Exercícios de
Física Computacional

Mestrado em Engenharia Física-Tecnológica (MEFT)

Fernando Barao
Departamento de Física do Instituto Superior Técnico
Ano Lectivo: 2014-15

Exercícios de Física Computacional

Programaçao em C/C++

Exercício 1 : O programa **addnumbers.C** é suposto adicionar todos os números inteiros entre dois números introduzidos pelo utilizador no programa.

a) Compile o programar e verifique se existe algum erro.

b) Crie o executável e corra o programa para os pares de valores:
$(\mathbf{5}, \mathbf{-2})$, $(\mathbf{5}, \mathbf{20})$ e $(\mathbf{10}, \mathbf{55})$.

Exercício 2 : Considere a fórmula matemática,

$$\boldsymbol{z(x) = x + f(x)}$$

com,

$$\boldsymbol{f(x) = sin^2(x)}$$

Realize um programa em C/C++ que calcule $\boldsymbol{z(x)}$ para os valores de $\boldsymbol{x = 0.4, 2.1, 1.5}$ e imprima os resultados no monitor do computador.
Nota: a função $\boldsymbol{f(x)}$ *deve ser construída autonomamente*

Exercício 3 : O espaço em memória ocupado pelas variáveis depende do seu *tipo*. Realize um programa em C/C++ que calcule o número de bytes ocupado em memória pelas variáveis dos seguintes tipos: **short int**, **int**, **long int**, **unsigned int**, **float**, **double**, **long double**

Exercício 4 : Realize um programa em C/C++ que determine o valor da constante $\boldsymbol{\pi}$ com precisão *float* e *double* a partir da função **atan()**. Compare o valor obtido com o valor exacto de $\boldsymbol{\pi}$. Determine a precisão obtida em *float* e *double*.

Exercício 5 : Aspectos relacionados com a implementação do C++ em cada arquitectura podem ser encontrados na *C++ standard library*
<limits> .

function	provides
numeric_limits<type>::max()	largest type value
numeric_limits<type>::min()	smallest type value

Realize um program em C/C++ que avalie e imprima no monitor do computador os limites máximo e mínimo dos seguintes tipos de variáveis: **int**, **unsigned int**, **float**, **double**.

Exercício 6 : Calcule o quadrado de um número inteiro positivo, $\boldsymbol{x^2}$, usando somente as operações:

a) adição, subtracção, multiplicação ($\times \mathbf{2}$)

b) junte a hipótese de chamar uma função de forma recorrente (*recursion*)

Exercício 7 : Tendo como base o programa **addnumbers.C**, construa um outro programa *addnumbersS.C* que adiciona os quadrados dos números inteiros compreendidos entre os limites inseridos no programa pelo utilizador. O resultado da soma deve ser calculado em tipo **int** e em **double**. Compare os resultados para o caso $(\mathbf{1}, \mathbf{5000})$.

Exercício 8 : Realize um programa em C/C++ composto das funções **main()** e **int fact(int)** que determine o factorial do número $\boldsymbol{n}$ introduzido pelo utilizador no programa, $\boldsymbol{n!}$. Obtenha primeiramente o código objecto **.o** e só depois o código executável **.exe**.

Exercício 9 : Realize um programa em C/C++ que calcule:

$$\sum_{i=0}^{100} \sum_{j=5}^{300} \cos\left(i^2 + \sqrt{j}\right)$$

Codifique a soma numa função do tipo,

```
double Sum(int* vi, int*vj); //vi, vj= limites de i e j
```

e realize um programa *mainSum.C* donde chame a função.

Exercício 10 : A função *rand()* declarada em <cstdlib> gera um número pseudo-aleatório entre $\mathbf{0}$ e RAND_MAX. Realize um programa em C++ que:

a) gere $\mathbf{1000}$ números aleatórios $\boldsymbol{x}$ entre $\boldsymbol{x_{min}} = \mathbf{5}$ e $\boldsymbol{x_{max}} = \mathbf{55}$.

b) determine o valor de $y = \frac{x}{x-10}$ para cada aleatório.

c) determine o valor médio de x e o seu desvio padrão.

Exercício 11 : Pretende-se calcular a soma dos seguintes valores,

$$0.1 + 0.2 + \cdots + 5.4$$

tendo-se introduzido o seguinte código em C++ num programa:

```
...
double sum = 0;
for (double x=0; x!= 5.5; x += 0.1) {
  sum += x;
}
```

Realize um programa inserindo este código e confirme se este realize o que se pretende.

Programaçao em C/C++: gestão de memória e passagem de parâmetros

Exercício 12 : Realize um programa em que crie em memória *heap*:

a) um *array* de $\mathbf{100}$ números *int*

b) um *array* de $\mathbf{5}$ números *double*

Exercício 13 : Pretende-se obter o valor da função

$$\boldsymbol{f(x) = \sqrt{\sin(2x)}}$$

Escreva em C++ métodos que permitam o cálculo de $\boldsymbol{f(x)}$, em que $\boldsymbol{x}$ á dado em graus. Teste os diferentes métodos realizando um programa *main.C* donde os referencie.

a) o valor de $\boldsymbol{f(x)}$ é retornado pelo método:

```
double func(double);
```

b) o valor de $\boldsymbol{f(x)}$ é retornado por referência:

```
void func(double x, double& f);
```

c) o valor de $\boldsymbol{f(x)}$ é retornado por *pointer*:

```
void func(double x, double* f);
```

d) modifique o método anterior de forma a que o valor em graus de $\boldsymbol{x}$ e o *pointer* não sejam modificáveis no interior do método.

