Instituto Superior de Engenharia de Coimbra
Departamento de Engenharia Informática e de Sistemas
Engenharia Informática

Programação Orientada a Objectos 2022/2023

Exercícios

Ficha Nº 2

Encapsulamento e abstração
Classes
Excepções
Construtores / destrutores
Initializer_list
std::vector, iteradores

1. Defina uma estrutura chamada *Tabela*, que contenha uma matriz de inteiros. No decorrer das alíneas seguintes teste a funcionalidade pedida através de uma função *main* cujo código fica em aberto.

a) Escreva uma função que preencher toda a matriz da estrutura com um valor especificado.

b) Escreva uma função que permita listar o conteúdo da estrutura.

c) Escreva uma função que permita obter o valor num elemento da matriz, membro da estrutura, dado a posição do elemento pretendido. Este acesso deve ser protegido de tentativas de utilização de índices inválidos. Estas funções não interagem diretamente com o ecrã.

d) Escreva funções que permitam atualizar o valor num elemento da matriz, membro da estrutura, dado a posição do elemento pretendido e o novo valor. Este acesso deve ser protegido de tentativas de utilização de índices inválidos. Esta função não interage diretamente com o utilizador.

e) Unifique a funcionalidade das funções das duas alíneas anteriores numa só, cuja utilização se exemplifica a seguir. O acesso aos elementos da matriz da estrutura **deve ser protegido de tentativas de utilização de índices não válidos**, usando a o conhecimento atual da matéria.

```
int main(){
   Tabela a;
   cout << elementoEm(a,10);     // aparece um determinado valor
   elementoEm(a,10) = 15;        // notar que a chamada à função fica
                                 // do lado esquerdo da atribuição
   cout << elementoEm(a,10);     // aparece 15
}
```

f) Oiça primeiro a explicação acerca de **excepções em C++** dada pelo professor no início desta alínea (trata-se de uma primeira explicação que será completada mais tarde quando houver mais matéria). Depois, utilize o mecanismo de exceções em C++ para garantir que o acesso aos dados da matriz é válido (o elemento solicitado existe).

- Invoque a função que acede ao elemento a partir de uma segunda função de forma a melhor perceber a ***propagação de excepções***. São envolvidas três funções: a main(), esta, e a que acede ao elemento.

Neste exercício existem duas questões importantes

- Restrições importantes acerca do uso de referências, neste caso como retorno de função
- Forma de lidar com situações de erro que não podem ser recuperadas pela função/bloco de código onde ocorrem essas situações

É importante que no final do exercício tenha compreendido estes assuntos de uma forma que não seja meramente superficial. Interaja com o professor ainda durante a aula para garantir que percebeu estas duas questões

Relativamente ao assunto das excepções, garanta que percebeu estes pontos importantes (pergunte ao professor caso subsistam dúvidas ou caso nem sequer reconheça estes tópicos):

- Sintaxe de excepções. Clausulas try, catch. Funções noexcept. Operador noexcept.
- Tratamento de excepções; lançar excepções com throw e apanhar excepções com catch.
- Propagação de excepções.
- Importância de não propagar excepções excepto em caso de justificada necessidade.
- Excepções não são uma alternativa ao retorno de códigos de erro de funções, e ambas as estratégias tem os seus cenários de aplicação próprios.

Objetivos do exercício

- Treinar a capacidade de abstração e representação de informação.
- Entender o uso de referências como retorno de funções, quais as restrições associadas ao seu uso e formas de resolver essas restrições.
- Consciencializar que as funções que são responsáveis por armazenar e manipular dados não devem interagir diretamente com os periféricos nem com o utilizador, e vice-versa.
- Preparação para o exercício seguinte.

2. Pretende-se agora passar a estrutura *Tabela* do exercício anterior a classe de C++, com os mesmos membros variáveis da estrutura do exercício anterior e funções membros com funcionalidade análoga às funções consideradas para a referida estrutura.

a) Defina a classe Tabela com as características enunciadas. Se mantiver a estrutura do exercício anterior tenha em atenção que deve ter um nome diferente. Inclua uma função *main* que permita testar a classe *Tabela*.

b) Acrescente a seguinte requisito à classe: cada objecto de Tabela, quando é criado deve ter todos os seus valores inicializados a 0. Acrescente o código necessário à classe para cumprir este requisito.

c) Acrescente à sua classe a possibilidade de inicializar os valores com um valor que é indicado pelo programa. Importante: na inicialização não pode interagir com o utilizador.

d) Acrescente uma funcionalidade que permita obter uma "lista" dos valores contidos na tabela. O objeto Tabela não pode interagir com o utilizador. Oiça a explicação do professor acerca disto e das razões para tal. Nota "lista" não significa lista, mas apenas algo que contenha a informação acerca dos valores.

e) Identifique as vantagens desta nova versão de *Tabela* e anote-as no seu caderno.

