Linguagens de Programação

Caderno de exercícios

Editado por

Filipe Rodrigues
Raquel Bernardino

Instituto Superior de Economia e Gestão
2024

Guião 1

1. *Hello world!*

Escreva o seu primeiro programa em C++ que imprima no ecrã a mensagem: "*Hello world!*".

2. Output

Sem usar variáveis, escreva um programa que imprima no ecrã o seguinte:

```
Total = 100%
IVA   =  17%
IRS   =  15%
--------------
Liq.  =  68%
```

3. Input e Output

Escreva um programa que peça ao utilizador o seu primeiro e último nome e a sua idade e escreva uma mensagem com esses dados.

Exemplo: Se o primeiro nome do utilizador for "Pedro" e o último for "Costa" e se ele tiver "21" anos, o programa deve apresentar a seguinte mensagem:

```
Ola Pedro Costa, bem-vindo.
Que bom que e ter 21 anos.
```

4. Concatenação de *strings*

Escreva um programa que leia três nomes. Crie uma variável que contenha a concatenação dos três nomes (com espaços entre os nomes) e imprima-a no ecrã.

5. Primeira aritmética

Escreva um programa que peça ao utilizador um número inteiro e outro decimal. Crie três novas variáveis para guardar a soma, subtração e multiplicação dos valores introduzidos e imprima no ecrã uma mensagem com essas variáveis.

Exemplo: Ao introduzir os números 3 e 2.1 o programa deverá imprimir o seguinte output:

```
Soma: 3 + 2.1 = 5.1
Subtracao: 3 - 2.1 = 0.9
Multiplicacao: 3 * 2.1 = 6.3
```

6. Área e perímetro do retângulo

Escreva um programa que permita determinar a área e o perímetro de um retângulo. O programa deve pedir ao utilizador a altura e o comprimento do retângulo e devolver a sua área e o seu perímetro.

Exemplo: Ao introduzir a altura 2 e o comprimento 3 o programa deverá imprimir o seguinte output:

```
O perimetro do retangulo e 10 e a sua area e 6.
```

7. Operações em variáveis numéricas

Considere duas variáveis do tipo `int` e duas do tipo `double`. Veja qual o resultado das operações `x/y` e `x%y` permutando as posições das variáveis nos operadores.

8. Média

Escreva um programa que leia 5 medições inteiras de uma distância e devolva a sua média.

Exemplo: Ao introduzir as medições 3, 2, 8, 9 e 5, o programa deverá imprimir o seguinte output:

```
A media das medicoes introduzidas e 5.4.
```

9. Nota de uma disciplina

Escreva um programa que leia as notas de 4 testes de avaliação e do exame final de um aluno numa disciplina e que calcule a classificação final dessa disciplina. Assuma que cada um dos 4 testes vale 10% e o exame os restantes 60%.

Exemplo: Ao introduzir as notas dos testes 17, 16, 15 e 18 e a nota de exame 17, o programa deverá imprimir o seguinte output:

```
A sua nota final na disciplina e de 16.8 valores.
```

10. Operadores de divisão

Escreva um programa que receba dois inteiros a e b e escreva a sua divisão na forma "a=quociente*b+resto". O programa deve também mostra o resultado da divisão decimal.

Exemplo: Ao introduzir os valores $a = 8$ e $b = 5$, o programa deverá imprimir o seguinte output:

```
8 = 1 * 5 + 3    e    8/5 = 1.6
```

11. Converter segundos

Escreva um programa que converta segundos para horas/minutos/segundos.

Exemplo: 4000 segundos = 1h6m40s.

12. Ordem inversa (para corajosos)

Escreva um programa que peça ao utilizador um número inteiro de 3 dígitos e escreva esse número pela ordem inversa. **Sugestão:** Use os operadores de divisão para separar os algarismos das unidades, dezenas e centenas.

Exemplo: Ao introduzir o número 135, o programa deverá imprimir o seguinte output:

```
O numero introduzido foi 135 e o numero pela ordem inversa e 531
```

Guião 2

13. IMC

Escreva um programa que receba como input o peso (em kg) e a altura (em metros) de uma pessoa, valide esses dados (abortando o programa se algum dos dados introduzidos não for válido) e calcule o seu IMC, classificando a pessoa segundo a escala:

IMC	Classificação
≤ 18.5	Abaixo ou muito abaixo do peso
entre 18.5 e 24.9	Peso normal
≥ 25	Acima do peso ou obeso

Nota: IMC=peso/altura^2

Exemplo: Ao introduzir o peso 59 e a altura 1.63, o programa deverá imprimir o seguinte output:

```
Peso normal (IMC=22.2063)
```

Ao introduzir o peso 59 e a altura -1.63, o programa deverá imprimir o seguinte output:

```
A altura introduzida não é válida
```

14. Ano bissexto

Escreva um programa que permita ao utilizador introduzir um ano e verificar se é bissexto. Um ano é bissexto se e só se for divisível por 4 mas não por 100 ou se for divisível por 400.

Exemplo: Os anos 1700, 1800 e 1900 não foram bissextos mas os anos 1600, 2000 e 2012 foram.

15. Equações de segundo grau

Faça um programa que permita resolver equações de grau menor ou igual a dois que seja suficientemente robusto para lidar com qualquer combinação de valores introduzida pelo utilizador. **Sugestão:** Comece por fazer um fluxograma que explique como resolveria uma equação de grau menor ou igual a dois.

16. Data

Escreva um programa que peça uma data ao utilizador na forma dia/mês/ano e verifique se ela é válida.

Exemplo: A data 29/02/2022 não é válida e a data 03/02/1596 é válida.

17. Triângulo retângulo

Escreva um programa que receba 3 números reais e verifique se eles podem ser as medidas dos lados de um triângulo retângulo. **Sugestão:** Utilize o Teorema de Pitágoras.

Exemplo: As medidas 2, 5 e 9 não podem ser as medidas dos lados de um triângulo retângulo mas as medidas 3, 4 e 5 podem.

18. Idade

Faça um programa que peça ao utilizador a sua data de nascimento e a data do dia de hoje e com esses dados calcule a sua idade.

19. Nota de uma disciplina com exclusão da pior nota

Numa determinada disciplina existem 5 momentos de avaliação: 4 exercícios e um exame. Para o cálculo da nota final de um aluno são contabilizados apenas os 3 melhores exercícios e o exame final, sendo que o exame vale 50% e os exercícios os restantes 50%. Além disso, existe uma nota mínima de 7 valores no exame para obtenção de aprovação à disciplina. Escreva um programa que permita determinar a classificação final da disciplina e que valide todas as notas introduzidas.

Exemplo: Ao introduzir as seguintes classificações $e_1 = 15$, $e_2 = 13$, $e_3 = 16$, $e_4 = 18$ e $exame = 18$, o programa deve dar o seguinte resultado:

```
A nota final da disciplina e 17.1667 valores.
```

Substituindo a nota do exame por $exame = 6$, o programa deve dar o seguinte resultado:

```
Nao obteve nota minima no exame por isso reprovou.
```

20. Algoritmo de Euclides

Escreva um programa que permita determinar o máximo divisor comum entre dois inteiros. **Sugestão:** Use o algoritmo de Euclides começando por fazer o seu pseudocódigo.

Exemplo: mdc(132, 80) = 4 e mdc(150, 50) = 50.

21. Ciclo *for*/*while*

Escreva um programa que peça um inteiro n e imprima todos os números pares contidos no intervalo $[1, n[$. Use um ciclo *for* na implementação. Repita depois o exercício usando um ciclo *while*.

Exemplo: Ao introduzir o número inteiro 10, o programa deverá imprimir o seguinte output:

```
Os numeros pares contidos no intervalo [1,10[ sao : 2 4 6 8
```

22. Ciclos e critério de paragem

Escreva um programa que peça sucessivamente ao utilizador números inteiros positivos. O programa deve terminar quando o utilizador introduzir um número inteiro não positivo e nessa altura deve apresentar o mínimo, o máximo e a média dos valores positivos inseridos pelo utilizador.