Exercício 14 : Um método em C++ desenvolvido para calcular a soma dos elementos contidos num *array*, possuia a seguinte declaração:

```
void sum(const double* const v,  int n);
```

a) escreva o código em C++ que implemente o método

b) diga se é possível retornar a soma dos elementos no 1º elemento do *array* . Justifique.

c) altere a declaração da função de forma a retornar o valor da soma

Exercício 15 : Realize o seguinte códigos em C++:

a) uma função que inicialize uma variável inteira com um valor aleatório e retorne o seu *pointer* :

```
int* func1();
```

b) uma função que inicialize uma variável inteira com um valor aleatório e retorne o sua referência:

```
int& func2();
```

c) um programa *main.C* que chame as funções $\mathbf{10^6}$ vezes. Verifique se tem *memory leakage* no programa. Liberte a memória que eventualmente tenha alocado.

Exercício 16 : Realize um código C++ no qual se definam métodos que realizem as seguintes tarefas:

a) calcular e retornar o traço da matriz

$$\begin{bmatrix} \mathbf{2} & \mathbf{10} \\ \mathbf{5} & \mathbf{7} \end{bmatrix}$$

```
double Trace(const int** const mx, int n);
```

b) retorne un *array* com os elementos da linha $\boldsymbol{i}$ da matriz $\boldsymbol{m} \times \boldsymbol{n}$,

$$\begin{bmatrix} \mathbf{2} & \mathbf{10} & \mathbf{5} \\ \mathbf{3} & \mathbf{2} & \mathbf{7} \end{bmatrix}$$

```
int* Mrow(int i, const int** const mx, const int m, const int n);
```

c) retorne um array com o resultado da multiplicação de uma matrix $\boldsymbol{M(n \times m)}$ por um vector coluna de $\boldsymbol{V(m)}$ elementos.

$$\boldsymbol{V(m) = \begin{bmatrix} 2 \\ 5 \\ 7 \end{bmatrix}}$$

d) aproveitando o resultado da alínea anterior, determine o resultado da multiplicação das seguintes matrizes:

$$\boldsymbol{\begin{bmatrix} 2 & 10 & 5 \\ 3 & 2 & 7 \end{bmatrix} \times \begin{bmatrix} 5 & 1 & 3 \\ 10 & 1 & 5 \\ 15 & 1 & 4 \end{bmatrix}}$$

Exercício 17 : Realize um método em C++ e teste-o, que receba uma matriz de $\boldsymbol{n \times m}$ elementos e um vector coluna de $\boldsymbol{m}$ componentes e calcule o vector produto, usando a seguinte declaração:

```
void Mmultiply(const double** const mx, const double* const vr,
               int n, int m, double* cosnt pt);
```

com:
mx = matriz $\boldsymbol{n \times m}$
vr = vector coluna
pt = vector resultado

Escreva um programa *main.C* que determine o resultado da alínea c) do problema anterior.

Programaçao em C/C++: biblioteca STL (Standard Template Library)

Exercício 18 : Realize uma função *rand2vec* e um programa *main()* onde se teste a função, cujo objectivo é proceder à geração de $\boldsymbol{n}$ números aleatórios $\boldsymbol{x}$ com valores compreendidos entre $\mathbf{0}$ e $\mathbf{360}$ e que devem ser devovidos ao programa principal usando a estructura *vector<double>*.

```
vector<double> rand2vec(int n);
vector<double>* rand2vecp(int n);
```

Exercício 19 : Realize uma função *array2vec* e o respectivo programa *main()*, cujo objectivo é transferir um *array* de números inteiros para uma estructura STL *vector*.

```
vector<int> array2vec(int, int*);
```

a) aplique a função aos *arrays* seguintes: $\boldsymbol{a} = (\mathbf{1}, \mathbf{10}, \mathbf{5}, \mathbf{6}, \mathbf{9}, \mathbf{3})$ e $\boldsymbol{a} = (\mathbf{2}, \mathbf{5}, \mathbf{5}, \mathbf{7}, \mathbf{3})$

b) elabore numa função *array2vecs*, utilizando a biblioteca *<algorithm>*, a seriação dos valores de cada vector, quer na ordem crescente, quer na ordem decrescente. Retorne o vector ordenado.

```
vector<int> array2vecmaxs(int n, int* a);
```

c) elabore uma função *array2vecmax*, que determine o valor máximo existente em cada um dos *arrays*

```
int array2vecmax(int n, int* a);
```

d) elabore uma função *array2vecfind*, que localize a posição do valor $\mathbf{7}$ em cada um dos arrays

```
int array2vecfind(int n, int* a, int value);
```

e) realize as modificações necessárias nos métodos desenvolvidos de forma a impedir que os valores n e a seja modificado no interior das funções.

f) realize a desalocação de memória que tenha utilizado antes do programa terminar.

