

Aula prática 2

Os exercícios 1 a 6 têm como objetivo aplicar conhecimentos básicos de apontadores e referências. Os restantes exercícios têm como objetivo introduzir conceitos relacionados com as classes em C++.

Consulte a respetiva pasta incluída em **P02.zip**, disponível no Moodle.

Os exercícios 3, 4, 6, 7 e 8 podem fazer no moodle, com o CodeRunner.

1 - Considere o programa pointers.cpp. O programa tem como objetivo a familiarização com os conceitos de apontadores e referências. Para isso está dividido em três alíneas em que se passa o parâmetro das funções de forma diferente sendo estas formas: por valor, por referência com argumentos apontadores e por referência com argumentos de referência. Em cada alínea terá de realizar 3 ações em cada função e ver o resultado.

- a) Altere a função `square_by_value` para que:
 - i) imprima o endereço de `n`
 - ii) calcule o quadrado de `n`
 - iii) imprima o valor de `n`. Interprete o resultado.
- b) Altere a função `square_by_reference_point_args` para que:
 - i) imprima o endereço de `pN`
 - ii) calcule o quadrado de `pN`
 - iii) imprima o valor de `pN`. Interprete o resultado.
- c) Altere a função `square_by_reference_ref_args` para que
 - i) imprima o endereço de `rN`
 - ii) calcule o quadrado de `rN`
 - iii) imprima o valor de `rN`. Interprete o resultado.

Depois de implementar as alíneas, o programa deverá apresentar um resultado semelhante ao exemplo:

```
----- Testing the square_by_value function -----
In main(): the number value is 8 and its address is 0x7ffffcc2c
In square_by_value(int n) its address is 0x7ffffcc00 and its value after
modifying is 64
Its value in main after calling square_by_value(number) is 8

----- Testing the square_by_reference_point_args function -----
In main(): the number value is 8 and its address is 0x7ffffcc2c
In square_by_reference_point_args(int *pN): its address is 0x7ffffcc2c and
its value after modifying is 64
Its value in main after calling square_by_reference_point_args(&number) is
64

----- Testing the square_by_reference_ref_args function -----
In main(): the number value is 8 and its address is 0x7ffffcc2c
In square_by_reference_ref_args(int &rN): its address is 0x7ffffcc2c and
its value after modifying is 64
Its value in main after calling square_by_reference_ref_args(number) is 64
```


2 - Considere o programa `functions_ref.cpp`. Semelhante ao exercício anterior, este exercício tem como objectivo a familiarização com os conceitos de apontadores e referências. Para isso está dividido em três alíneas em que se passa o parâmetro das funções de forma diferente bem como se devolve de forma diferente.

- a) Altere a função `square_reference` para que: i) imprima o endereço de `rN`, ii) calcule o quadrado de `rN` e iii) imprima o valor de `rN`. Interprete o resultado.
- b) Altere a função `square_pointer` para que: i) imprima o endereço de `pN`, ii) calcule o quadrado de `pN` e iii) imprima o valor de `pN`. Interprete o resultado.

Depois de implementar as alíneas, o programa deverá apresentar um resultado semelhante ao exemplo:

```
----- Testing the square_reference function -----
In main(): the number value is 8 and its address is 0x7ffffcc1c
In square_reference(int &rN) its address is 0x7ffffcc1c and its value
after modifying is 64
The number value in main after calling square_reference(number) is 64
The result value in main after calling square_reference(number) is 64 and
its address is 0x7ffffcc1c


----- Testing the square_pointer function -----
In main(): the number value is 8 and its address is 0x7ffffcc1c
In square_pointer(int *pN): its address is 0x7ffffcc1c and its value after
modifying is 64
The number value in main after calling square_pointer(&number) is 64
The pointer result value in main after calling square_pointer(&number) is
64 and its address is 0x7ffffcc1c
```

3  - Considere o programa `doubles.cpp`. Construa 3 funções:

- a) `doublePointers((int* matrix, int lines, int col)` – função que recebe uma matriz bidimensional como apontador e calcula o dobro de cada elemento da matriz.
- b) `doubleReference ((int (&matrix)[3][3], int lines, int col)` – função que recebe uma matriz bidimensional como referência e calcula o dobro de cada elemento da matriz.
- c) `print(int (&matrix)[3][3], int lines, int col)` - função que recebe uma matriz bidimensional como referência e imprime os elementos por linha.