Exemplo: Ao introduzir a sequência de números 5, 7, 3, 2, 20, 21 e -5, o programa deverá imprimir o output:

```
O valor minimo introduzido foi 2, o valor maximo foi 21 e a media foi 9.66667.
```

23. Soma dos n primeiros números

Escreva um programa que receba um número inteiro positivo n, valide esse número (pedindo sucessivamente o valor de n ao utilizador enquanto não for introduzido um valor válido) e calcule a soma desde 1 até n. O programa deve ainda escrever a soma dos números entre 1 e n múltiplos de 3.

Exemplo: Ao introduzir o valor $n = 10$, o programa deve dar o seguinte resultado:

```
Soma: 1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10 = 55
Soma dos multiplos de tres: 3 + 6 + 9 = 18
```

24. Número de divisores

Escreva um programa que receba um número inteiro positivo n, valide esse número (pedindo sucessivamente o valor de n ao utilizador enquanto não for introduzido um valor válido) e indique o seu número de divisores.

Exemplo: O número de divisores de 12 é 6.

25. Número perfeito

Escreva um programa que receba um inteiro positivo n introduzido pelo utilizador e verifique se esse número é perfeito. **Nota:** Um número é perfeito se for igual à soma dos seus divisores próprios (divisores diferentes do próprio número).

Exemplo: 6 é um número perfeito pois os divisores próprios de 6 são 1, 2 e 3 e 1+2+3=6.

26. Número primo

Escreva um programa que receba um valor introduzido pelo utilizador e que verifique se esse número é primo ou não.

27. Lista de números primos

Escreva um programa que receba um inteiro positivo n introduzido pelo utilizador e imprima todos os números primos entre 1 e n.

Exemplo: Ao introduzir o número 10, o programa deverá imprimir o seguinte output:

```
Os numeros primos de 1 a 10 sao: 2 3 5 7.
```

28. Soma da série

Escreva um programa que peça um inteiro positivo N ao utilizador e calcule a soma da série $\sum_{k=1}^{N} \frac{k}{k+1}$.

Exemplo: $\sum_{k=1}^{5} \frac{k}{k+1} = 3.55$.

29. Ciclos encadeados

Escreva um programa que imprima o seguinte output usando ciclos

```
5 4 3 2 1
4 3 2 1
3 2 1
2 1
1
```

30. Ciclos encadeados 2

Escreva um programa que imprima o output abaixo usando ciclos. De seguida, modifique o programa para imprimir um output semelhante mas desta vez para um qualquer inteiro positivo n que não necessariamente 7.

```
1******
12*****
123****
1234***
12345**
123456*
1234567
```

31. Calendário (para corajosos)

Faça um programa que peça ao utilizador um determinado mês do ano e o dia da semana em que o mês começa e imprima o calendário desse mês.

Exemplo: Ao introduzir o ano 2023, o mês 3 e o dia da semana quarta, o programa deverá imprimir o seguinte output:

```
|S |T |Q |Q |S |S |D |
|  |  |1 |2 |3 |4 |5 |
|6 |7 |8 |9 |10|11|12|
|13|14|15|16|17|18|19|
|20|21|22|23|24|25|26|
|27|28|29|30|31|  |  |
```

32. Jogo *hi-lo* (para os muito corajosos)

Implemente o jogo do *hi-lo*. Primeiro, o programa deve gerar um número aleatório entre 1 e 100. Ao utilizador são dadas 7 tentativas para adivinhar o número gerado pelo computador.

Se o utilizador não adivinhar o número correto, o programa deve dizer ao utilizador se o seu palpite está abaixo ou acima do número correto. Se o utilizador acertou no número correto, o programa deve dizer ao utilizar que ganhou. Se o utilizar não consegue adivinhar o número correto ao fim das 7 tentativas o programa deve dizer-lhe que perdeu e indicar qual o número correto.

Nota: Para gerar um número inteiro aleatório entre 1 e 100 deve usar os seguintes comandos:

`srand (time(NULL));`.

`int numeroAleatorio = rand() % 100 + 1;`.

Exemplo: O programa deve apresentar o seguinte output:

```
Vamos jogar um jogo. Estou a pensar num numero de 1 a 100
e tem 7 tentativas para o adivinhar.

    Tentativa #1: 50
    O numero introduzido e muito baixo.

    Tentativa #2: 75
    O numero introduzido e muito baixo.

    Tentativa #3: 87
    O numero introduzido e muito alto.

    Tentativa #4: 80
    O numero introduzido e muito alto.

    Tentativa #5: 77
    O numero introduzido e muito baixo.

    Tentativa #6: 79
    Correto! Ganhou!
```

Guião 3

33. Construção de vetores

Escreva um programa que:

i) crie o vetor de inteiros v=(1, 2, 3) através de uma lista;

ii) crie um vetor de *strings* s de dimensão 2. Preencha a primeira posição com a string "AA" e a segunda com a string "BB";

iii) crie um vetor de números reais r sem especificar a dimensão. Redimensione o vetor para ter dimensão 3 e insira os valores 1.5, 5 e 0.6 nas posições já criadas;

iv) crie um vetor de carateres c sem especificar a dimensão. Sem usar o método *resize*, adicione ao vetor os valores 'a', '*', '+' e 'v';

v) ao vetor v criado no ponto i) adicione o número 4;

vi) imprima para o ecrã os vetores criados nos pontos anteriores.

34. Construção de vetores 2

Escreva um programa que crie os seguintes vetores de forma automática:

i) v=(1, 2, 3, ..., 25);

ii) u=(2, 4, 6, ..., 50);

iii) z=(0, 1, 0, 1 ..., 1) (vetor com dimensão 30);

35. Norma de um vetor

Escreva um programa que peça um vetor ao utilizador e calcule a sua norma.

Exemplo: Ao introduzir o vetor $(1, 3, 5, 7, 9)$, o programa deverá imprimir o seguinte output:

```
||(1, 3, 5, 7, 9)|| = 12.8452.
```

36. Manipulação de vetores

Escreva um programa que peça sucessivamente valores ao utilizador e os guarde num vetor de inteiros. Após receber o vetor, e sem nunca o ordenar, o programa deve:

i) imprimir o vetor por ordem inversa;

ii) calcular a soma dos valores introduzidos no vetor;

iii) indicar quantos números pares existem no vetor;

iv) verificar se existem valores negativos no vetor;

v) calcular o valor mínimo do vetor;

vi) determinar a posição do primeiro número par existente no vetor. Caso não existam números pares no vetor, deverá ser indicada a posição -1.

vii) determinar a posição do último número par existente no vetor. Caso não existam números pares no vetor, deverá ser indicada a posição -1.

viii) criar um novo vetor que contenha apenas os números ímpares presentes no vetor inicial;

ix) calcular o último índice do valor máximo do vetor;

x) verificar se existem elementos repetidos no vetor.

Exemplo: Ao introduzir o vetor $v = (3, 4, 5, 7, 8, -2, 8, 4)$, temos o seguinte output:

```
i)   O vector introduzido pela ordem inversa e: (4, 8, -2, 8, 7, 5, 4, 3)
ii)  A soma dos valores introduzidos no vetor e 37.
iii) Foram introduzidos 5 valores pares.
iv)  Existem valores negativos.
v)   O valor minimo introduzido e -2.
vi)  A posicao do primeiro numero par existente no vetor e 1.
vii) A posicao do ultimo numero par existente no vetor e 7.
viii)Os numeros impares presentes no vetor inicial sao: (3, 5, 7).
ix)  A ultima posicao onde aparece o maximo do vetor e 6.
x)   Existem numeros repetidos no vetor.
```

37. Produto interno

Escreva um programa que comece por pedir sucessivamente valores reais ao utilizador até que ele introduza um valor não numérico e guarde-os num vetor $v1$. Repita o processo para criar um segundo vetor $v2$. Verifique se os vetores têm a mesma dimensão e em caso afirmativo calcule o produto interno entre eles.

