Programação Orientada a Objectos 2022/2023

Exercícios

Ficha Nº 1

Input/Output simples
Objetos string
Referências
Overloading
Parâmetros com valor por omissão
Namespaces

0. Verifique que o seu ambiente de compilação está a funcionar escrevendo o programa "Hello World" indicado abaixo. Este programa contém aspetos que não irão ser explicados todos já, e serve apenas para confirmar que o seu IDE está a funcionar.

```
#include <iostream>
using namespace std;
int main() {
    cout<<"Hello World";
    return 0;
}
```

O programa deverá apresentar "Hello World".

Se não tiver um ambiente de desenvolvimento a funcionar:

- Terá que instalar um em tempo fora de aulas, segundo as indicações dadas nas aulas teóricas.
- Como desenrasque, nesta aula use um compilador *online* (não serve para aulas mais adiante nem para o trabalho prático).

  https://www.onlinegdb.com/online_c++_compiler

Objetivos do exercício

- Garantir que tem as ferramentas necessárias para trabalhar, mostrando ao professor um IDE para desenvolvimento em C++ instalado e configurado (CLion, VSCode, outros)

1. Pretende-se um programa que peça o nome e a idade do utilizador e depois imprima essa informação.

a) Implemente o programa pretendido utilizando apenas o que conhece da linguagem **C** e as funções biblioteca C standard. Pretende-se apenas uma solução "habitual" parecida com e representativa das que normalmente usaria nos seus programas. O nome pode ser só uma palavra, bastando identificar o que seria necessário fazer se fossem mais do que uma palavra (diga como faria).

b) Identifique os pontos fracos no código que podem dar origem a valores errados e *runtime errors*. Explique as fraquezas que identificou.

- Procure especificamente situações que podem corromper informação ou até fazer o programa terminar.
- É importante que perceba quais são os pontos fracos (e porquê) de forma a, mais adiante, comparar com a implementação melhor em C++ e perceber como estas situações são evitadas em C++. Verifique junto do professor que identificou todas as situações corretamente.

No final deste exercício deverá

- Ter uma noção muito clara acerca dos pontos fracos da linguagem C no que diz respeito à possibilidade de cometer sem bugs sem que o compilador os detecte, senso o uso de scanf/printf um exemplo

Objetivos do exercício

- Entender os pontos fracos da utilização de funções *printf* e *scanf* decorrentes da ausência de validação por parte do compilador nos tipos de dados dos parâmetros destas funções.

2. Escreva um programa em C++ que peça o nome e a idade do utilizador e depois imprima essa informação. Utilize os objetos da biblioteca standard C++ que existem especificamente para input/output indicados nas aulas. Utilize os tipos de dados que já conhece da linguagem C para armazenar os dados introduzidos pelo utilizador, mas utilize objetos de classes standard de C++, explicados pelo professor e tal como indicado nas alíneas seguintes:

a) Escreva o programa pretendido em C++. Não pode utilizar funções *scanf* nem *printf* nem nenhuma outra função da biblioteca C *standard*. Em vez disso, deve usar os objetos **cin** e **cout** da biblioteca C++, conforme a explicação e exemplos apresentados pelo professor no início do exercício.

b) Compare a solução com o código do exercício 1 e explique porque é que a versão em C++ é mais robusta em termos de compatibilidade de tipos de dados e menos propensa a *runtime errors*. Não deverá avançar enquanto não tiver a certeza que compreendeu a razão pela qual esta implementação é melhor que a do exercício 1.

c) Volte ao código da alínea a) e garanta que i) o valor inteiro introduzido é mesmo um inteiro e com valor positivo (se não for, o programa deve pedir o valor novamente), e que ii) consegue ler mais do que uma palavra para o nome. Na validação deve usar os mecanismos existentes nos objetos da biblioteca C++ que usou para efetuar a entrada de dados e apenas esses (***cin***).

No final do exercício deverá perceber, de forma clara e concreta, os seguintes tópicos

- O que são *cin*, *cout*, e *cerr*, e o que são e significam << e >> nesse contexto.
- Como *cin* e *cout* são um exemplo de como a linguagem C++ evita a necessidade de código propenso a erros.
- Formas simples de uso de cin e de validação de dados de entrada.

Importante: A partir deste exercício e durante o resto do semestre deixa de poder usar as funções *printf*, *scanf* e similares. Esta "proibição" aplica-se às aulas, trabalho prático e exames.

- A justificação para esta "proibição" deve ser óbvia com base no que aprendeu na alínea b). Se restarem dúvidas nesse aspeto tire já essa dúvida com o professor antes da aula terminar.

