracteres. Por uma questão de performance, o objeto deverá manter armazenado o tamanho, atualizando-o sempre que a cadeia de caracteres muda.
- É possível obter a cadeia de caracteres armazenada internamente. No entanto, deve ser salvaguardado o seguinte: o resto do programa pode ver, mas não pode alterar o conteúdo da cadeia de caracteres.
- É possível obter, através de uma função **at()** o caracter numa determinado posição (que é indicada). A função deve permitir alterar esse caracter sendo usada do lado esquerdo de uma atribuição. A posição indicada é, naturalmente, validada, sendo rejeitados acessos a caracteres que não existem.
- Deve ser possível adicionar (concatenar) um conjunto de caracteres ao que já existe no objeto, usando para tal a função membro **concat()** que recebe um outro objeto **MyString**.
- Deve ser possível apagar os caracteres, ficando uma cadeia de caracteres vazia com uma função **clear()**.
- É natural e muito provável que os objetos **MyString** sejam criados, atribuídos, usados em funções, passados para e retornados de funções, por referência e sem ser por referência. É tão natural e provável, que normalmente nem é preciso referir isto, e o mais certo é não voltar a ser referido, sem que isso remova a obrigação das suas classes de terem este aspeto em consideração.

b) **TPC** (estudo autónomo). Acrescente à sua classe o necessário para que seja possível fazer o seguinte (sendo **a**, **b** e **c** objetos MyString)

- Concatenação alterando o objeto da esquerda: `(a += b) += c;`
- Concatenação sem alterar os originais: `a = b + c;`
- Acesso a um caracter para obter e modificar: `a[2] = b[3];`
- Comparação: `if (a==b) { … }`
- Comparação relacional lexicográfica: `if (a<b && c>b) { … }`
- Apresentação no ecrã: `cout << a << b;`
- Introdução de caracteres obtidos via stdin: `cin >> a >> b;`
- Obter do número de caracteres na cadeia de caracteres: `int len = (int) a;`
- Verificar se a cadeia de caracteres está vazia: `if (a) { /* não está vazia */ };`
- Obter uma cadeia de caracteres que representa um número: `MyString d(123);`

No final do exercício deverá ter

- Percebido quais as implicações de ter recursos dinâmicos numa classe em situação de composição e quais as implicações que isso traz nas operações de cópia, atribuição e destruição.
- Consolidado o uso e sintaxe de alocação dinâmica em C++.
- Consolidado o uso de operadores (alínea b em formato TPC/estudo autónomo)

3. Exercício preparatório - Pessoa – Classe de apoio

Crie um novo projeto e acrescente nele a classe Pessoa cujo código é apresentado abaixo. Esta classe não constitui exercício em si mesma e serve para apoio aos dois exercícios seguintes. Não pode acrescentar código a esta classe.

Pessoa.h

```
// includes e outras declarações omitidos

class Pessoa {
public:
    Pessoa(string nome, int bi, int nif);
    string getNome() const;
    int getBI() const;
    int getNIF() const;
    void setNome(string nome);
    string descricao() const;
private:
    string nome;
    int bi, nif;
};
```

Classe *Pessoa*, descrita por nome, número de bilhete de identidade (BI), e número de contribuinte (NIF), sendo necessário indicar todos estes dados na sua inicialização. Permite obter cada um dos seus dados, e permite atualizar o nome. Permite obter uma *string* com a descrição do seu conteúdo.

Pessoa.cpp

```
// includes e outras declarações omitidos

Pessoa::Pessoa(string _nome, int _bi, int _nif) : nome(_nome), bi(_bi), nif(_nif) { }
string Pessoa::getNome() const { return nome; }
int Pessoa::getBI() const { return bi; }
int Pessoa::getNIF() const { return nif; }
void Pessoa::setNome(string _nome) { nome = _nome; }
string Pessoa::descricao() const {
    ostringstream oss;
    oss << nome << " " << bi << " " << nif;
    return oss.str();
}
```

4. Registo Civil de pessoas – Composição de recursos dinâmicos e uma coleção

Escreva uma classe **RegistoCivil** semelhante às versões anteriormente já feitas deste tema, mas com algumas simplificações e pequenas diferenças. Reproduz-se aqui o essencial, com as diferenças assinaladas em ***itálico negrito***:

