Programação Orientada a Objetos 2024/2025

Exercícios

Ficha Nº 2

Namespaces
Overloading
Referências
Parâmetros com valor por omissão
Exceções

1. Assuma que está a fazer um projeto em C++ grande, cujo código está organizado por áreas de funcionalidade, entre as quais as duas seguintes:

- DataStore – Tem a ver com o armazenamento de dados do programa em ficheiros
- UserInterface – Tem a ver com a interação com o utilizador

É necessário definir um mecanismo para verificar a validade dos dados (*string*) que são lidos/introduzidos. Esse mecanismo vai ser uma função global. Segundo as boas práticas de programação, o nome da função descreve o que ela faz ou a pergunta a que ela responde e, portanto, o nome será dadosSaoValidos, com o protótipo seguinte:

```
bool dadosSaoValidos(string).
```

O problema é que são precisas duas destas funções, com o mesmo nome e parâmetros, que fazem coisas diferentes:

- uma para a á área funcional DataStore – a *string* é válida se tiver entre 5 e 10 caracteres.
- outra para a área funcional UserInterface – a *string* é válida se começar por uma maiúscula.

Em ambos os casos o protótipo é o mesmo e isso colide com as regras de *overloading*.

a) Proponha e concretize uma forma de definir ambas as funções sem alterar o nome nem parâmetros.

b) Tendo descoberto que a solução passa por usar *namespaces*, escreva uma função *main* onde utiliza as duas funções

i) Sem usar declaração *using namespace …*
ii) Usando *using* mas não *using namespace…*
iii) Usando um *using namespace …* apenas
iv) Usando dois *using namespaces…*

No final deste exercício deverá garantir que percebeu:

- O que são *namespaces* e quais os seus casos de uso em projetos de C++
- Qual a sintaxe relacionada com *namespaces* e as várias alternativas quanto ao seu uso, nomeadamente às diversas alternativas quanto a *using*.

2. Usando o que ouviu nas aulas teóricas sobre ***overloading***, complementado com uma explicação eventual sobre o assunto nesta aula, e escreva o código necessário para que a seguinte função *main()* se execute sem erros.

```
int main(){
   imprime("programação orientada a objetos");
   imprime("horas por aula teórica ", 2);
   imprime(3, " horas em cada aula prática");
   return 0;
}
```

No final deste exercício deverá garantir que percebeu:

- O que é o *overloading* e qual a sintaxe associada
- Em que situações o *overloading* é vantajoso.
- Quais as restrições ao seu uso e em que casos surgem erros de compilação por ambiguidade.

3. Escreva a função ou funções **multiplica( )** de modo que o programa seguinte corra sem erros.

```
int main(){
   cout << "\n" << multiplica() < "\n" << multiplica(5);
   cout << endl << multiplica(2,3) << endl << multiplica(2,3,4);
}
```

O output to programa deverá ser

```
1
5
6
24
```

a) Resolva o exercício usando apenas o mecanismo de *overloading* (como no exercício anterior)

b) Veja no seu caderno os seus apontamentos sobre a explicação na aula teórica sobre ***parâmetros com valor por omissão***, complementada com alguma informação dada nesta aula e resolva novamente o exercício usando apenas funções com parâmetros com valores por omissão. Durante esta alínea, coloque o código anterior em comentários porque vai ser necessário mais adiante.

c) Remova os comentários do seu código da alínea a) e tente manter no mesmo programa as funções da alínea a) e b). Consegue compilar o programa? Explique a razão do compilador se queixar.

No final deste exercício deverá garantir que percebeu:

- O que são os parâmetros com valor por omissão
- Em que situações é vantajoso
- Quais as restrições ao seu uso: sintáticas, derivadas de criação de situações de ambiguidade, e de uso em simultâneo com *overloading*.
- As restrições quanto à ordem dos parâmetros que têm valores por omissão, na lista de parâmetros das funções.

4. Pretende-se uma função que troque os valores de duas variáveis inteiras pertencentes ao *contexto de onde a função é chamada* (pertencem ao código de quem chama e não à função que é chamada).
Exemplo:

```
int main(){
   int a = 5, b = 10;
   troca(a, b);
   cout << "\na = " << a << "\nb = " << b;
} // deve aparecer a = 10 e b = 5
```

a) Escreva a função pretendida utilizando apenas o que conhece da linguagem C.

b) Se tiver usado ponteiros, explique por que razão são necessários os ponteiros.

c) Usando que ouviu acerca de ***referências*** na teórica, complementada com eventual informação dada nesta aula escreva a função pretendida sem usar ponteiros.

d) Experimente usar valores literais na chamada à função. Experimente usar *const* nos parâmetros da função (experimente uma destas alternativas de cada vez, e depois em simultâneo). Extraia daí as conclusões quando à sintaxe e regras de uso de referências.
Compare ponteiros com referências a nível de funcionamento, a nível de sintaxe, e a nível de restrições de uso. Visualize a localização dos dados e variáveis na memória do computador. Faça diagramas para o ajudar na visualização.