Exemplo: Ao introduzir o vetor (1, 2, 1) e o vetor (-1, 4, 6) o o programa deve escrever:

```
(1, 2, 1) * (-1, 4, 6) = 13
```

38. Vetor do triplo

Escreva um programa que peça um vetor de inteiros ao utilizador. Após receber o vetor, o programa deve criar um novo vetor com a mesma dimensão do inicial e preenche-lo de modo que cada elemento seja o triplo do elemento correspondente no vetor original.

Exemplo: Ao introduzir o vetor (1, 8, -2) o programa deve criar e escrever o vetor (3, 24, -6).

39. Vetor com números primos

Escreva um programa que peça um vetor de inteiros positivos ao utilizador. Após receber o vetor, o programa deve indicar quantos e quais são os números primos existentes no vetor.

Exemplo: Ao introduzir o vetor $v = (6, 9, 3, 5, 1, 8, 7)$, o programa deverá imprimir o seguinte *output*:

```
Existem 3 numeros primos no vetor.
Os numeros primos existentes no vetor sao: (3, 5, 7)
```

40. Vetor normalizado

Escreva um programa que peça um vetor de reais ao utilizador e que depois normalize o vetor, isto é, que altere o vetor de modo a que cada entrada seja dividida pelo máximo dos números inseridos. Escreva o vetor resultante para o ecrã.

Exemplo: Ao introduzir o vetor $(3, 4, 1, 2)$, o programa deve criar e escrever o vetor $(0.75, 1, 0.25, 0.50)$.

41. Vetor de *strings*

Escreva um programa que peça 7 *strings* ao utilizador e as guarde num vetor de *strings*. O programa deverá depois alterar o vetor de *strings* criado de modo a que todas as strings com mais do que 5 carateres sejam substituídas pela letra G e as strings com 5 ou menos carateres sejam substituídas pela letra P. **Sugestão:** o método *.length* indica o número de carateres de uma *strings*.

Exemplo: Ao introduzir o vetor (pao, arroz, batata, farinha, massa, queijo, chocolate) o programa deve criar e escrever o vetor (P, P, G, G, P, G, G).

42. Matrizes

Escreva um programa que leia uma matriz quadrada $n \times n$, sendo o valor de n pedido ao utilizador. O programa deve depois imprimir essa matriz e calcular o seu traço. **Nota:** O traço de uma matriz é a soma dos elementos da sua diagonal.

Exemplo: Ao introduzir a matriz

$$\begin{bmatrix} 1 & 7 & 7 \\ 9 & 2 & 4 \\ 5 & 11 & 3 \end{bmatrix}$$

o programa deverá imprimir o seguinte output:

```
A matriz introduzida foi :
1 7 7
9 2 4
5 11 3
O traco da matriz introduzida e 6.
```

43. Vetores e matrizes

Escreva um programa que leia uma matriz quadrada A de dimensão $n \times n$, e um vetor $\vec{v}$ de dimensão n. O programa deve depois imprimir o resultado da operação matricial $A\vec{v}$.

Exemplo:

$$A \times \vec{v} = \begin{bmatrix} 1 & 3 & 5 \\ 7 & 8 & 9 \\ 13 & 4 & 2 \end{bmatrix} \times \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 9 \\ 24 \\ 19 \end{bmatrix}$$

44. Norma da matriz

Escreva um programa que peça uma matriz ao utilizador e que calcule a sua norma infinito. **Nota:** A norma infinito de uma matriz é o máximo das soma dos valores absolutos dos elementos das linhas da matriz.

Exemplo:

```
A matriz introduzida tem norma 15 e e:
  1  3  7
 -9  2  4
  3 -1  3
```

45. Operações com matrizes

Escreva um programa que permita fazer operações entre matrizes (soma, subtração, multiplicação). O programa deve apresentar o seguinte menu interativo:

– – – – – – – – – – – – – – –

Escolha uma das opões:

1. Soma de matrizes
2. Subtração de matrizes
3. Multiplicação de matrizes
4. Sair

– – – – – – – – – – – – – – –

Opção:

Cada vez que o utilizador selecionar uma opção, terá que inserir duas matrizes (indicando primeiro as suas dimensões). O programa deve verificar se as matrizes têm as dimensões corretas para que a operação matricial escolhida seja feita e, em caso afirmativo, efetuar a operação matricial. A matriz resultante deve ser impressa no ecrã.

46. Histograma

Escreva um programa onde comece por criar um vetor de dimensão 90 com o nome *vetorAleatorio* e preencha-o com números inteiros aleatórios entre 0 e 20 (ver nota abaixo). Tendo por base o vetor criado, crie depois um novo vetor *histograma* que na i-ésima posição contém o número de valores iguais a i do vetor *vetorAleatorio*. Imprima o histograma no ecrã.

Nota: Para gerar um vetor de tamanho 90 com números inteiro aleatórios entre 0 e 20 deve usar os seguintes comandos:

```
srand (0);
vector<int> vetorAleatorio(90);
for(int i = 0; i < 90; i++){
    vetorAleatorio[i] = rand() % 21;
}
```

Exemplo: O programa deve imprimir o seguinte *output*:

```
Nota | Freq. rel. |
0    | 4          | ****
1    | 5          | *****
2    | 8          | ********
3    | 5          | *****
4    | 5          | *****
5    | 1          | *
6    | 6          | ******
7    | 8          | ********
8    | 5          | *****
9    | 0          |
10   | 4          | ****
11   | 3          | ***
12   | 3          | ***
13   | 5          | *****
14   | 5          | *****
15   | 6          | ******
16   | 2          | **
17   | 5          | *****
18   | 2          | **
19   | 2          | **
20   | 6          | ******
```

Guião 4

47. Funções

Implemente as seguintes funções que têm como argumentos 3 valores inteiros:

i) Uma função que devolva o máximo dos valores introduzidos;

ii) Uma função que devolva o mínimo dos valores introduzidos;

iii) Uma função que devolva a média dos valores introduzidos;

iv) Uma função que escreva os valores por ordem crescente;

v) Uma função booleana que verifique se algum dos três números é repetido.

vi) Uma função que devolva a diferença entre o valor máximo e mínimo introduzidos.

Na função *main* peça 3 valores inteiros ao utilizador e chame as funções implementadas.

Exemplo: Ao introduzir os valores 5, 3 e 3, o programa deverá imprimir o seguinte *output*:

```
O maximo dos valores introduzidos e 5.
O minimo dos valores introduzidos e 3.
A media dos valores introduzidos e 3.66667.
3 <= 3 <= 5
Existem numeros repetidos nos valores introduzidos.
A diferenca entre o maximo e o minimo e 2.
```

48. Potência

Crie uma função que receba dois valores positivos, uma base b (real) e um expoente a (inteiro), e que calcule a potência b^a. **Nota:** Não utilize a função *pow* do pacote *cmath*.

Exemplo: $2^{11} = 2048$.

49. Fatorial

Crie uma função que, dado um número inteiro positivo n, calcule o fatorial desse número ($n!$).

Exemplo: $7! = 5040$.

50. Fibonacci

A sucessão de Fibonacci é dada por $F_n = F_{n-1} + F_{n-2}$ para $n > 1$, com $F_0 = F_1 = 1$. Escreva uma função que receba um inteiro positivo n e que determine o n-ésimo elemento da sequência de Fibonacci.

Exemplo: $F_7 = 21$.

51. Passagem por referência *versus* passagem por valor

Implemente as seguintes funções:

```
double funcao1(double x){
    x*=10;
    return x;
}
double funcao2(double & x){
    x*=10;
    return x;
}
double funcao3(const double & x){
    //x*=10; //ERRO!
    double z = x*10;
    return z;
}
```

De seguida, na sua função *main*, escreva o código abaixo.

```
  double y = 1.0;
  cout <<"valor: " << y << " | funcao1: " <<  funcao1(y) <<" | valor: " << y << endl;
  cout <<"valor: " << y << " | funcao2: " <<  funcao2(y) <<" | valor: " << y << endl;
  cout <<"valor: " << y << " | funcao3: " <<  funcao3(y) <<" | valor: " << y << endl;
  cout <<"----------------------------------------------------------------" << endl;
  cout <<"valor: " << y << " | funcao1(1.0): " <<  funcao1(1.0) << endl;
  //cout <<"valor: " << y << " | funcao2(1.0): " <<  funcao2(1.0) << endl;  //ERRO!
```

Execute o programa tal como está e analize o output obtido. De seguida, analize as duas linhas comentadas e explique porque é que elas geram erros no programa.