Depois de implementar as alíneas, o programa deverá apresentar um resultado semelhante ao exemplo:

```
Matrix:
[ 1 2 3 ]
[ 4 12 6 ]
[ 7 8 9 ]
Matrix double using the function with pointers is:
[ 2 4 6 ]
[ 8 24 12 ]
[ 14 16 18 ]
Matrix double using the function with reference is:
[ 4 8 12 ]
[ 16 48 24 ]
[ 28 32 36 ]
```

- 4  - Considere o programa person.cpp. Construa 2 funções para preencher uma estrutura:
- a) modifyByPointe(Person* person,string name, int age)r – função que recebe a estrutura como apontador e preenche os elementos da estrutura.
 - b) modifyByReference(Person& person,string name, int age) – função que recebe a estrutura por referência e preenche os elementos da estrutura.

Depois de implementar as alíneas, o programa deverá apresentar um resultado semelhante ao exemplo:

```
Name: João, Age: 22
Name: Pedro, Age: 34
return: -1
```

- 5 - Dado o seguinte código, sem compilar nem executar tente perceber o que faz o código e comente nos locais com //.

```
#include <iostream>
using namespace std;


//
int function1(int (&arr)[5], double& x) {
    int y = 0;
    //
    for (int i = 0; i < 5; ++i) {
        y += arr[i];
    }
    //
    x = y / 5.0;
    return y;
}

int main() {
    int numeros[5] = {1, 2, 3, 4, 5};
    double a;
    int b;

    //
    b=function1(numeros, a);

    cout << "Resultado 1 dos elementos: " << b << endl;
    cout << "Resultado 2 dos elementos: " << a << endl;

    return 0;
}
```


6  – Dado uma estrutura **Person** e um vetor dessas estruturas, construa uma função `import(const string name_file, Person (&v)[10])`, que preencha o vetor com a informação dada num ficheiro. Esse ficheiro lista.txt tem em cada linha um nome de um pessoa, a sua idade e o seu país. Esses dados estão separados por virgula.

Dica: utilize o sstream (https://cplusplus.com/reference/sstream/stringstream/#google_vignette) e o

`getline (istream& is, string& str, char delim);`

Depois de implementar a função, o programa deverá apresentar um resultado semelhante ao exemplo:

```
Name= Maria Joana--- Age= 12--- Country= Portugal
Name= Luis Teixeira--- Age= 18--- Country= Espanha
Name= Francisco Matos--- Age= 23--- Country= Portugal
Name= Bruno Gouveia--- Age= 12--- Country= França
Name= Susana Silva--- Age= 45--- Country= Portugal
Name= Pedro Rocha--- Age= 5--- Country= Espanha
Name= João Monteiro--- Age= 67--- Country= Espanha
Name= Filipe Melo--- Age= 56--- Country= Italia
Name= Ana Sousa--- Age= 16--- Country= Italia
Name= Tiago Lima--- Age= 13--- Country= França
return: -1
```

7.  Considere o ficheiro Point2d.h que contém a definição da classe Point2d, que representa pontos em 2 dimensões do tipo double:

```
class Point2d {
public:
    Point2d();
    Point2d(const Point2d& p);
    Point2d(double a, double b);
    double get_x() const;
    double get_y() const;
    void set_x(double p);
    void set_y(double p);
    void translate(const Point2d& t);
    double distance_to(const Point2d& p) const;
private:
    double x;
    double y;
};
```


Escreva o código para as funções membro `translate()` e `distance_to()` no ficheiro `Point2d.cpp`, assumindo que:

- `a.translate(t)` altera `a` com uma translação dada por `t`, i.e., se `a` tem as coordenadas `(x,y)` inicialmente, então deverá ter as coordenadas `(x + t.x, y + t.y)` após a função.

- `a.distance_to(b)` devolve a distância Euclidiana entre `a` e `b`.
-

Depois de implementar as alíneas, o programa deverá apresentar um resultado semelhante ao exemplo:

```
0.70 2.60
1.41
0.00 0.00 0.00
1.50 1.00 1.50
0.00 2.50 2.69
```

8.  Neste exercício pretende-se criar um programa de gestão de alunos em C++, utilizando o conceito de classes. O programa terá o nome GAC (Gestão de Alunos com Classes) e a introdução de dados para teste do programa poderá ser feita com recurso à edição da rotina `main()`. É possível encontrar dentro da pasta `ex1` um esqueleto de implementação, que deverá ser complementado.