Objetivos do exercício
(na continuação do anterior)

- Consolidar os objetivos atingidos no exercício anterior.
- Entender e treinar o conceito de encapsulamento e compreender as suas vantagens, incluindo: compreender a diferença entre funções globais e funções membro, as diferenças entre dados globalmente acessíveis e dados privados, e as vantagens de *data-hiding*.
- Utilizar os modificadores de visibilidade de C++ para definir o que é público e o que é privado.
- Entender o conceito de função membro que é chamada sobre um objeto e tomar conhecimento do ponteiro *this*.
- Entender as diferenças e as semelhanças entre estrutura e classe.
- Utilização de construtor.

3. Defina uma estrutura *Caderno* contendo os seguintes dados: marca, cor, número de folhas, tamanho. Na resolução deste exercício (e genericamente, em linguagens orientadas a objetos) evite funções membro que interajam diretamente com o utilizador a não ser que a classe tenha por objetivo especifico essa interação. Teste o código do exercício com uma função *main* adequada.

a) Escreva uma função que preencha o conteúdo de uma variável do tipo *Caderno*.

b) Escreva uma função para obter cada dado individual da estrutura.

c) Escreva uma função para modificar cada dado individual da estrutura.

Objetivos do exercício

- Treinar a capacidade de abstração e representação de informação.
- Consolidar a noção de *separação de representação e manipulação de dados para um lado, interação com o utilizador para outro*.
- Preparação para o exercício seguinte.

4. Pretende-se agora *Caderno* como classe, com os mesmos membros variáveis da estrutura do exercício anterior e funções membros com funcionalidade análoga às funções consideradas para a referida estrutura.

a) Construa a classe Caderno e construa ou modifique a função *main* para testar a classe.

b) Identifique as alterações principais quando Caderno de estrutura para classe.

c) Identifique e explique as vantagens que se obtêm ao passar Caderno para classe. Anote as suas conclusões.

Objetivos do exercício
(na continuação do anterior)

- Consolidar os objetivos atingidos no exercício anterior.
- Consolidar os conceitos de encapsulamento e compreender as suas vantagens, incluindo: compreender a diferença entre funções globais e funções membro, as diferenças entre dados globalmente acessíveis e dados privados, e as vantagens de *data-hiding*.
- Utilizar os modificadores de visibilidade de C++ para definir o que é público e o que é privado.
- Consolidar o conceito de função membro que é chamada sobre um objeto e tomar conhecimento do ponteiro *this*.
- Entender e consolidar o conceito de funções para aceder e para atualizar dados da classe.
- Consolidar o conhecimento acerca das diferenças e semelhanças entre estrutura e classe.
- Utilização de construtores.

5. Defina uma estrutura *Automovel* que contenha os dados necessários à representação de um automóvel (matrícula, combustível, marca, modelo ou outros). No decorrer das alíneas seguintes deve usar uma função *main* que lhe permita testar o código que for fazendo. A função *main* não faz diretamente parte do exercício.

a) Analise o problema e determine que dados devem pertencer à estrutura e quais os mecanismos e regras que devem ser materializados em funções para lidar com estas estruturas.

b) Defina a estrutura e escreva as funções que identificou.

c) Escreva uma função que preencha o conteúdo de uma variável do tipo Automovel com valores fornecidos. Evite interagir com o utilizador directamente dentro das funções que manipulam os dados da estrutura.

f) Escreva funções que permitam obter e atualizar cada dado desta estrutura.

Objetivos do exercício

- Treinar e consolidar a capacidade de abstração e representação de informação.
- Treinar a capacidade de determinação autónoma de dados e funcionalidade a incluir numa estrutura ou classe.
- Entender as desvantagens associadas a ter funções que lidam com dados e sua representação interna a interagirem com o utilizador, por oposição a ter para um lado, funções que lidam apenas com a representação e manipulação de dados, e para outro lado, funções que interagem com o utilizador.
- Preparação para o exercício seguinte.

