
Exercícios de Física Computacional

Mestrado em Engenharia Física-Tecnológica (MEFT)

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

1^a série

de problemas de Física Computacional

- ☐ Programação em C/C++: revisão
- ☐ Programação em C/C++: gestão de memória e passagem de parâmetros
- ☐ Programação em C/C++: biblioteca STL (Standard Template Library)
- ☐ Programação em C/C++: classes

Exercícios de Física Computacional

Programação em C/C++: revisão

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:
 $(5, -2)$, $(5, 20)$ e $(10, 55)$.

Exercício 2 : Considere a fórmula matemática,

$$z(x) = x + f(x)$$

com,

$$f(x) = \sin^2(x)$$

Realize um programa em C/C++ que calcule $z(x)$ para os valores de $x = 0.4, 2.1, 1.5$ e imprima os resultados no monitor do computador.

Nota: a função $f(x)$ 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 π com precisão *float* e *double* a partir da função **atan()**. Compare o valor obtido com o valor exacto de π . 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
<code>numeric_limits<type>::max()</code>	largest type value
<code>numeric_limits<type>::min()</code>	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, x^2 , usando somente as operações:

- adição, subtracção, multiplicação ($\times 2$)
- 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 (1, 5000).

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

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

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

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 0 e `RAND_MAX`. Realize um programa em C++ que:

- gere **1000** números aleatórios x entre $x_{min} = 5$ e $x_{max} = 55$.

- 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,

$$0.1 + 0.2 + \dots + 5.4$$

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.

Programação 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 100 números *int*
- b) um *array* de 5 números *double*

Exercício 13 : Pretende-se obter o valor da função

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

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

- a) o valor de $f(x)$ é retornado pelo método:

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

- b) o valor de $f(x)$ é retornado por referência:

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

- c) o valor de $f(x)$ é retornado por *pointer*:

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

- d) modifique o método anterior de forma a que o valor em graus de 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*, possuía 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 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

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

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

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

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

```
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 $M(n \times m)$ por um vector coluna de $V(m)$ elementos.

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

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

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

Exercício 17 : Realize um método em C++ e teste-o, que receba uma matriz de $n \times m$ elementos e um vector coluna de 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* const pt);
```

com:

mx = matriz $n \times m$

vr = vector coluna

pt = vector resultado

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

Programação 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 n números aleatórios x com valores compreendidos entre 0 e 360 e que devem ser devovidos ao programa principal usando a estrutura *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 estrutura STL *vector*.

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

- a) aplique a função aos *arrays* seguintes: $a = (1, 10, 5, 6, 9, 3)$ e $a = (2, 5, 5, 7, 3)$
- b) elabore numa função *array2vecs*, utilizando a biblioteca *<algorithm>*, a serialização 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 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

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

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 estrutura *vector* .

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

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

```
---? GetMatrix(int nrow, int mcol, int** M);
```

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

- b) armazene as três matrizes A, B e C sob as chaves "A", "B" e "C" no mapa *Mmap*.
- c) crie uma função *Mmapfind* que procure uma chave e retorne a matriz.

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

Programação 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* .

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

- Implemente o código da nova classe.
- Construa um *array* de objectos *alunoIST* com conteúdo.
- Construa uma função `[void Dummy(peessoa**)]` que receba um *pointer* genérico para um *array* de *pointers* de objectos *peessoa*. No interior da função circule sobre todos os objectos e chame a função membro *Print()*. A função que é chamada pertence a que class (*peessoa* 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.