52. Referências

Crie uma função que receba 3 argumentos. Um desses argumentos deverá ser um ângulo, utilize referências para retornar o valor do seno e do coseno desse ângulo.

Exemplo: Se o ângulo $\alpha = 90$ temos $\sin\alpha = 0.893997$ e $\cos\alpha = -0.44807$.

53. Funções 2

Implemente uma função para cada um dos pontos i) - x) do Exercício 36. Na função *main* faça a leitura do vetor pedido ao utilizador e chame cada uma das funções criadas de forma adequada.

54. Estatística descritiva

Escreva um programa de estatística descritiva que receba N números reais. Crie funções para calcular e retornar a média, a mediana, o desvio-padrão (a dividir por $N-1$) e a moda (caso todos os valores introduzidos tenham sido inteiros). **Nota:** No cálculo da moda deve utilizar uma função auxiliar que verifique se todos os números introduzidos são ou não inteiros.

Exemplo: Ao introduzir a amostra $(1, 3, 7, 8, 9, 5, 4, 7, 6, 6, 7)$, o programa deverá imprimir o seguinte *output*:

```
A media dos valores introduzidos e 5.72727.
A mediana dos valores introduzidos e 6.
O desvio-padrao dos valores introduzidos e 2.3277.
A moda dos valores introduzidos e 7.
```

55. Funções e matrizes

Execute as seguintes instruções:

1. Crie uma matriz quadrada de ordem n, sendo o valor de n pedido ao utilizador.
2. Preencha essa matriz com valores pedidos ao utilizador.
3. Crie uma função, cujo tipo de retorno é *void*, que receba uma matriz e a imprima.
4. Crie uma função booleana que indique se uma matriz quadrada é ou não simétrica.
5. Crie n submatrizes quadradas de ordem k = *1,...,n* de forma a que a submatriz de ordem k seja constituída pelas primeiras k linhas e k colunas da matriz original. Após criar cada matriz, deve usar as funções anteriores para as imprimir e verificar se são simétricas.

Exemplo: Ao introduzir a matriz

$$\begin{bmatrix} 1 & 2 & 1 \\ 2 & 1 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

o programa deverá imprimir o seguinte output:

```
Matriz 1 :
1
A submatriz e simetrica.

Matriz 2 :
1 2
2 1
A submatriz e simetrica.

Matriz 3 :
1 2 1
2 1 3
4 5 6
A submatriz nao e simetrica.
```

56. Recursividade

Repita os Exercícios 48, 49 e 50 implementando funções recursivas.

57. Combinação

A combinação de n a k ($n \geq k$) pode ser calculada de forma recursiva através da relação $C_k^n = C_k^{n-1} + C_{k-1}^{n-1}$. Escreva uma função recursiva que receba como argumento o valor de n e o valor de k e calcule a combinação C_k^n.

Exemplo: $C_2^4 = 6$.

58. Determinante de uma matriz (para os corajosos)

Crie uma função recursiva que calcule o determinante de uma matriz. **Nota:** Utilize a Regra de Laplace.

Guião 5

59. Tratamento de erros - *throw*

Considere o código abaixo. Proteja a função *raizQ* contra a introdução de argumentos negativos lançando uma exceção e terminando o programa sempre que tal acontecer.

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

double raizQ (int n){
    return sqrt(n);
}

int main(){
    int n;
    cout << "Valor de n: ";
    cin >> n;

    double x = raizQ(n);
    cout << "O valor da raiz quadrada de " << n << " e " << x << "\n";

    cout << "\nO programa chegou ao fim \n\n\n";
    return 0;
}
```

60. Tratamento de erros - *try...catch*

Considere o código do exercício anterior. Crie uma nova exceção para o caso do utilizador inserir um valor não numérico aquando da leitura do n na função *main*. Use um bloco *try...catch* na função *main* para lidar com esta exceção e com a do exercício anterior apresentando uma mensagem de erro específica para cada caso.

61. Tratamento de erros - classe `runtime_error`

Proteja o código do exercício 59 contra todos os erros que possam ocorrer utilizando a classe `runtime_error` e o bloco *try...catch*. Deve apresentar mensagens de erro específicas para cada caso.

62. Tratamento de erros - classe `out_of_range`

Crie um vetor de zeros com 3 elementos (posições de 0 até 2) e imprima depois os elementos desse vetor entre as posições 0 e 10. Use primeiro o operador `[]` e em seguida use o método `.at()` para aceder às posições do vetor e veja o que acontece.

63. Tratamento de erros

Considere o seguinte código:

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

double divisao (int numerador, int denominador){
    return (double)(numerador)/denominador;
}

int main(){
    int num, den;
    cout<<"Introduza o valor do numerador e do denominador.\n";
    cin >> num >> den;
    double div = divisao (num, den);
    cout<<"O resultado da divisao de "<< num << " por "<< den << " e "<< div << ".\n";
    return 0;
}
```

Altere o programa de modo a lidar com todo o tipo de erros que possam ocorrer durante a sua execução. Imprima mensagens de erro específicas para cada tipo de erro.

Guião 6

64. Módulos: vetores

Neste exercício vamos criar um módulo que contenha algumas funções úteis para manipular vetores. Crie dois ficheiros extra no seu projeto, um ficheiro cabeçalho (onde se declaram funções) e um ficheiro corpo (onde se definem as funções).

1. Declare e defina uma função `X_norma` neste módulo que permita calcular a norma de um vetor de inteiros. A função deve ser declarada no ficheiro *header* e definida no ficheiro *corpo*.
2. Adicione uma nova função `X_imprime` ao módulo que permita imprimir um vetor. Esta função deve retornar uma string. Exemplo: `"(1,2,3)"`.
3. Adicione uma nova função booleana `X_existe` ao módulo, que verifique se um dado elemento pertence ao vetor.
4. Adicione uma função `X_ordenar` que receba um vector de inteiros e retorne um vetor ordenado de forma ascendente. **Nota:** Não pode utilizar o método `sort`.
5. Adicione uma função `X_produtoInterno` ao seu módulo que receba dois vetores de inteiros e retorne o produto interno entre eles.
6. Adicione uma nova função do tipo *void* ao seu módulo chamada `X_extremizar` que receba um vetor de inteiros e o altere de modo a que cada entrada passe a ser igual ao elemento com maior valor absoluto.

Na função *main* crie um vetor de inteiros e chame depois todas as funções criadas de forma apropriada. Não se esqueça de fazer *include* do ficheiro *header* na função *main*.

Exemplo:

```
X = (2,7,5,3,11,1)
||X|| = 14.4568
u=5: Esta no vetor
u=4: Nao esta no vetor
O vetor X ordenado e (1,2,3,5,7,11)
X*Y = (2,7,5,3,11,1)*(1,-3,4,8,-1,9) = 23
O vetor extremo de X e (11,11,11,11,11,11)
```

65. *Namespaces*

No exercício anterior, defina um *namespace* apropriado ao qual as funções criadas devem pertencer. Modifique a chamada das funções para que a função *main* as identifique de novo.

66. Árvore de âmbitos

Crie um novo ficheiro cabeçalho e dentro dele crie um *namespace* `M_mat` que contenha dois *namespaces* `It` e `Rec`. Dentro do *namespace* `It` declare uma função com o nome `Fatorial` que permita calcular o fatorial de um número de forma iterativa. Dentro do *namespace* `Rec` declare uma função também com o nome `Fatorial` que permita calcular o fatorial de um número de forma recursiva. Defina as funções declaradas de forma apropriada num ficheiro corpo e chame-as na função `main`.

67. Módulo: matrizes

Crie um módulo (com um *namespace* apropriado) que contenha funções para a manipulação de matrizes quadradas de inteiros. Este módulo deverá conter uma função que:

1. Leia as entradas de uma matriz e a crie.
2. Imprima uma matriz.
3. Calcula o traço de uma matriz.
4. Faça a soma de duas matrizes.
5. Faça a subtração de duas matrizes.
6. Faça a multiplicação de duas matrizes.