6. Pretende-se passar a estrutura *Automovel* para classe de C++, mantendo os dados e funcionalidade. Acompanhe o código deste exercício com uma função *main* que lhe permita testar a sua classe. Este exercício também pode ser feito sem passar primeiro pelo anterior, caso em que bastará ler o enunciado do exercício anterior para saber as características de *Automovel*.

a) Construa a classe *Automovel* com as características enunciadas e inclua na sua função *main* código que permita testar a classe.

b) Analise as regras mais óbvias do mundo real acerca de automóveis, nomeadamente no que diz respeito à sua construção e inicialização. Transporte para o seu programa e classe tanto quanto possível essas regras de forma que a construção de objetos da classe *Automovel* respeite sempre essas regras.

c) Escreva uma função que permita obter a representação textual do conteúdo da estrutura. Não se pretende necessariamente imprimir essa informação no ecrã e evite faze-lo diretamente em código da função ("obter" ≠ "imprimir").

Objetivos do exercício
(na continuação do anterior)

- Consolidar os conceitos de encapsulamento e compreender as suas vantagens.
- Utilizar os modificadores de visibilidade de C++ para definir o que é público e o que é privado.
- Consolidar o conceito de função membro que é chamada sobre um objeto e tomar conhecimento do ponteiro *this*.
- Entender e consolidar o conceito de funções para aceder e para atualizar dados da classe.

- Consolidar o conhecimento acerca das diferenças e semelhanças entre estrutura e classe.
- Utilização de construtores.
- Consolidar o entendimento acerca das desvantagens associadas às funções membro que interagem diretamente com o utilizador (a evitar) e as vantagens de delegar essa interacção para outras partes do programa mantendo as funções membro apenas a receber ou devolver dados.
- Conhecer a classe *ostringstream* e as suas vantagens na construção de *strings* com aplicação prática na obtenção da representação textual agregada de dados.

7. Pretende-se uma classe chamada *MSG* cujos objetos respeitem as seguintes propriedades:

- Armazenam uma letra e um número inteiro. A letra pode ser fornecida na construção dos objetos. Se não for, será assumida a letra 'x'. O número é um valor gerado automaticamente: cada novo objeto tem o número seguinte ao do objeto anterior. O primeiro objeto tem o número 1.
- Deve existir um mecanismo para obter os dados dos objetos sob a forma de uma *string* com o formato "letra: … número …".
- Sempre que um objeto é criado deve ser automaticamente apresentada a sua informação (formato "criado: letra: … número …").
- Sempre que um objeto deixa de existir deve ser automaticamente apresentada a sua informação no ecrã (formato "terminado: letra: … número …").

>> Este exercício tem um objetivo muito forte a nível de ordem de criação e destruição de objetos e quais as diversas situações que levam a que um objeto seja explicita ou implicitamente criado. A maior parte das alíneas destina-se a mostrar as formas, muitas vezes automáticas, que levam à criação de objetos auxiliares ou temporários e quais os mecanismos e situações envolvidas. Este exercício é extenso, mas de execução simples e é importante que seja levado até ao fim.

a) Construa a classe com as características enunciadas e teste a sua funcionalidade criando dois objetos, *a* e *b*, indicando ao primeiro a letra 'a', e ao seguinte nenhuma letra. Execute o programa de forma a que este, ao terminar, aguarde pelo pressionar de uma tecla. Observe as mensagens no ecrã e relacione a ordem pela qual aparecem com a ordem pela qual os objetos são criados e destruídos.

(1) No construtor explore a sintaxe de lista de inicialização e verifique que é melhor que a atribuição de valores dentro do construtor, a qual não é uma verdadeira inicialização, mas sim um "mero uso".

(2) Experimente inicializar os objetos *a* e *b* com parêntesis e com chavetas e verifique que tem ambas as possibilidades sintáticas à sua disposição.

b) Acrescente à função *main*, a seguir às variáveis que já existem, uma referência com o nome *c* para o objeto que tem a letra 'x'. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.

c) Acrescente à função *main*, a seguir às variáveis que já existem, a seguinte declaração: *MSG d=b;* Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.

d) Acrescente à função *main* a atribuição *main: a = b;* . Execute o programa e confirme que o número e ordem de mensagens não se altera (apenas a o conteúdo relativo ao objeto *a*). Comparando com a alínea anterior, explique porquê e remova a atribuição que tinha acrescentado.

e) Acrescente um *array* de objetos de *MSG* de dimensão dois. É possível indicar a letra inicial aos objetos desse *array*? Verifique a sintaxe disponível para esses casos. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.

f) Remova da classe *MSG* a característica que permite que os seus objetos fiquem com a letra 'x' se nenhuma letra for especificada. Explique porque é que o programa deixa de compilar.