No final deste exercício deverá garantir que percebeu:

- O que são as referências (por agora, apenas do tipo ***lvalue***)
- Passagem parâmetros de por referência e a diferença para passagem de parâmetros por cópia.
- Em que situações é vantajoso o uso de referências, e entendido os exemplos mostrados de passagem de parâmetros por referência e retorno de funções por referência

- Quais as restrições ao seu uso, nomeadamente ***quando ao uso de constantes ou valores literais*** nas chamadas às funções que têm parâmetros do tipo referência, ou no retorno de funções por referência.
- Quais as situações em que podem substituir o uso de ponteiros e quais as situações em que não podem substituir ponteiros (e que continua a ter que se usar ponteiros), nomeadamente quando à ***impossibilidade de mudar a referência para referir uma outra variável***.

5. Pretende-se uma função selecione uma das variáveis que lhe são passadas por parâmetro, de acordo com um código (também por parâmetros) e que permita a variável selecionada ser usada em contextos de atribuição de novos valor. O exemplo seguinte esclarece melhor.

Exemplos

Selecionar a menor das variáveis, atribuindo-lhe o valor 0

```
int main(){
   int a = 5, b = 10;
   seleciona(a, b, 'm') = 0;
   cout << "a = " << a << " b = " << b;   // aparece 0 10
}
```

Selecionar a maior das variáveis, diminuindo o seu valor em 3

```
int main(){
   int a = 5, b = 10;
   seleciona(a, b, 'M') -= 3;
   cout << "a = " << a << " b = " << b;   // aparece 5 7
}
```

O último parâmetro indica qual a variável a selecionar.

m -> selecionar a menor

M -> selecionar a maior

p -> selecionar a primeira

u -> selecionar a última

A indicação de um código desconhecido deve gerar uma exceção.

No final deste exercício deverá:

- Perceber os aspetos envolvidos no retorno de referências
- Quais as restrições relativas ao uso de referências relacionadas com a duração da variável referida, com particular aplicação no caso de retorno de referências
- Exceções: o que são, objetivos, casos de uso no programa, situações onde se deve lançar exceções e situações onde não se devem lançar exceções. Lançamento, propagação e captura de exceções.
- Sintaxe relacionada com exceções: try, catch, throw. Funções noexcept, operador noexcept

6. Pretende-se uma estrutura chamada Tabela, que contenha uma matriz de inteiros. O número de inteiros é conhecido à partida e durante a compilação (use *const* e não um #define para o definir). Neste exercício a função *main()* é assessória, definida pelo aluno, servindo apenas para testar a funcionalidade.

a) Escreva a estrutura pretendida e duas funções globais para:

- Preencher toda os valores da matriz da estrutura com um valor especificado.
- Listar o conteúdo da estrutura.

b) Escreva duas globais funções para

- 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.
- 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.

Importante: esta funções não interagem diretamente com o ecrã/teclado.

c) Para cada uma das funções da alínea anterior, considere o uso de exceções para a proteção de dados inválidos. Numa delas faz mais sentido usar exceções, enquanto que na outra faz mais sentido não usar e escolher uma forma alternativa. Identifique qual é qual, justificando e confirmando com o professor do laboratório (este exercício só funciona se participar na aula).

d) Unifique a funcionalidade das funções da alínea b) 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, devendo experimentar duas estratégias:

(1) Sem usar exceções

(2) Usando exceções

A análise feita na alínea c) continua a aplicar-se a esta, mas pretende-se que experimente ambas as estratégias com e sem exceções

```
int main(){
   Tabela a;                    // a dimensão é de 20 elementos
   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
}
```

e) Invoque a versão da função que foi protegida através do mecanismo de exceções a partir de uma função qualquer testa(), por sua vez chamada a partir da função main(). Experimente várias combinações na forma de chamada à função (dados válidos e captura/não-captura de exceção) de forma a ver o mecanismo de propagação de exceções a atuar.

No final deste exercício deverá garantir que percebeu:

- Referências: todas as questões relacionadas com retorno de referência, nomeadamente restrições ao seu uso.
- Exceções: ter visto e experimento em primeira mão um caso de aplicação de exceções e ter experimentado o mecanismo de tratamento e de propagação de exceções
- Deve ter adquirido uma primeira noção acerca de onde devem estar os mecanismos de interação com o utilizador e onde não devem estar esses mecanismos