Programação Orientada a Objetos 2023/2024

Exercícios

Ficha Nº 6

Coleções: vector, set, map
Algoritmos

1. **Pessoa** – Classe de apoio

Crie um novo projeto e acrescente nele a classe Pessoa cujo código da classe Pessoa apresentado abaixo (em alternativa, obtenha o código da classe pessoa já anteriormente usado, por exemplo, nos exercícios 6, 7 e 8 da ficha 4, mas garanta que não tem um construtor por omissão). Esta classe não constitui exercício em si mesma e serve para apoio ao exercício propriamente dito que continua no número seguinte.

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;
    // vai precisar de ser acrescentado por causa de RegCiv
    // Pessoa();
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();
}
```

2. Registo Civil de pessoas – sem arrays nem memória dinâmica

Escreva uma classe **RegistoCivil** semelhante ao RegistoCivil do exercício 7 da ficha 4: representa o mesmo conceito com a mesma funcionalidade genérica, mas com algumas 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 finito, ***mas indeterminado de pessoas. Não pode assumir nenhum limite máximo específico***.
- 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 adicionar pessoas lidas de um ficheiro de texto (nome do ficheiro é indicado). O ficheiro tem uma pessoa por linha, no formato nome BI NIF. Nesta funcionalidade assuma que o nome tem apenas uma palavra.
- ***Permite exportar para ficheiro de texto todas as pessoas (não as apaga). O formato do ficheiro é compatível com o que ficheiro usado na funcionalidade referida no ponto anterior.***
- Permite apagar uma pessoa dado o seu BI.
- 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, ***de forma compatível com os ficheiros referidos atrás.***
- Permite atualizar o nome de uma pessoa dado o seu BI e o novo nome.
- Permite obter o número de pessoas atualmente existentes no país.
- ***Permite obter o número de pessoas cujo nome contém uma determinada palavra.***
- ***Permite apagar todas as pessoas cujo BI está entre dois valores especificados.***
- ***Apaga todas as pessoas.***

Restrições na implementação:

- Não pode usar memória dinâmica (isso acontecerá noutra versão do exercício), nem *arrays*.
- Não pode modificar a classe Pessoa.
- Não pode usar objetos de Pessoa "fictícios" (por exemplo BI = -1 significa que não é uma pessoa real, como aconteceu no exercício equivalente da ficha 4).

a) Implemente a classe (.h e .cpp + função *main*) usando ***vectores*** da biblioteca C++. Os algoritmos das funções membro devem ser escritos explicitamente sem recorrer a nenhum algoritmo de biblioteca. Na funcionalidade de adição de pessoas, deve experimentar duas alternativas de implementação: com e sem o uso de ***emplace***. Teste a funcionalidade com algumas operações sintéticas na função *main*.
Esta versão da implementação vai ser usada mais adiante noutra ficha. Garanta que a faz e que a guarda.

b) Refaça a alínea a), mas agora usando ***sets*** da biblioteca standard. Explique ao professor a forma como o set o ajuda e simplifica na questão de garantir que não haverá duas pessoas com o mesmo BI. **Importante**: nesta alínea poderá ter que acrescentar alguma coisa à classe Pessoa – identifique o que é e faça essa alteração. Garanta que tem mantém o código anterior de forma independente desta nova versão.

c) Refaça a alínea b), mas agora usando ***maps*** da biblioteca standard. Neste caso deve considerar o BI como chave e o objeto Pessoa como sendo o valor. **Discuta com o professor esta questão**: "*de que forma o facto do BI nunca poder ser modificado contribui para que a relação chave-valor em BI-Pessoa nunca fique incoerente?*" (é importante que entenda isto). Garanta que tem mantém o código anterior de forma independente desta nova versão.

d) Acrescente à sua classe Pessoa (ficheiro .h) o seguinte

```
Pessoa(const Pessoa &) = delete;
```

Quais das alíneas a) ,b) e c) deixam de compilar. Explique a razão ao professor e volte a colocar a classe Pessoa como estava.

e) Tendo por base a alínea a) (mas a questão aplica-se também a sets e maps), faça:

i. Acrescente ao Registo Civil a funcionalidade de obter o **ponteiro** para uma pessoa dado o seu BI.
ii. Na função *main* crie um RegistoCivil com uma Pessoa qualquer.
iii. Obtenha e armazene o ponteiro para essa pessoa
iv. Insira uma nova pessoa.
v. Obtenha o nome da primeira pessoa através do ponteiro que armazenou
vi. Se o programa tiver rebentado: explique o que aconteceu ou pergunte ao professor. É muito importante
vii. Se não tiver rebentado: acrescente novas pessoas e remova algumas (mas não a primeira) até rebentar. É inevitável que vai rebentar e **é muito importante que perceba a razão**.

f) Na sequência da experiência que fez na alínea anterior, considere que o vector não é `vector<Pessoa>` mas sim `vector <Pessoa *>`, ou seja, ***no vector fica apenas o ponteiro, sendo a pessoa armazenada fora do vector***. Sem implementar nada - considere apenas - responda a estas duas questões:

i) Se repetisse a experiência da alínea anterior, o problema continuava a manifestar-se? **Porquê?**

ii) Se copiar um objeto de Registo civil (por exemplo, um parâmetro de função por cópia) o programa iria funcionar bem? **Porquê?** Este assunto vai ser explorado na próxima ficha.

g) Refaça as alíneas a), b) e c) **usando algoritmos de biblioteca** sempre que possível, simplificando o seu código. Mantenha o código das versões anteriores para que tenha acesso a todas as versões como material de estudo.

No final do exercício deverá ter

- Explorado as classes de coleção de biblioteca standard: vector, set e map.
- Tido contacto com a o assunto de *requisitos que uma classe X deve cumprir para que possa ser usada em coleções*
- Percebido a essência de *emplace* em vez da forma de inserção normal e quais as suas vantagens.
- Percebido quais os perigos de armazenar, no programa, ponteiros (permanentes) para elementos que estão dentro de coleções.
- Tido um primeiro contacto com os problemas associados a objetos que detêm (por composição) recursos alocados dinamicamente
- Explorado o uso de algoritmos da biblioteca standard e definido algumas expressões *lambda*.

3. **Clube de bairro de pessoas** vectores ou set e algoritmos

Refaça a classe que representa o conceito de *clubes de bairro* em C++ do exercício 8 da ficha 4 usando:

- Primeiro um vector, depois um set (em alternativa)

  Importante: no vector (ou set) do **Clube** ficará algo que identifique a Pessoa, mas não o ponteiro pelas razões descobertas no decorrer das alíneas e) e f) do exercício anterior. Isto obriga a uma lógica de "consulta tipo base de dados", com inevitáveis perdas de performance. Seria melhor ter acesso direto por ponteiro ao objeto controlado pelo **RegistoCivil**, o que obriga a que este último armazene as suas pessoas por ponteiro e não diretamente dentro do seu vetor, mas esta questão surge apenas na próxima ficha.
- Algoritmos sempre que possível

No final deste exercício terá

- Consolidado os conhecimentos acerca de vector e set da biblioteca *standard*
- Consolidado as questões relacionadas com armazenamento de ponteiros para objetos armazenados diretamente no interior de coleções
- Consolidado a experiência em algoritmos da biblioteca *standard*