(1) Explore a possibilidade de indicar valores iniciais aos elementos do *array:* as várias alternativas sintáticas ao seu dispor e os seus efeitos através das mensagens que são (ou não) apresentadas no ecrã.

Volte a por a classe como estava (a inicialização com a letra 'x').

g) Acrescente uma função *teste* do tipo *void* e sem parâmetro nenhum. Declare nessa função um objeto classe *MSG* com o nome *aux* e com a letra inicial 'y'. Invoque a função teste na função *main*. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.

h) Modifique a função *teste* de forma a receber como parâmetro um objeto da classe deste exercício. O objeto é passado por valor. Na função *main* adapte a chamada à função teste passando-lhe o objeto que tem a letra 'x'. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.

i) Acrescente um construtor privado à classe *MSG* que recebe como parâmetro o seguinte: *const MSG & z* e dê um corpo vazio a esse construtor. Compile o programa e confirme que não é possível. Explique porquê.

j) Passe o construtor a publico e coloque no seu corpo apenas a seguinte linha *cout << "construído por cópia";* . Verifique que já consegue compilar o programa e explique porque é que já é possível.

k) Execute o programa e repare que os valores relativos ao objeto parâmetro da função *teste* não são inicializados. Explique porquê. Coloque o código adequado ao construtor por cópia de forma a que

os objetos inicializados por cópia fiquem corretamente inicializados, mas o valor numérico fique o simétrico do original (para se distinguirem). Execute o programa e confirme e explique as mensagens que aparecem.

l) Modifique função *teste* de forma a que o seu parâmetro seja passado por referência. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.

m) Modifique a função *teste* de forma a retornar por valor um objeto *MSG* e remova o parâmetro. No corpo da função retorne o objeto *aux*. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.

n) Modifique função *teste* de forma a que o retorno passe a ser por referência. Execute o programa, analise e justifique as mensagens e a ordem pela qual aparecem. Explique as diferenças nas mensagens relativamente à alínea anterior.

o) Escreva no seu caderno todas as conclusões acerca do comportamento da criação e destruição de objetos que observou ao longo deste exercício.

Objetivos do exercício

Explorar os seguintes

- Uso de construtores. Sintaxe no construtor: lista de inicialização vs. mera atribuição dentro do construtor.
- Sintaxe na inicialização de objetos com () e {}
- Uso de dados membros estáticos.
- Ordem de criação e destruição de objetos
- Criação e destruição de objetos em *arrays*
- Inicialização de objetos em *arrays*
- Passagem de objetos por cópia. Retorno de objetos por cópia. Construtor por cópia
- Passagem/retorno de objetos por cópia *vs.* por referência.
- Objetos *const* e funções membro *const*

8. Transporte para o computador em C++ o conceito de *pessoa* com as características (dados e funcionalidade) principais habituais associadas a esta entidade no mundo real. Esta classe deve respeitar os princípios de encapsulamento. Tente incluir o conceito de *constante* na funcionalidade que não se destina a alterar os objetos desta classe. Uma parte importante deste exercício é a modelização sensata e não existem soluções únicas. Este exercício foi planeado para haver interação entre o docente e os alunos. Isto não significa que os alunos devam ficar à espera da solução, pois uma parte importante deste exercício tem a ver com a proposta autónoma de dados e funções. Não existe uma solução única ou oficial. O percurso para atingir a solução é tão importante como a solução em si.

Objetivos do exercício

- Treinar a representação de entidades do mundo real em classes de C++.

- Treinar a identificação de funcionalidade e dados necessários mediante uma especificação em que se indicam as características necessárias e não os dados e funções diretamente. Isto inclui deduzir dados e funções mediante o conhecimento das entidades do mundo real e mediante o bom senso e que podem não ser diretamente mencionados no enunciado.
- Uso de construtores.
- Uso de *const*.