Exercício 20 : Realize uma função *kolmogrand* e o respectivo programa *main()* de teste, que:

- faça a geração de números aleatórios x entre os limites *int* x_{min} e x_{max}
- realize a sua distribuição acumulada em 100 intervalos de largura igual

$$f(x_m) = \sum_{i=1}^{m} x_i \qquad m = 1, 2, ..., n$$

Determine o desvio máximo (teste de Kolmogorov) da distribuição acumulada obtida com a esperada.

```
double kolmogrand(int n, int xmin, int xmax);
```

Exercício 21 : Pretende-se realizar uma estrutura *map* usando a biblioteca STL do C++ que armazene matrizes de dimensão $n \times m$ qualquer, usando uma chave do tipo *string* , emparelhada com uma estructura *vector* .

```
map <string, ...> Mmap;
```

a) defina uma estrutura STL capaz de armazenar as matrizes abaixo, definindo uma função que devolva a estrura STL

```
---? GetMatrix(int nrows, int mcols, int** M);
```

$$A = \begin{bmatrix} 2 & 10 & 5 \\ 3 & 2 & 7 \end{bmatrix} \qquad B = \begin{bmatrix} 5 & 1 & 3 \\ 10 & 1 & 5 \\ 15 & 1 & 4 \end{bmatrix} \qquad C = \begin{bmatrix} 5 & 1 \\ 10 & 2 \\ 15 & 1 \end{bmatrix}$$

b) armazene as três matrizes A, B e C sob as chaves "A", "B"e "C"no mapa *Mmap*.

c) crei uma função *Mmapfind* que procure uma chave e retorne a matriz.

```
---? Mmapfind(string c);
```

Programaçao em C/C++: classes

Exercício 22 : Construi-se uma classe genérica *pessoa* que pretende possuir as características associadas às pessoas (aqui tratadas como objectos!). A declaração da classe é a seguinte:

```
class pessoa {

  public:
  //constructor (nome do aluno, data de nascimento)
   pessoa(string, unsigned int);
   void SetName(string); //set name
   void SetBornDate(unsigned int); //nascimento
   string GetName(); //get name
   unsigned int GetBornDate();
   virtual void Print(); // print

  private:
   string name; //nome
   unsigned int DataN; //data de nascimento
}
```

a) Implemente o código associado aos *function members* da classe escrevendo sempre em cada método o código necessário que imprima o nome da classe e do método *[class::method]* de forma a sabermos quando é chamado. Compile o código e veja se não existem erros.

b) Para testar o código da classe realize um programa *main* onde construa um *array* de 10 objectos pessoa:

```
pessoa P[10];
```

Que constructor é chamado? Corrija o código e declaração da classe caso existam erros.

c) Admita agora que pretendia construir um *array* de N pointers para objectos pessoa. Construa uma função que retorne o ponteiro para o *array* .

```
pessoa** DoArray(int N);
```

Inclua a informação do nome dos alunos e a sua data de nascimento.

Exercício 23 : Construa agora uma classe *alunoIST* que derive da classe *pessoa*. A nova classe deverá ter novos *data members* como por exemplo:
- Número do aluno: *int number*
- Curso: *string branch*

e novas funções que interajam com os novos data members.

```
class alunoIST : public pessoa {
  public:
   //constructor (numero e nome do aluno)
   alunoIST(int number, string curso);
   void SetNumber(int);
   int GetNumber();
   void Print();
   ...
  private:
   int number;
   string branch;
}
```

a) Implemente o código da nova classe.

b) Construa um *array* de objectos *alunoIST* com conteúdo.

c) Construa uma função *[void Dummy(pessoa**)]* que receba um *pointer* genérico para um *array* de pointers de objectos *pessoa*. No interior da função circule sobre todos os objectos e chame a função membro *Print()*. A função que é chamada pertence a que class (*pessoa* ou *alunoIST*)?

Exercício 24 : Na sequência das classes anteriores podemos prosseguir o exercício criando agora a classe *Turma* que não necessita de derivar de nenhuma das classes anteriores, antes usando os objectos da classe *alunoIST*. Uma declaração ainda que incompleta da classe seria:

```
class Turma {
  public:
   Turma(string, int n); //nome da turma, num de alunos
   ~Turma(); //destructor
  private:
   alunoIST **va; //pointer to array de pointers de objectos
   int Nalunos;
}
```

a) Complete a declaração da classe de forma a incluir os seguintes métodos:
   - *default constructor*
   - *copy constructor*
   - *copy assignment*
   - métodos *void SetAluno(alunoIST* const* e *alunoIST* FindAluno(int numero)*
   - método *int GetNalunos()*

b) Implemente o código da classe e em particular o método *alunoIST* FindAluno(int)* deveria ser implementado da forma mais eficaz usando a procura dicotómica.

c) Construa um programa *main()* onde possa testar a classe definindo a turma de MEFT T21.