Na função *main* chame todas as funções criadas de forma apropriada.

Guião 7

68. Classes: Complexo

Crie uma classe `Complexo` que represente um número complexo. A classe deve ter:

i) Atributos privados que façam sentido e métodos públicos para alterar e visualizar esses atributos;

ii) Um construtor por omissão;

iii) Um construtor que permita definir a parte real e a parte imaginária de um número complexo;

iv) Um método *Imprime* para imprimir o complexo na forma a+bi;

v) Um método *Modulo* para retornar o módulo do complexo;

vi) Um método *Simetrico* para retornar o simétrico do complexo;

vii) Um método *Conjugado* para retornar o conjugado do complexo;

viii) Um método booleano *IPuro* que verifique se o complexo é um imaginário puro.

No ficheiro cabeçalho que criou, declare fora da classe funções que devolvam a soma, a subtração e a multiplicação de dois complexos. Todas estas funções devem ser definidas no ficheiro `corpo` já existente. Na função *main* teste tudo o que implementou através das seguintes instruções:

i) Crie um complexo z_1 para representar o complexo 3+2i usando o segundo construtor que definiu e imprima-o no ecrã usando o método *Imprime*;

ii) Altere a parte real do complexo z_1 para 7;

iii) Crie um complexo z_2 e preencha-o com valores pedidos ao utilizador;

iv) Escreva os complexos z_1 e z_2 no ecrã usando o método *Imprime*;

v) Use o construtor por omissão para criar um novo complexo z_3 onde deverá guardar o simétrico do complexo z_2;

vi) Crie um novo complexo z_4 que seja o conjugado do complexo z_3;

vii) Crie um novo complexo z_5 que corresponda à subtração de z_3 com z_4 e imprima-o no ecrã;

viii) Use o método *IPuro* para verificar se z_5 é um imaginário puro.

69. Classes: Fração

Crie uma classe `Fracao` que represente uma fração. Esta classe deve estar preparada para lidar com todo o tipo de erros que possam ocorrer lançando exceções com mensagens de erro específicas sempre que necessário. A classe deve:

1. armazenar o numerador e o denominador da fração como atributos privados (utilize long long int com typedef para simplificar);
2. ter um construtor por omissão e um construtor pertinente;
3. ter um método público que devolva uma fração escrita como string;
4. ter um método privado que devolva o máximo divisor comum entre dois números;
5. ter um método público que permita simplificar a fração.

Fora da classe, crie uma função que receba duas frações e retorne o seu produto e outra que retorne a sua soma. Na função `main` crie duas frações e calcule a sua soma e o seu produto apresentando o resultado simplificado.

70. Classes: Livro

Crie uma classe que represente um livro. A classe deverá ter um construtor por omissão e um construtor com argumentos pertinentes. Deverá ter os seguintes membros privados: título, autor e ano de publicação. Deve criar membros públicos que permitam a alteração dos membros privados e outros que os retornem. Deve ainda ter um método público que imprima a informação do livro na forma

```
Titulo: ...
Autor: ...
Ano: ...
```

Na função `main` peça ao utilizador o título do livro, o autor e o ano, crie um livro e imprima a informação sobre o livro introduzida pelo utilizador.

71. Classes: Polinómio

Crie uma classe que represente um polinómio com coeficientes inteiros. Esta classe deve estar preparada para lidar com todo o tipo de erros que possam ocorrer lançando exceções sempre que necessário. Neste exercício é fornecida a função *main* que deverá ser usada para testar a classe e o output esperado associado. A única alteração que pode fazer na função *main* é a introdução de um bloco *try..catch* para lidar com as exceções lançadas. A classe deve:

1. armazenar o grau e os coeficientes do polinómio;
2. ter um construtor por omissão vazio, outro que aceite o grau do polinómio e inicie todos os seus coeficientes a zero e outro que aceite o grau e um vetor de coeficientes;
3. ter métodos para definir/alterar um coeficiente do polinómio;
4. ter um método que permita imprimir um polinómio;

5. ter um método para calcular o valor do polinómio num dado x;
6. ter um método que permita alterar o grau do polinómio. Se o novo grau for inferior ao anterior, o polinómio deve ser truncado, caso contrário, os novos coeficientes devem ser pedidos ao utilizador.

Fora da classe, deve criar ainda uma função que permita somar dois polinómios.

O programa deve funcionar com a seguinte função `main`:

```
int main()
{

    Polinomio p1(4);
    for(int i=0; i<=4; i++)
        p1.AlteraCoef(i,1);
    cout << "p1 = "<< p1.Imprime();

    Polinomio p2(5, {1,-2,0,4,-1, 3});
    cout << "\np2 = "<<p2.Imprime();
    cout << "\nO valor de p1 em x = 2 e " << p1.Valor(2) <<endl;

    p1.AlteraGrau(2);
    cout << "Diminuir grau de p1 de 4 para 2\n  p1 = " << p1.Imprime();

    cout << "\nAumentar grau de p1 de 2 para 5\n";
    p1.AlteraGrau(5);
    cout << "\n p1 = " << p1.Imprime() << "\n\n";

    Polinomio p3(2, {1,1,1});
    Polinomio p4(4, {2,2,2,2,2});
    Polinomio p5 = Soma(p4, p3);
    cout << "Soma: \n( "<< p3.Imprime() <<" ) + ( " << p4.Imprime() << " ) ";
    cout << "=  " << p5.Imprime() << "\n\n";
    return 0;
}
```

O output esperado é:

```
p1 = 1 + x + x^2 + x^3 + x^4
p2 = 1 - 2x + 4x^3 - x^4 + 3x^5
O valor de p1 em x = 2 e 31
Diminuir grau de p1 de 4 para 2
  p1 = 1 + x + x^2
Aumentar grau de p1 de 2 para 5
v[3] = 6
v[4] = 5
v[5] = 8

 p1 = 1 + x + x^2 + 6x^3 + 5x^4 + 8x^5

Soma:
( 1 + x + x^2 ) + ( 2 + 2x + 2x^2 + 2x^3 + 2x^4 ) =  3 + 3x + 3x^2 + 2x^3 + 2x^4
```

72. Classes: Retângulo

Crie uma classe que represente um retângulo. Esta classe deve estar preparada para lidar com todo o tipo de erros que possam ocorrer. A classe deve ter:

1. quatro membros privados: altura, comprimento e duas variáveis que representem as duas coordenadas do canto superior esquerdo do retângulo;
2. um construtor por omissão que atribui as dimensões de um quadrado 1×1 e um construtor que aceite quatro argumentos para preencher os membros privados;
3. funções para modificar as dimensões do retângulo;
4. uma função que retorne a área e outra que retornar o perímetro do retângulo;
5. uma função para representar o retângulo no ecrã. Assuma que o ecrã é uma tela de 40×20 caracteres e considere que uma unidade na horizontal corresponde a dois caracteres e na vertical a um *new line*.

Implemente a função `main` de forma a dar o output indicado em baixo. As instruções começadas por `>>` são o input do utilizador e não necessitam de ser apresentadas da mesma forma que está no output.

```
O retangulo do construtor por omissao e:
 __
|__|
>> Altura e o comprimento do retangulo: altura = 2, comprimento = 4
>> Coordenadas (x,y) onde deve ficar o canto superior esquerdo do retangulo: (2, 2)
O perimetro do retangulo introduzido e 12 e a area e 8.
O retangulo introduzido e:


    ________
   |        |
   |________|
```

73. Classes: Pontos em $\mathbb{R}^2$

Crie a classe `PontoR2` que represente um ponto $(x, y) \in \mathbb{R}^2$. Crie um construtor por omissão, que inicializa o ponto como sendo a origem, e outro que aceite as coordenadas do ponto. A classe deve ter funções-membro públicas que permitam o acesso aos seus membros privados. Adicionalmente, a classe deverá ter funções-membro que permitam:

1. determinar a distância euclideana do ponto à origem (ponto $(0, 0)$);
2. determinar a distância euclideana do ponto a um outro ponto;
3. rodar o ponto α graus relativamente à origem. **Nota:** O ponto (x, y) rodado α graus dá origem ao ponto $(x \times \cos\alpha - y \times \sin\alpha, y \times \cos\alpha + x \times \sin\alpha)$.
4. verificar se o ponto é colinear com outro ponto. **Nota:** O ponto (x_1, y_1) é colinear com o ponto (x_2, y_2) se e só se $x_1 \times y_2 = x_2 \times y_1$.