- a) Pretende-se que qualquer objeto-instância da classe `Aluno` seja criado com um nome. Acrescente à classe `Aluno` um construtor que inicialize o atributo `nome` com o argumento que lhe for passado e que inicialize todos os outros atributos com valores nulos (0 para números, e "" para strings). O protótipo será:

```
Aluno(string nom);
```

Implemente também um outro construtor que inicialize todos os atributos com os valores fornecidos como argumentos (exceto `media`). Note que isto constitui um exemplo de sobrecarga de funções. O protótipo será:

```
Aluno(string nom, string cur, int num);
```

- b) Um aluno, durante o tempo de estudante, poderá mudar de curso e número algumas vezes, mas nunca de nome, e a sua média também poderá sofrer alterações. Escreva os membros-função que permitam efetuar as atribuições ou alterações permitidas (e.g. `setCurso(string cur)`) e escreva também as restantes funções de acesso típicas (e.g. `float getMedia()`).

- c) Implemente o membro função

```
void imprimir (ostream & os) const;
```

que imprime na *stream* de saída `os`, que poderá ser um ficheiro ou o ecrã, as informações de um determinado aluno, obedecendo ao seguinte formato:

```
<nome> | <curso> | <numero> | <media>
```

- d) Implemente o membro-função

```
string sigla() const;
```

que retorna a sigla de um determinado aluno, formada pelas iniciais do seu nome.

Por exemplo, o aluno Pedro Antunes Rocha terá a sigla “PAR”.

Depois de implementar as alíneas, o programa deverá apresentar um resultado semelhante ao exemplo:

Lista de alunos: (nome	curso	numero	media)	
Pedro Ferreira	Medicina	2020123		10.3
Ana Martins	Direito	2022045		18.2
Manuel Silva	Arquitetura	2021033		16
Alberto Mateus	Engenharia	2019197		0

PF
AM
MS

9. Neste exercício pretende-se escrever um programa que faça a gestão de animais de um Jardim Zoológico. Para já, considere a classe `Animal`, com os membros-dado indicados na estrutura presente no ficheiro `animal.h` da pasta `animal`.

- a) Como é possível observar, a classe `Animal` contém um membro que é um objeto de outra classe, `Veterinario`. Defina esta classe, cujos membros-dado são nome e especialidade, ambos do tipo `string` (exemplos de especialidade: mamíferos, répteis, aves, ...) e com dois construtores:

```
Veterinario (string nom, string esp);  
Veterinario ();
```

- b) Acrescente à definição da classe `Animal` dois construtores:

```
Animal (string esp, boolean sex, const char * nom, int id);  
Animal (string esp, boolean sex, const char * nom, int id, Veterinario vet);
```

Como deverá preencher o atributo `Animal::veterinario` para o primeiro construtor?

- c) Acrescente à classe `Animal` as adequadas funções de acesso: “get...() const” e “set...()”.
- d) Acrescente à classe `Animal` um construtor de cópia;
- e) Suponha, agora, que se desejava atribuir a cada animal criado, da classe `Animal`, um número identificador que fosse único (para todo o programa que usa a classe). Mediante a utilização do conceito de membros estáticos (modificador `static`), altere a definição de `Animal`, e altere todas as alíneas anteriores que julgue necessário, por forma a inserir o referido atributo identificador e se poder saber, numa dada ocasião de execução do programa de teste, o número de animais instanciados.

Depois de implementar as alíneas, o programa deverá apresentar um resultado semelhante ao exemplo:

```
Informação acerca de kurika  
Animal: cao, 1, kurika, 10; Responsavel:  
Informação acerca de bobo
```

```
Animal: morcego, 0, bobo, 2; Responsavel: rui silva
Informação acerca de banny
Animal: urso, 1, banny, 6; Responsavel: carlos macedo
Informação acerca de bobo
Animal: morcego, 0, bobo, 2; Responsavel: rui silva
```

10. Este exercício foca-se na classe BankAccount e contém três ficheiros:

bank_account.cpp, bank_account.h e batest.cpp.

bank_account.h é o ficheiro de definição da classe BankAccount; bank_account.cpp é onde os métodos de classe são desenvolvidos (este é o único ficheiro no qual deverá programar); batest.cpp é o ficheiro principal que contém todos os testes da classe BankAccount e a definição main.

A classe contém os seguintes parâmetros privados:

```
int accountNumber;
string accountHolderName;
double balance;
```

E contém os seguintes métodos públicos:

```
BankAccount(int accNum, string accHolder, double bal);
int getAccountNumber();
void setAccountNumber(int accNum);
string getAccountHolderName();
void setAccountHolderName(string accHolder);
double getBalance();
void setBalance(double bal);
void deposit(double amount);
void withdraw(double amount);
```

a) Complete os seguintes métodos da classe BankAccount (programe e complete apenas o ficheiro bank_account.cpp, onde estão assinaladas as alíneas):

Construtor (BankAccount(int accNum, string accHolder, double bal)):

Este é o construtor da classe BankAccount.

