# Aula 03:
# - Funções e procedimentos
# - Vetores e matrizes

Prof. Jesús P. Mena-Chalco
jesus.mena@ufabc.edu.br

3Q-2017

# Funções

# Funções (modularidade)

- Consiste em dividir uma atividade em componentes, rotulados e endereçáveis

```c
#include <stdio.h>

int maior(int a, int b) {
    if (a<b)
        return b;
    else
        return a;
}

void main() {
    printf("%d\n", maior(3, 10));
    printf("%d\n", maior(10, 3));
    printf("%d\n", maior(10, maior(5, 40)));
}
```

As variáveis locais **não** são conhecidas fora da função

- Uma função devolve um valor.
- A função invocadora (main) é **suspensa** quando executar a outra função.

# Exercício 1

Crie uma função que permita calcular uma aproximação de PI usando a serie de Gregory:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

A função deve aceitar um parâmetro que represente o número de termos a ser considerado na somatório.

```
4.000000
3.041840
3.131593
3.141493
```

```c
#include <stdio.h>

double pi(int t) {



}

void main() {
    printf("%lf\n", pi(1));
    printf("%lf\n", pi(10));
    printf("%lf\n", pi(100));
    printf("%lf\n", pi(1000));
}
```

# Exercício 1

Crie uma função que permita calcular uma aproximação de PI usando a serie de Gregory:

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

A função deve aceitar um parâmetro que represente o número de termos a ser considerado na somatório.

```
4.000000
3.041840
3.131593
3.140593
```

```c
#include <stdio.h>

double pi(int t) {
    int i, sinal=1;
    double soma = 0;

    for (i=1; i<=t; i++) {
        soma += 1.0/(2*i-1)*sinal;
        sinal *= -1;
    }

    return 4*soma;
}

void main() {
    printf("%lf\n", pi(1));
    printf("%lf\n", pi(10));
    printf("%lf\n", pi(100));
    printf("%lf\n", pi(1000));
}
```

# Exercício 2

Modifique a função anterior de tal forma que faça a somatória dos termos maiores ou iguais a **x**.

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

A função deve aceitar um parâmetro que represente o valor x (precisão).

```c
#include <stdio.h>

double pi(double precisao) {









}

void main() {
    printf("%.10lf\n", pi(0.01));
    printf("%.10lf\n", pi(0.0000001));
}
```

3.1215946526
3.1415924536

# Exercício 2

Modifique a função anterior de tal forma que faça a somatória dos termos maiores ou iguais a **x**.

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots$$

A função deve aceitar um parâmetro que represente o valor x (precisão).

```c
#include <stdio.h>

double pi(double precisao) {
    int i=1, sinal=1;
    double soma = 0;

    while( 1.0/(2*i-1)>precisao ) {
        soma += 1.0/(2*i-1)*sinal;
        sinal *= -1;
        i++;
    }

    return soma*4;
}

void main() {
    printf("%.10lf\n", pi(0.01));
    printf("%.10lf\n", pi(0.0000001));
}
```

3.1215946526
3.1415924536

Desafio: Quantos termos são necessários?

# Procedimentos?

# Procedimentos

Na linguagem C existem apenas funções, mas saiba que existem outras linguagens que aceitam funções e procedimentos (em pascal: Function e Procedure):

- **Uma função devolve sempre um valor.**
- **Um procedimento não devolve qualquer valor.**

A forma de invocar funções e procedimentos é diferente:

- Funções: `x = pi(0.001)`
- Procedimentos: `printf("%f", x)`

# Procedimentos

Uma função que devolve “**void**” é chamada de **procedimento**.

**void type**

void type means no value. This is usually used to specify the type of functions.

# Procedimento: exemplo

```
1  #include <stdio.h>
2
3  void linha(int n) {
4      int i;
5
6      for (i=0; i<n; i++)
7          printf("*");
8      printf("\n");
9  }
10
11 void main() {
12     linha(10);
13     linha(20);
14 }
```

# Vetores

# Armazenar 10 inteiros em um programa...

- Usando variáveis:

```
int a0 = 6;
int a1 = 30;
int a2 = 82;
int a3 = 0;
int a4 = 100;
int a5 = 8;
int a6 = 14;
int a7 = 83;
int a8 = 11;
int a9 = 20;
```

- Usando um vetor:

```
int a[] = {6, 30, 82, 0, 100, 8, 14, 83, 11, 20};
```

Para acessar a um elemento, use um índice.