Implemente a função `main` de forma a dar o output indicado em baixo. A instrução começada por `>>` é o input do utilizador e não necessita de ser apresentada da mesma forma que está no output.

```
A distancia euclideana do ponto (1.000000 , 2.000000)a origem e 2.23607
>> Introduza as coordenadas para um ponto: (2 , 5)
A distancia do ponto (1.000000 , 2.000000) ao ponto (2.000000 , 5.000000) e 3.16228
O ponto (1.000000 , 2.000000) rodado 90 graus e (-2.000000 , 1.000000)
Os pontos (-2.000000 , 1.000000) e (2.000000 , 5.000000) nao sao colineares
```

74. Classes: Campeonato de LP

Implemente as classes indicadas abaixo. Além dos métodos pedidos explicitamente, deve implementar também um método de impressão em cada classe. Se for necessário, pode também implementar métodos além dos indicados nos vários pontos.

1. Crie a classe `Jogador` que tem como membros privados o nome do jogador, o número do jogador no Sindicato dos Jogadores, que é um identificador único, a forma física do jogador (valor inteiro entre 1 e 100) e o talento do jogador (valor inteiro entre 1 e 100). Crie uma função que permita calcular a aptidão do jogador de acordo com a seguinte fórmula: $0.60 \times$forma física $+ 0.40 \times$talento. Na implementação do método de impressão da classe, pode usar as instruções `left` e `setw(·)` disponíveis no pacote `iomanip`.

2. Crie a classe `Equipa` que tem como membros privados o nome da equipa, o nome do desporto e um vetor de objetos do tipo `Jogador`. Além das funções de acesso aos membros privados da classe, crie funções para adicionar e remover um jogador da equipa recebido como argumento.

3. Crie a classe `Jogo` que tem como membros privados as duas equipas que o vão disputar. Adicione uma função que calcule o vencedor do jogo sabendo que ganha a equipa cuja aptidão média dos seus jogadores seja a maior. Em caso de igualdade das aptidões médias, gere um número binário aleatório $r \in \{0, 1\}$ usando a instrução `r=rand()%2`. Se $r = 0$, a equipa vencedora é a primeia, caso contrário será a segunda.

4. Crie uma classe `Liga` que tem como membros privados o nome da liga, um vetor de objetos `Equipa` e outro de `Jogo`. Crie uma função que permita determinar o vencedor da liga, isto é, a equipa que ganhou mais jogos sabendo que cada equipa joga uma vez com todas as outras.

As classes implementadas devem funcionar com a seguinte função *main*.

```
int main(){
    Jogador j1("Joao", 1, 56, 90); Jogador j2("Manuel", 2, 78, 20);
    Jogador j3("Maria", 3, 82, 63); Jogador j4("Joana", 4, 23, 20);
    Jogador j5("Rita", 5, 98, 68); Jogador j6("Francisca", 6, 25, 99);
    Jogador j7("Caetana", 7, 69, 64); Jogador j8("Cristovao", 8, 48, 87);
    Equipa e1 = Equipa("Equipa_1", "-");
    e1.AdicionarJogador(j8); e1.AdicionarJogador(j7); e1.AdicionarJogador(j2);
    e1.RemoverJogador(j8);
    Equipa e2 = Equipa("Equipa_2", "-");
    e2.AdicionarJogador(j8); e2.AdicionarJogador(j4);
    Equipa e3 = Equipa("Equipa_3", "-");
    e3.AdicionarJogador(j5); e3.AdicionarJogador(j6);
    Equipa e4 = Equipa("Equipa_4", "-");
    e4.AdicionarJogador(j1); e4.AdicionarJogador(j3);
```

```
    Liga liga = Liga("LigaLP");
    liga.AdicionaEquipa(e1); liga.AdicionaEquipa(e2);
    liga.AdicionaEquipa(e3); liga.AdicionaEquipa(e4);

    auto equipas = liga.VeEquipas();
    for(size_t i = 0; i < equipas.size() - 1; ++i){
        for(size_t j = i+1; j < equipas.size(); ++j){
            if(equipas[i].VeNome() == equipas[j].VeNome())
                continue;
            Jogo jogo = Jogo(equipas[i], equipas[j]);
            liga.AdicionaJogo(jogo);
        }
    }
    liga.Imprime();
    auto vencedorLiga = liga.CalculaEquipaVencedora();
    cout << "A vencedora da liga " << liga.VeNome() << " e " << vencedorLiga.VeNome() << endl;
    return 0;
}
```

o output esperado do programa é o seguinte:

```
##########################################
####               Equipas            ####
##########################################
Nome: Equipa_1
  Caetana    ID:7   F:69   T:64    Apt: 67
  Manuel     ID:2   F:78   T:20    Apt: 54.8

Nome: Equipa_2
  Cristovao  ID:8   F:48   T:87    Apt: 63.6
  Joana      ID:4   F:23   T:20    Apt: 21.8

Nome: Equipa_3
  Rita       ID:5   F:98   T:68    Apt: 86
  Francisca  ID:6   F:25   T:99    Apt: 54.6

Nome: Equipa_4
  Joao       ID:1   F:56   T:90    Apt: 69.6
  Maria      ID:3   F:82   T:63    Apt: 74.4

############################
##         Jogos          ##
############################
  Equipa_1 vs Equipa_2 -> Equipa_1 ganha
  Equipa_1 vs Equipa_3 -> Equipa_3 ganha
  Equipa_1 vs Equipa_4 -> Equipa_4 ganha
  Equipa_2 vs Equipa_3 -> Equipa_3 ganha
  Equipa_2 vs Equipa_4 -> Equipa_4 ganha
  Equipa_3 vs Equipa_4 -> Equipa_4 ganha

A vencedora da liga LigaLP e Equipa_4
```

75. Classes: O Jogo do Galo

Pretende-se implementar a classe `JogoGalo` para jogar ao Jogo do Galo. Esta classe deve ter dois atributos: uma matriz de `char`, que representa o quadro de jogo, e um `char`, que indica qual o símbolo do próximo jogador a jogar (X ou O).

1. Crie um construtor que receba o símbolo do primeiro jogador a jogador e que inicialize o quadro de jogo com o `char` -.
2. Crie um método `VerJogador` que permita aceder ao simbolo do jogador.
3. Crie um método *void* `MostrarTabuleiro` que imprima o tabuleiro de jogo para o ecrã no formato abaixo. **Nota:** No exemplo está representado o tabuleiro vazio.

```
 - | - | -
-----------
 - | - | -
-----------
 - | - | -
```

4. Crie um método booleano `FazerMovimento` que receba a linha e coluna onde deve ser colocado o símbolo e que altere o símbolo do jogador.
5. Crie um método `VerificarVencedor` que devolve o símbolo do jogador vencedor, o símbolo E em caso de empate e o símbolo - caso o jogo ainda não tenha acabado.

Preencha a função *main* apresentada de seguida com os nomes dos métodos apropriados substituindo os `(...)` para poder jogar ao Jogo do Galo. Sempre que utilizador insira informação que não seja válida, peça de novo a informação até que seja introduzido um valor válido.

```
int main(){
    char symbol;
    cout << "Insira o primeiro jogador a jogar: ";
    cin >> symbol;
    JogoGalo jogo = (...);
    int l, c;
    while (true) {
        cout << "Tabuleiro atual:" << endl;
        jogo.(...);
        cout << "Vez do Jogador " <<  jogo.(...) << ". Insira linha e coluna (0-2): ";
        cin >> l >> c;
        jogo.(...);
        char res = jogo.(...);
        if (res == 'X' || res == 'O')
            cout << "Jogador " << res << " ganha!" << endl; break;
        else if (res == 'E') {
            cout << "E um empate!" << endl; break;
    }
    cout << "Tabuleiro final:" << endl;
    jogo.(...);
    return 0;
}
```

Guião 8

76. Sobrecarga de operadores: Complexo

Neste exercício, pretende-se redefinir vários operadores para manipular números complexos. Nos mesmos ficheiros criados na implementação da classe `Complexo` defina os seguintes operadores:

i) Operadores binários para a soma, a subtração e a multiplicação de complexos;

ii) Operador unário para o simétrico de um complexo;

iii) Operador de `ostream` e de `istream` para ler e escrever um complexo, respetivamente. Use o formato "a+bi" para representar um número complexo.