Parâmetros:

accNum (int): Número da conta a ser definida para a conta bancária.

accHolder (string): Nome do titular da conta a ser definido para a conta bancária.

bal (double): Saldo inicial a definir para a conta bancária.

Retorna: Nenhum.

Objetivo: inicializa o objeto BankAccount com o número da conta, nome do titular da conta e saldo fornecidos.

b) Métodos getter para cada atributo:

```
int getAccountNumber() const
string getAccountHolderName() const
double getBalance() const
```

Métodos getter para obter os valores das variáveis dos parâmetros privados.

Parâmetros: Nenhum.

Retorna:

getAccountNumber(): Retorna o número da conta bancária (int).
getAccountHolderName(): Retorna o nome do titular da conta (string).
getBalance(): Retorna o saldo da conta bancária (duplo).

Objetivo: Estes métodos dão acesso aos parâmetros privados accountNumber, accountHolderName e balance.

c) Métodos setter:

```
void setAccountNumber(int accNum)
void setAccountHolderName(string accHolder)
void setBalance(double bal)
```

Métodos setter para atualizar os valores dos parâmetros privados.

Parâmetros:

setAccountNumber(int accNum): Novo número de conta a ser definido (int).
setAccountHolderName(string accHolder): Novo nome do titular da conta a ser definido (string).
setBalance(double bal): Novo saldo a ser definido (double).

Retorna: Nenhum.

Objetivo: Estes métodos permitem a modificação dos parâmetros privados accountNumber, accountHolderName e balance.

d) Um método para depositar dinheiro na conta.

```
void deposit(double amount)
```

Este método é usado para depositar dinheiro na conta bancária.

Parâmetros:

amount (double): Quantidade de dinheiro a depositar.

Retorna: Nenhum.

Objetivo: Adiciona o valor especificado ao saldo atual da conta bancária.

e) Um método para retirar dinheiro da conta (que deve garantir saldo suficiente).

```
void withdraw(double amount)
```

Este método é usado para levantar dinheiro da conta bancária.

Parâmetros:

amount(double): Quantidade de dinheiro a levantar.

Retorna: Nenhum.

Objetivo: Retirar o valor especificado do saldo atual da conta bancária se houver fundos suficientes. Caso contrário, imprime uma mensagem que indica saldo insuficiente.

Para compilar o programa de testes com a biblioteca:

```
g++ -Wall -o ba.o batest.cpp bank_account.cpp
```

Para correr: ./ba.o

Depois de implementar as alíneas, o programa deverá apresentar um resultado semelhante ao exemplo:

```
Account Number: 123456
Account Holder Name: John Doe
```


Balance: \$1000
Updated Account Number: 654321
Updated Account Holder Name: Jane Smith
Updated Balance: \$2000
Deposited: \$500
Current balance after deposit: \$2500
Withdrawn: \$200
Current balance after withdrawal: \$2300
Insufficient balance.
Current balance after attempted withdrawal: \$2300

11. Dado a seguinte classe, responda às perguntas .

```
class User {  
    // Porque é que os atributos estão em private ?  
private:  
    string username;  
    string name;  
    string country;  
    vector<int> favoriteGenres;  
    vector<TVSerie&> watchedSeries;  
    vector<int> rating;  
    vector<int> episodesWatched;  
public:  
  
    // Que função é esta? O que faz?  
    User(string uname,string completename, string usercountry, vector<int>  
    genres) {}  
    // Estas 3 funções servem para ...  
    void setUsername(string user) {}  
    void setName(string name) {}  
    void setCountry(string country) {}  
  
    // Para que serve ter o const nestas funções ?  
    string getUsername() const {}  
    string getName() const {}  
    string getCountry() const {}  
    vector<int> getFavoriteGenres() const {}  
    vector<TVSerie&> getWatchedSeries() const {}  
    vector<int> getRating() const {}  
    vector<int> getEpisodesWatched() const {}  
    void addFavoriteGenre(string genre) {}  
    void addWatchedSeries(TVSerie& series) {}  
    void addRating(TVSerie Tv,float rating) {}  
    void addEpisodesWatched(TVSerie Tv,int n) {}  
}
```