- Na sua inicialização exige o nome do país a que diz respeito. Inicialmente não tem pessoa nenhuma.
- É possível obter, mas não mudar o nome do país.
- Suporta a existência de um número **indeterminado de pessoas**, limitado apenas pelas capacidades da máquina. Neste exercício **vai obrigatoriamente usar um vetor** da STL.
- Permite adicionar uma nova pessoa, a qual é construída no contexto da classe e não fora dela. Para esta operação são indicados os dados da nova pessoa. A operação falha se já existir nesse país uma pessoa com o BI indicado. *Não havendo máximo específico, devem ser suportadas tantas pessoas quanto as que couberem em memória.*
- Permite obter o nome de uma pessoa dado o seu BI.
- Permite obter a listagem de todas as pessoas numa *string*, uma pessoa por linha.
- Permite atualizar o nome de uma pessoa dado o seu BI e o novo nome.
- Permite obter o número de pessoas atualmente existentes.
- Permite obter o ponteiro para uma pessoa dado o BI. Este ponteiro irá ser usado por outras classes de forma permanente, pelo que **é importante que o objeto Pessoa não mude de lugar enquanto existir**.
- Não existe, para já, forma de apagar pessoas. No entanto, isso não é garantia automática que os objetos Pessoa não mudem de sítio na memória. Se tem dúvidas na interpretação desta afirmação, fale com o professor.

a) Implemente a classe (.h e .cpp + função *main*) e experimente a sua funcionalidade com algumas operações a partir da função main. Garanta que a classe está apta a ser usada em todas as operações em que um objeto pode ser usado em C++ (por exemplo: ser criado, destruído, copiado, atribuído, etc.). Esta chamada de atenção refere-se a algo que é óbvio e é natural que deixe de ser feita desta forma explícita. ***Este exercício é importante para o que se segue. Garanta que o faz.***

b) Considere que iria usar um **set** em vez de um **vector**. De que forma isso o ajudaria a garantir que os objetos **Pessoa** permaneceriam sempre no mesmo local de memória, melhor possibilitando a existência de ponteiros para eles no resto do programa? Trata-se de uma questão para pensar e interagir com o professor, não devendo alterar nada no código já feito.

No final do exercício deverá ter

- Lidado com uma situação em que uma classe tem recursos dinâmicos que deve gerir de forma explícita apesar de recorrer a uma coleção.
- Explorado um cenário onde é exportado um ponteiro para um objeto interno e percebido os riscos que isso acarreta.

5. Clube – Recursos dinâmicos apesar de ser cenário de agregação

Refaça a classe que representa o conceito de *clubes de bairro* em C++ de uma ficha anterior. Nesta versão, a funcionalidade é exatamente a mesma, mas a implementação tem as seguintes diferenças:

- Vai usar ponteiros para Pessoa para estabelecer a ligação entre o clube e os seus sócios. Neste cenário não precisa de ter conhecimento direto acerca do registo civil.
- O número de sócios é indeterminado/ilimitado, mas não pode recorrer a nenhuma coleção da STL.

A classe deve ter a seguinte funcionalidade:

- Inicialização dado o nome da atividade do clube. Inicialmente fica sem sócio nenhum.
- Adicionar um sócio dado o ponteiro para essa pessoa. Não pode haver sócios repetidos (mesmo BI).
- Remover um sócio dado o seu BI.
- Obter a lista com os nomes de todos os sócios (um nome por linha)

A classe deve estar preparada para que os seus objetos se comportem corretamente no contexto da linguagem C++.

a) Implemente a classe **Clube** (.h e .cpp), adicionando-a ao projeto do exercício anterior e experimente a sua funcionalidade com algumas operações a partir da função *main*.

b) Acrescente a seguinte funcionalidade ao **RegistoCivil**: remover uma pessoa dado o seu BI.

- Concretize mesmo essa funcionalidade e garanta que não ocorrem "*memory leaks*".
- Analise as consequências para o Clube e identifique possíveis problemas que podem agora surgir. Debata com o professor esta questão e proponha soluções possíveis (sem implementar nada). Uma forma possível de solução passa por um novo tipo de ponteiro que é abordado noutra ficha e é importante que percebe o problema que se coloca agora.

No final do exercício deverá ter

- Lidado com uma situação em que uma classe está relacionada com um cenário de agregação, mas mesmo assim podem estar envolvidos recursos dinâmicos por composição.
- Explorado as consequências de ter ponteiros *raw* para um recurso dinâmico por agregação, sendo que esse recurso é controlado por outra parte do programa e pode ser eliminado a qualquer momento.