Teste os operadores que implementou com a seguinte função *main*:

```
int main(){
    Complexo z1(2,3);
    Complexo z2;
    cout << "Introduza um numero complexo: \n";
    cin >> z2;

    cout << "O complexo introduzido foi: " << z2;

    Complexo zSimetrico = -z2;
    cout << "O simetrico do complexo introduzido e: " << zSimetrico << endl;

    Complexo zSoma = z1+z2;
    Complexo zSubtracao = z1-z2;
    Complexo zMultiplicacao= z1*z2;
    cout << "Soma: " << zSoma << endl;
    cout << "Subtracao: " << zSubtracao << endl;
    cout << "Multiplicacao: " << zMultiplicacao << endl;
    return 0;
}
```

77. Sobrecarga de operadores: Polinómio

Neste exercício pretende-se redefinir vários operadores para manipular polinómios. Nos mesmos ficheiros criados na implementação da classe `Polinomio` defina os seguintes operadores:

i) Operadores binários para a soma e a subtração de polinómios;

ii) Operadores unários [ ] para aceder e modificar os coeficientes do polinómio;

iii) Operador unário () que receba um escalar e calcule o valor do polinómio nesse escalar;

iv) Operador de `ostream` para escrever um polinómio. Use o formato "c_0 + c_1x + ... + c_nx^n" para representar um polinómio.

78. Sobrecarga de operadores: Frações

Neste exercício, pretende-se redefinir vários operadores para manipular frações. Nos mesmos ficheiros criados na implementação da classe `Fracao` defina os seguintes operadores:

i) Operadores binários para a soma, a subtração, a multiplicação e a divisão de frações;

ii) Operador unário para o simétrico de uma fração;

iii) Operadores de `ostream` e de `istream` para ler e escrever uma fração, respetivamente. Use o formato "numerador/denominador" para representar uma fração.

79. Classes: Data (com sobrecarga de operadores)

Neste exercício pretende-se criar uma classe para manipular datas. A classe, chamada `Data`, deve ter como atributos privados o dia, o mês e o ano. A classe deve ainda ter:

i) Um construtor por omissão e outro definido pelo utilizador que valide a data recebida (deve continuar a pedir datas ao utilizador até ser introduzida uma data válida);

ii) Métodos públicos que permitam visualizar o dia, mês e o ano;

iii) Operadores unários `++`, que permite incrementar um dia na data, e `--`, que permite decrementar um dia na data;

iv) Operadores de `ostream` e de `istream` para ler e escrever uma data, respetivamente. Use o formato "dia/mês/ano" para representar uma data.

80. Sobrecarga de operadores: Pontos em $\mathbb{R}^2$

Neste exercício, pretende-se redefinir vários operadores para manipular pontos em $\mathbb{R}^2$. Nos mesmos ficheiros criados na implementação da classe `PontoR2` defina os seguintes operadores:

i) Operadores binários para a soma e a subtração de pontos em $\mathbb{R}^2$;

ii) Operador binário * que devolve o produto interno entre dois pontos de $\mathbb{R}^2$;

iii) Operador unário para o simétrico de um ponto em $\mathbb{R}^2$;

iv) Operadores de `ostream` e de `istream` para ler e escrever um ponto, respetivamente. Use o formato "(x, y)" para representar um ponto em $\mathbb{R}^2$.

81. Sobrecarga de operadores: Matriz

Defina uma classe `Matriz` que tenha como atributos privados o número de linhas, o número de colunas e a matriz (utilize um `vector<vector<int>>`).

i) Defina um construtor por omissão e outro que receba o número de linhas e colunas da matriz bem como os elementos da matriz na forma de um vetor de vetores;

ii) Crie o operador `[ ]` para aceder aos elementos da matriz;

iii) Crie também o operador `ostream` para imprimir a matriz;

iv) Crie os operadores de soma, de subtração e de multiplicação matricial. Os operadores devem reconhecer as dimensões das matrizes e portanto verificar se as operações são possíveis.

Guião 9

82. Herança e polimorfismo: Edifício

Execute os seguintes passos:

i. Crie uma classe base chamada `Edificio` cujos atributos são a morada, o tamanho em metros quadrados e o número de andares. A classe deve ter um único construtor que receba argumentos pertinentes e um método que permita imprimir informação sobre o edifício.

ii. Crie duas classes derivadas `Casa` e `Escritorio` que herdem publicamente da classe `Edificio`. A classe `Casa` deve ter como atributos o número de quartos e de casas de banho. Já a classe `Escritorio` deve ter como atributo o número de cubículos.

iii. Crie um método em cada uma das classes derivadas que permita imprimir não só a informação específica sobre elas como também a informação geral que caracteriza um qualquer edifício. Evite repetições de código no(s) ficheiro(s) *header*.

iv. Crie um método puramente virtual na classe `Edificio` chamado `Avaliacao` que indique quanto vale um edifício. Implemente este método nas classes derivadas com as funções de avaliação dadas na tabela seguinte.

Tipo	Função avaliação
Casa	$20\times$N.º quartos $+$ $5\times$N.º casas de banho $+$ $10\times$N.º andares
Escritório	$10\times$N.º cubículos $+$ $10\times$N.º andares

v. Crie uma outra classe `Rasteira` que herde publicamente da classe `Casa` e que tenha como intuito representar casas construídas no rés-do-chão (sem andares). Esta classe deve ter um único construtor que receba no máximo quatro argumentos.

vi. Na função `main`, crie uma `Casa`, um `Escritorio` e uma casa `Rasteira`. Imprime toda a informação relativa a cada uma delas, incluindo a sua avaliação.

83. Herança e polimorfismo: Pessoa

Crie quatro classes que representem pessoas que frequentam o ISEG. A classe base, `Pessoa`, deverá ter como atributos um nome e a data de nascimento (representada por 3 inteiros). Além disso, deve ter um método público que receba uma data (três inteiros) e que calcule a idade da pessoa nessa data. Implemente também um método público puramente virtual chamado `Profissao()`. Este método deve ser concretizado nas classes devolvendo a profissão da pessoa.

As classes derivadas devem ser: `Estudante`, `Professor` e `Funcionario`.

- A classe `Estudante` deve ter o nome do curso que frequenta e a média como atributos privados.

- A classe `Professor` deve ter os seguintes atributos privados: categoria e anos de trabalho.
- A classe `Funcionario` deve ter os seguintes atributos privados: nome do departamento e anos de trabalho.

Cada classe deverá ter construtores pertinentes e métodos para retornar os atributos privados.

Crie uma função que tenha como argumento uma `Pessoa`. Esta função deve chamar o método `Profissao()` para demonstrar a capacidade de polimorfismo.