Objetivos do exercício

- Tomar conhecimento e entender como se usam os objetos ***cin*** e ***cout***. Neste primeiro contacto não se pretende ainda o domínio completo destes objetos, mas apenas o necessário para interagir com o utilizador usando os tipos de dados *built-in*.
- Entender o conceito de *namespaces* e da declaração *using*. Tomar conhecimento da biblioteca *iostream* e da declaração *#include <iostream>*
- Entender a razão pela qual a utilização de *cin* e *cout* é intrinsecamente mais segura que os seus análogos *scanf* e *printf* da linguagem C.

3. Escreva um programa em C++ que peça o nome e a idade do utilizador e depois imprima essa informação. Não pode utilizar funções *scanf* nem *printf*. Também não pode usar *arrays* de caracteres.

A partir deste exercício deverá ficar muito claro que usar *arrays* de caracteres para guardar cadeias de caracteres como se fazia em linguagem C é bastante desvantajoso e fortemente desaconselhado. Este "desaconselhamento" aplica-se às aulas, TP e exames. O exercício serve para perceber o porquê.

a) Escreva o programa pretendido em C++ com as restrições que foram enunciadas. Sugestão: utilize objetos da classe *string*.

b) Compare a solução com o código do exercício 2 e identifique as vantagens e desvantagens da utilização dos objectos *string* face às matrizes de caracteres. Explique porque é que a versão usando objectos *string* é menos propensa a erros de memória relacionados com a ultrapassagem de limites de arrays

c) Acrescente ao programa a seguinte funcionalidade: i) imprimir o número de caracteres do nome. Ii) imprimir uma letra do nome em cada linha (utilize um for tradicional e depois o "for-each" de C++, mediante a explicação dada acerca deste tipo de ciclo).

No final deste exercício deverá ter ficado coma saber

- O que é a classe *string* e para que serve.
- Quais as funções membro principais de classe *string*.
- Como aceder a caracteres individuais de uma *string*, o número de caracteres, como comparar *string*, como mudar o seu valor, como percorrer os caracteres de uma *string*.
- Como obter informação acerca das funções membro das classes biblioteca de C++

Objetivos do exercício

- Tomar conhecimento e entender como se usam os objetos *string*. Neste primeiro contacto pretende-se apenas que se consiga armazenar e posteriormente obter cadeias de caracteres através desses objetos. A manipulação caracter a caracter fica para outro exercício.
- Tomar conhecimento da ajuda *on-line* e usá-la para obter pormenores acerca da classe *string*.
- Entender a diferença entre matrizes de caracteres de tamanho fixo com as restrições e precauções inerentes às matrizes, e os objetos *string* que gerem automaticamente o seu espaço interno para armazenamento de caracteres.
- Entender as vantagens e desvantagens de utilização de objetos *string* em vez de matrizes de caracteres de tamanho fixo para interação com o utilizador.

4. Oiça com atenção a explicação acerca do **mecanismo de *overloading*** que o professor fez na aula e escreva o código necessário para que a seguinte função *main* se execute sem erros.

```
int main(){
   imprime("ola");
   imprime("a idade é: ", 25);
   imprime(100, "euros");
   return 0;
}
```

No final deste exercício deverá garantir que percebeu:

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

Objetivos do exercício

- Entender e usar a característica de *overloading* da linguagem C++ e as situações onde se usa.
- Entender as restrições sintácticas de *overloading* e as situações de ambiguidade a evitar.

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

```
int main(){
   cout << "\n" << soma() << soma(1);
   cout << soma(1,2) << soma(1,2,3);
}
```

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

b) Oiça com atenção a explicação acerca de **parâmetros com valor por omissão** que o professor fez na aula e utilize agora apenas funções com parâmetros com valores por omissão na resolução.

c) Tente manter no mesmo programa as funções da alínea a) e b) em simultâneo. 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.

Objetivos do exercício

- Consolidar conhecimento acerca da característica de *overloading* da linguagem C++.
- Entender e usar a característica de parâmetros com valor por omissão da linguagem C++.
- Determinar quando usar *overloading*, parâmetros por omissão, ou ambos.
- Entender as restrições inerentes ao uso simultâneo de *overloading* e parâmetros por omissão.