◆ ❖ ◆

9. Pretende-se modelizar televisões em classes de C++. As televisões têm:

- Um conjunto predefinido de canais que conseguem sintonizar. Cada canal corresponde a um nome (o nome desse canal).

As televisões suportam a seguinte funcionalidade:

- Está ligada ou desligada. Pode-se passar de um estado para o outro (ligar / desligar). Note-se que "ligar" ≠ "construir" e "desligar" ≠ "destruir".
- Está num determinado nível de volume. O nível menor é 0 (sem som), e o máximo é 10. Pode-se aumentar e diminuir o som, sempre uma unidade de cada vez. Só se consegue mudar o volume se a televisão estiver ligada.
- Está num determinado canal. Muda-se de canal invocando uma determinada funcionalidade à qual se indica o número de canal pretendido. Se o canal for válido, a televisão envia para o ecrã o nome do canal para o qual passou. Caso contrário, mantêm-se no mesmo canal e nada acontece. Só se consegue mudar de canal se a televisão estiver ligada.
- Deve ser possível obter a representação textual do estado interno da televisão. Se estiver desligada, essa informação corresponde a "deligada". Se estiver ligada, a informação é o nome do canal, o número do canal (o primeiro nome corresponde ao canal 1, o segundo ao 2, etc.) e o volume de som. Note-se que "obter" ≠ "imprimir".

As televisões, quando são construídas, obrigam a que seja definido logo um conjunto de canais e esse conjunto nunca mais se pode alterar.

a) Construa a classe pretendida assumindo que existe uma capacidade máxima de 10 canais. Construa uma função *main* para a testar. Construa várias televisões com diferentes conjuntos de canais. Ligue-as e desligue-as. Mude o volume e de canal. Não fique demasiado preso aos programas desportivos e escreva no caderno as conclusões deste exercício.

b) Modifique a classe assumindo que afinal não existe limite específico quanto ao número de canais. Não se pretende para já uma implementação com memória dinâmica.

Objetivos do exercício

- Treinar o conceito de abstração e correta representação de entidades do mundo real em classes de C++.
- Treinar a identificação de funcionalidade e dados necessários mediante uma especificação em que se indicam as características necessárias e não os dados e funções diretamente. Isto inclui deduzir dados e funções que não são sequer mencionados no enunciado.

- Uso de construtores.
- Uso de *const*.
- Uso da classe *ostringstream*.
- Primeiro contacto com a classe *vector* da *STL*.

◆ ❖ ◆

10. Pretende-se modelizar o conceito de lista de eleitores em C++. Um eleitor é representado por um número inteiro positivo entre 1000 e 9999. Uma lista de eleitores:

- Pode estar vazia
- Não pode ter eleitores repetidos
- Pode ser inicializada com um número arbitrário de eleitores (nenhum ou vários).
- Permite a adição de um ou mais eleitores numa única operação
- Permite a eliminação de um eleitor dado o seu número
- Permite a eliminação de todos os eleitores cujo número esteja entre a e b, sendo a e b indicados.
- Permite obter uma descrição textual que indica o menor número de eleitor, o maior, e o número total de eleitores presentes na lista.

Escreva uma classe que cumpra estes requisitos. Teste-a com uma função *main* à sua escolha. Deve:

- Preocupar-se em primeiro lugar com os requisitos ("o que a classe diz que faz") e só depois com a implementação ("como é que a classe faz").
- Procurar representações de dados que lhe removam trabalho enquanto programador sem, no entanto, prejudicar a robustez ou performance da classe.
- Manter inacessível ao programa a funcionalidade de uso interna à classe (funções auxiliares).
- Explorar as diversas alternativas sintáticas quanto à inicialização dos objetos.

Requisitos de implementação

- Entre os vários construtores (um ou mais), deve existir um que receba dois eleitores. Verifique em que circunstâncias é que ele é chamado caso tenha algum outro que use *initializer_list*.
- Caso a restrição de eleitores repetidos fosse levantada, haveria algum construtor que pudesse ser simplificado? Qual e como?

Objetivos do exercício

- Explorar *initializer_list* no construtor e em funções membro. Verificar as consequências de ter um construtor com *initializer list*.
- Explorar a funcionalidade dos *vectores* de C++
- Explorar iteradores para percorrer e manipular *vectores*
- Ver como se eliminam elementos em *vectores*, quer um elemento de cada vez, como vários.
- Rever e treinar o uso de *ostringstream*

◆ ❖ ◆