84. Herança e polimorfismo: Veículo

Execute os seguintes passos:

i. Crie uma classe mãe `Veiculo` para guardar informação comum a todos os tipos de veículos (marca, ano de produção e matrícula) e um construtor pertinente para inicializar objetos desta classe. Crie um método público que permita imprimir informação geral sobre um veículo de acordo com o exemplo abaixo:

```
Informacao Geral:
Marca: Opel
Ano: 1996
Matricula: 12-29-JE
```

ii. Crie uma classe `Pesado` que seja derivada de `Veiculo`, e que herde *publicamente* dela. Crie atributos específicos para o tipo `Pesado`, nomeadamente a carga máxima e a indicação se é ou não refrigerado. Crie um construtor pertinente para a classe `Pesado`.

iii. Crie uma outra classe mãe que represente o seguro de um certo veículo, chame-lhe `Seguro`. Esta classe deve conter os atributos validade e valor do seguro, sendo a validade um objeto da classe `Data` criada no Guião 8 e um construtor pertinente. Crie um método público que permita imprimir informação do seguro de um veículo segundo o exemplo abaixo:

```
Informacao de Seguro:
Validade: 20/2/2029
Valor: 67
```

iv. Altere a classe `Pesado` para herdar também da classe `Seguro`.

v. Crie uma classe `Motorizado` que seja derivada de `Veiculo` e de `Seguro`, e que herde *publicamente* delas. Essa classe deve ter como único atributo o número de rodas. Crie um construtor pertinente para esta classe.

vi. Crie um método público puramente virtual chamado `TipoVeiculo` na classe `Veiculo`. Este método deve se concretizar nas classes derivadas escrevendo para o ecrã o tipo de veículo em causa.

vii. Crie uma função global `Tipo` no ficheiro *main.cpp* que receba um objeto do tipo `Veiculo` e imprima a informação geral sobre esse veículo bem como o tipo de veículo em causa.

viii. Na classe `Veiculo` implemente um método privado que faça a validação de uma matrícula (use o formato 00-00-AA). Alguns comandos úteis: `.size()` - dimensão de uma string; `.at(i)` - carater na posição i; `isdigit(c)` - verifica se o carater c é um digito entre 0 e 9; `isupper(c)` - verifica se o carater c é uma letra maiúscula entre A e Z. Altere o construtor da classe `Veiculo` para verificar se a matrícula recebida é válida e, em caso negativo, pedir sucessivamente ao utilizador uma nova matrícula até ser introduzida uma que seja válida.

Guião 10

85. Leitura de ficheiros - `ifstream`

Crie manualmente um ficheiro *Notas.txt* que contenha as notas finais de uma disciplina. Leia essas notas para o seu programa guardando-as num vetor. Calcule depois a sua média e imprima-a no ecrã.

Dados do ficheiro: 12 15 9 18 15 10 9 8 19

86. Escrita de ficheiros - `ofstream`

Crie um programa que escreva informação sobre uma turma num ficheiro com o nome "Turma.txt". Comece por pedir ao utilizador o número de alunos da turma. Peça depois ao utilizador o nome e as três notas de cada aluno e escreva-as no ficheiro de acordo com o formato abaixo indicado.

Exemplo do formato do ficheiro:

```
Joana/12/14/15
João/13/20/7
...
```

87. Escrita de ficheiros - `ofstream, ifstream, ios_base::out/app, getline`

Crie um programa que peça uma frase ao utilizador e a escreva num ficheiro de texto. Caso o ficheiro já exista, o programa deve perguntar ao utilizador se quer apagar o conteúdo do ficheiro existente ou adicionar informação a esse ficheiro.

88. Alternativa ao `to_string`: `ostringstream, .str()`

Crie um programa que peça ao utilizador dois números reais que representem o numerador (n) e o denominador (d) de uma fração. Crie depois uma `string` para guardar a expressão `"n/d=r"` usando a função `to_string` e imprima essa `string`. Crie depois uma nova `string` com o mesmo conteúdo usando uma `ostringstream` e imprima-a no ecrã. Compare os resultados.

89. Contagem de palavras: `istringstream`

Escreva um programa que leia uma `string` introduzida pelo utilizador. Determine o número de palavras da `string` introduzida utilizando um objeto do tipo `istringstream`.

90. Numerar palavras: `ostringstream`, `istringstream`

Escreva um programa que leia uma frase da consola. Crie uma `string` com a mesma frase em que no final de cada palavra aparece a ordem numérica dessa palavra.

Exemplo:

```
Introduza uma frase:
Esta e uma frase que tem imensos espacos em branco

Esta[1] e[2] uma[3] frase[4] que[5] tem[6] imensos[7] espacos[8] em[9] branco[10]
```

91. Equipas Futsal

Crie manualmente o seguinte ficheiro referente a idades de jogadores de três equipas de futsal:

```
AFT  23  18  19  22  29  30  25  24
WET  20  31  19  27  35  30  27  20
FCR  17  32  22  22  35  37  21  24
```

Escreva um programa que leia a informação deste ficheiro e que determine qual a idade do jogador mais velho e mais novo de cada equipa.

92. Notas do ficheiro Turma

Crie um programa que leia a informação de uma turma (utilize o ficheiro "Turma.txt" criado no exercício 85). Deve imprimir no ecrã o nome e a média de cada aluno com duas casas decimais (use o manipulador `setprecision` disponível na biblioteca `iomanip`).

Exemplo:

```
Joana: 13.67
Joao: 13.33
```

93. Frequência de cada palavra

Crie um programa que peça uma frase ao utilizador e que indique o número de vezes que cada palavra aparece na frase.

94. Apagar linha

Escreva um programa que permita apagar uma linha de um ficheiro. O programa deve pedir ao utilizador qual o número da linha que pretende apagar.

95. Calculo do IMC

Crie manualmente o ficheiro abaixo cujo formato é "`Numero. Nome_Idade Peso Altura`". Escreva depois um programa que leia esse ficheiro e: (i) indique quais as pessoas que têm menos de 23 anos; (ii) escreva um ficheiro com o nome da pessoa e o seu índice de massa corporal (na forma `Numero. Nome IMC`).

```
1. Maria_25 56 1.75
2. Joao_22 60 1.6
3. Lara_30 52 1.57
4. Pedro_21 80 1.85
...
```

96. Brincar com ficheiros

Crie um programa que leia uma lista de valores inteiros de um ficheiro chamado *input.txt* e que:

1. Imprima o valor mais alto para o ecrã.
2. Guarde os valores ao quadrado num novo ficheiro chamado *output1.txt*.
3. Remova os valores em duplicado e que escreva a lista sem esses valores num novo ficheiro chamado *output2.txt*.
4. Escreva os números por ordem decrescente num novo ficheiro chamado *output3.txt*.
5. Imprima para o ecrã a maior sequência crescente de valores presente no ficheiro.

97. Nova escala de notas

Crie um programa que leia uma lista de notas (10-20) de um ficheiro chamado *Notas.txt*. Cada linha do ficheiro contêm o nome e a nota de um aluno (separadas por um espaço).

1. Atribua uma classificação a cada estudante baseada na seguinte escala: Suficiente (10-12), Bom (13-15), Muito Bom (16-18), Excelente (19-20). O programa deve guardar as notas na nova escala num novo ficheiro chamado *NotasNovaEscala.txt* no formato *"Nome Classificação"*.
2. Escreva as 10 notas mais altas num novo ficheiro chamado *TopNotas.txt*
3. Escreva o histograma das notas num novo ficheiro chamado *Histograma.txt*. O histograma deve mostrar o número de estudantes que receberam cada classificação.

98. Frase com mais palavras

Escreve um programa que leia uma lista de frases de um ficheiro chamado *frases.txt* e que imprima para o ecrã a frase com mais palavras.

99. Ordem inversa

Escreva um programa que leia uma lista de palavras de num ficheiro chamado *palavras.txt* e que as escreva pela ordem inversa num ficheiro chamado *PalavrasInverso.txt*

100. Ordenação de datas

Escreva um programa que leia uma lista de datas no formato `DD/MM/AAAA` de um ficheiro chamado *Datas.txt* e que as escreva por ordem cronológica num ficheiro chamado *DatasCronologicas.txt*.

101. Interpretação de expressões matemáticas

Escreva um programa que peça sucessivamente expressões matemáticas ao utilizador e devolva o seu resultado. As expressões matemáticas devem envolver apenas as operações básicas (`-`, `+` e `*`), não podem conter parêntesis e cada operador deve ter um espaço antes e depois. O programa termina quando o utilizador introduzir a palavra "sair".

Exemplo:

```
-> 2 + 4 * 2 - 7
R: 3

-> -5 + 10 * 2
R: -15

-> 2 + 3 + 2 * 2 - 5 - 2 * 6 - 3 - 2 * 5
R: -21

-> sair
```