6. Pretende-se uma função que troque os valores de duas variáveis inteiras pertencentes ao *contexto de chamada* (pertencem ao código que 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) Oiça com atenção a explicação acerca de **referências** que o professor fez na aula e escreva a função pretendida utilizando parâmetros do tipo referência.

d) 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 é a passagem de parâmetros por cópia
- O que são as referências (por agora, apenas referências ***lvalue***)
- Passagem parâmetros de por referência e a diferença para passagem de parâmetros por cópia.
- Em que situações é vantajoso o seu uso, 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***.

Objetivos do exercício

- Entender e usar a característica de referências da linguagem C++ e o modo como estas funcionam.
- Compreender as semelhanças e as diferenças entre referências e ponteiros.
- Entender as restrições sintáticas associadas ao uso de referências.
- Entender as situações onde usar e as situações onde não usar referências.

7. Escreva um programa em C++ que peça o nome completo do utilizador e depois imprima os vários nomes desse utilizador, cada um numa linha diferente. Se um dos nomes do utilizador for “Silva” o programa deve avisar que conhece alguém com esse nome.
Requisito de implementação: a linha é lida de uma só vez. Depois de lida, então é processada.

a) Usando apenas cin >> *objecto-de-string*

b) Usando um objeto **istringstream**

No final do exercício deverá

- Ter uma noção cerca de como funciona o buffer de caracteres associado à entrada de dados e como são consumidos por cin >>
- Ter tido uma primeira introdução à classe istringstream e como esta pode ajudar a controlar a forma como os caracteres são consumidos (analogia: sscanf)

Objetivos do exercício

- Perceber como os caracteres podem ficar armazenados num buffer interno e serem apenas consumidos numa leitura subsequente
- Perceber como se pode ler uma linha de uam só vez para análise subsequente
- Treinar o processamento de *strings* contendo espaços e extrair palavras individuais.
- Experimentar a comparação de objetos *string*.

8. Escreva um programa em C++ que peça palavras ao utilizador. Após cada palavra lida, o programa escreve essa palavra ao contrário no ecrã. Se a palavra for um palíndromo (fica igual ao original quando escrita ao contrário), o programa escreverá à frente "palíndromo". Antes de proceder a nova leitura de palavra o programa deve apresentar a mensagem "carregue em *enter* para prosseguir" e aguardar que seja premida essa tecla. O programa termina quando é escrita a palavra "fim".

Objetivos do exercício

- Treinar o processamento de *strings* caracter a caracter.
- Treinar métodos de efetuar pausas através de objectos *cin*.
- Consolidar competências de consulta da ajuda online e do IDE acerca das bibliotecas C++.

9. Escreva um programa que leia texto a partir do teclado. Se o utilizador escrever um número por extenso ("um", "dois"…), o programa responderá imprimindo esse número em decimal (1,2…). Se o utilizador tiver escrito um número em formato decimal, então o programa responderá com esse mesmo número por extenso. Lide apenas com números entre 1 e 10 e ignore tudo o que não for número. Antes de passar ao próximo número o programa aguarda que se carregue em *enter*. O programa termina quando for escrita a palavra fim.

Objetivos do exercício

- Treinar a leitura de informação de diversos tipos de dados a partir do teclado.
- Conseguir detetar o tipo de informação introduzido (exemplo: texto ou inteiro) e agir em conformidade.
- Tomar conhecimento e usar a classe *istringstream*.
- Treinar a conversão de cadeias de caracter para inteiros usando *instringstream*.

10. Escreva um programa que leia um número por extenso ("um", "dois"…) e depois um número inteiro. O programa deverá verificar se o número por extenso corresponde ao número inteiro e indicar a palavra "certo" ou "errado" consoante o caso. De seguida procede a nova leitura de número por extenso / inteiro, sem qualquer pausa. O programa termina quando é escrita a palavra "fim" em vez de um número por extenso.

Objetivos do exercício

- Treinar a leitura de informação de diversos tipos de dados a partir do teclado.
- Treinar a conversão de cadeias de caracter para inteiros.
- Lidar com situações em que é necessário controlar a entrada de dados, por exemplo, limpar o *buffer*.

11. Assuma que está a fazer um projeto grande, organizado por áreas funcionais, entre as quais as duas seguintes:

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

É necessário definir um mecanismo (uma função) para verificar a validade dos dados (uma string) lidos/introduzidos. Segundo a boa norma que diz que o nome de uma função deve ser algo que descreve o que a função faz, o seu gestor de projeto determinou que a função deve ter o protótipo

bool dadosSaoValidos(string).

O problema é que são precisas duas destas funções e 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 de começar por uma maiúscula.

mas 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…*

Objetivos do exercício

- Treinar o uso de *namespaces*: cenários de aplicação e sintaxe
- Rever a classe *string*