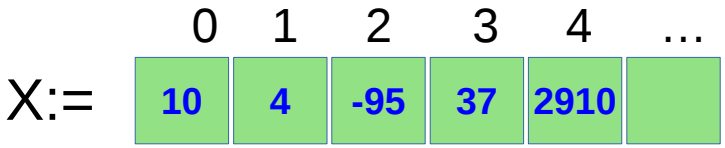
a[0] → 6

a[4] → 100

# Vetores

Os elementos de um vetor são armazenados/**alocados de forma consecutiva** na memória.

Os elementos são **acessados por seu índice** dentro do vetor.

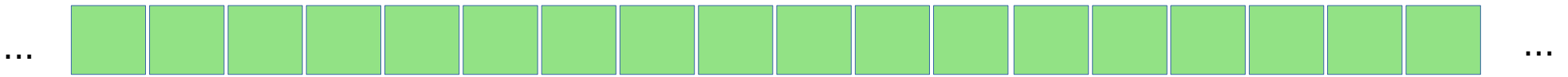


# Memória (hardware)

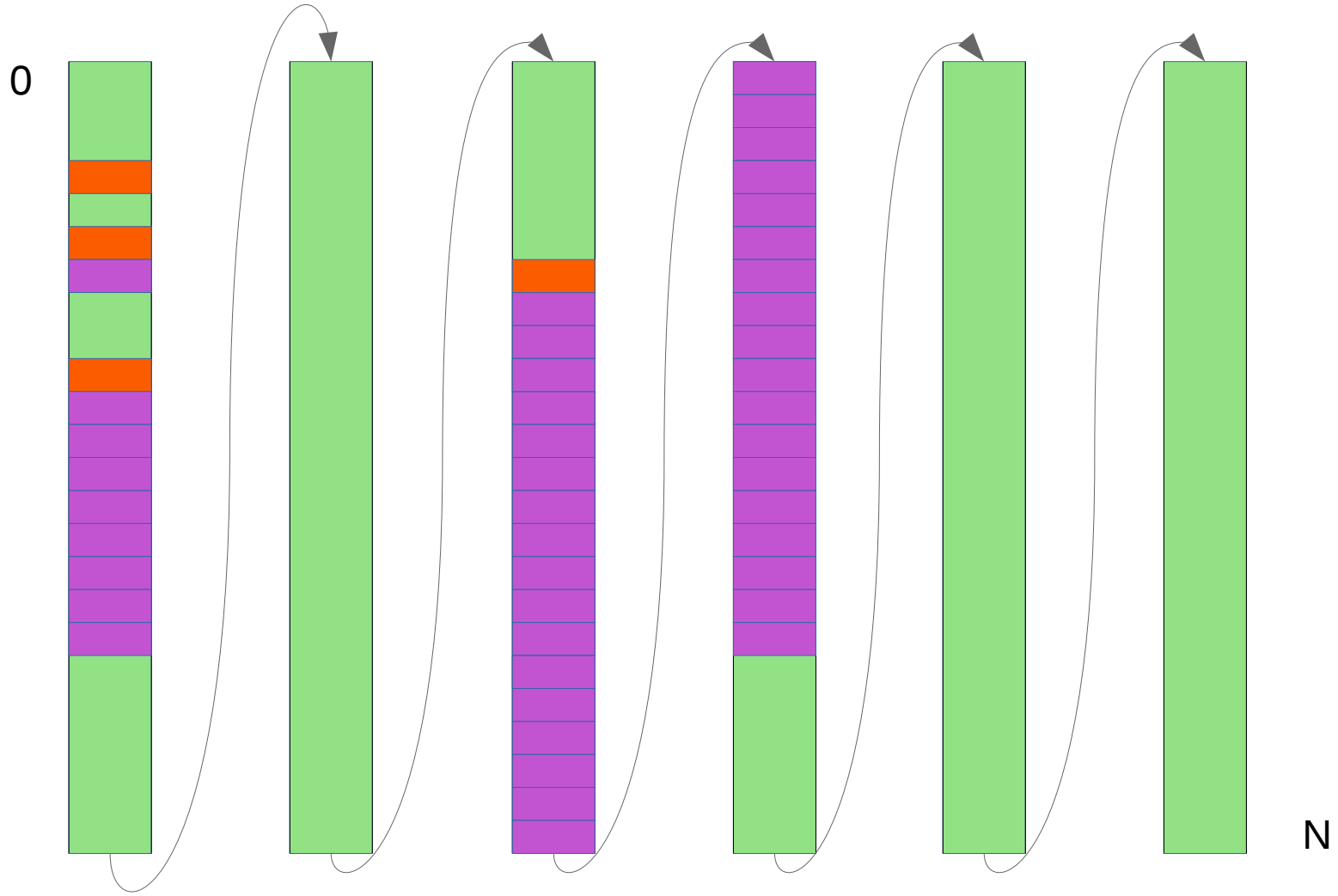
RAM

HDD

SSD

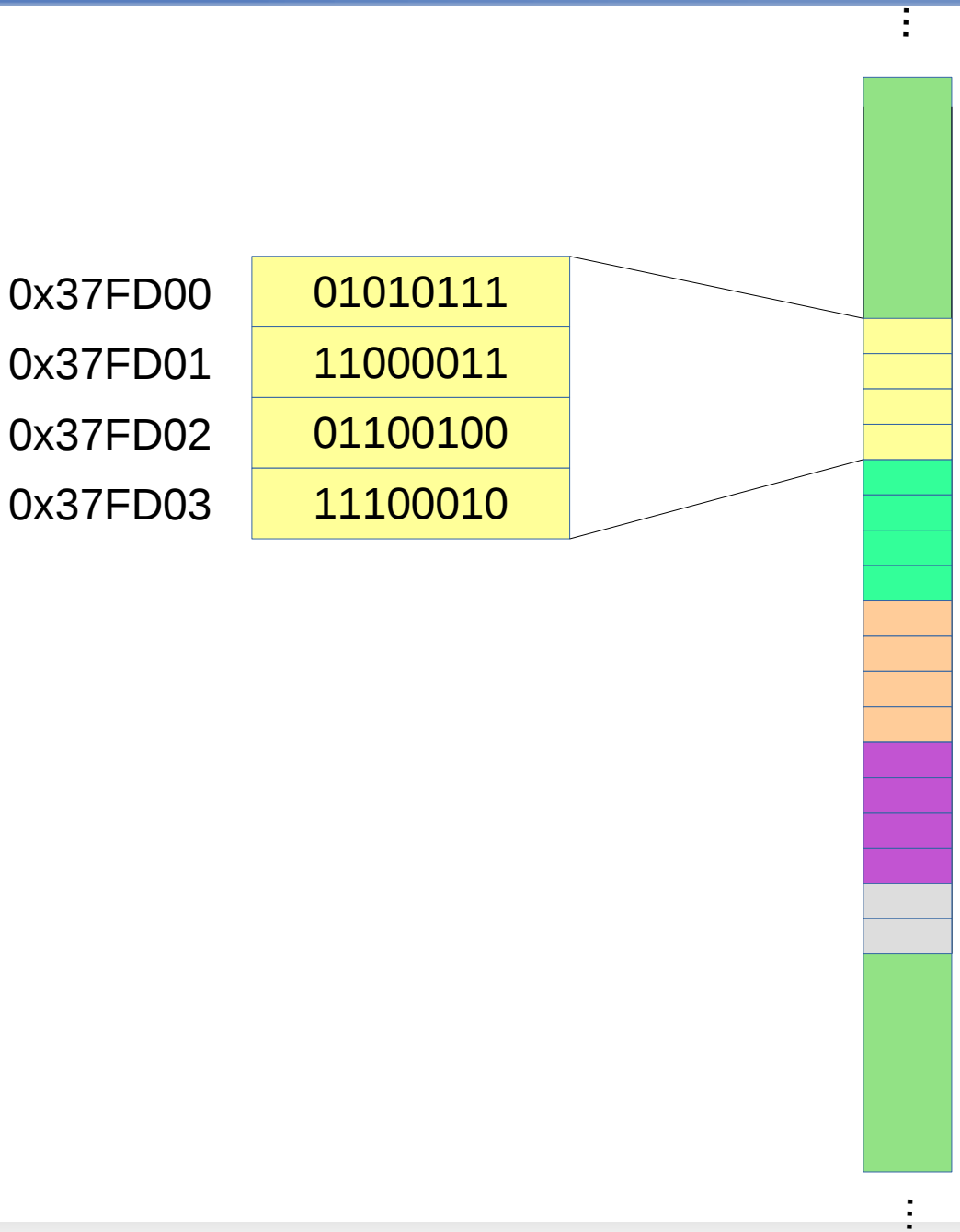


# Memória (hardware)




Geralmente o endereço do “objeto“ é o endereço do 1ro byte.

# Terminologia



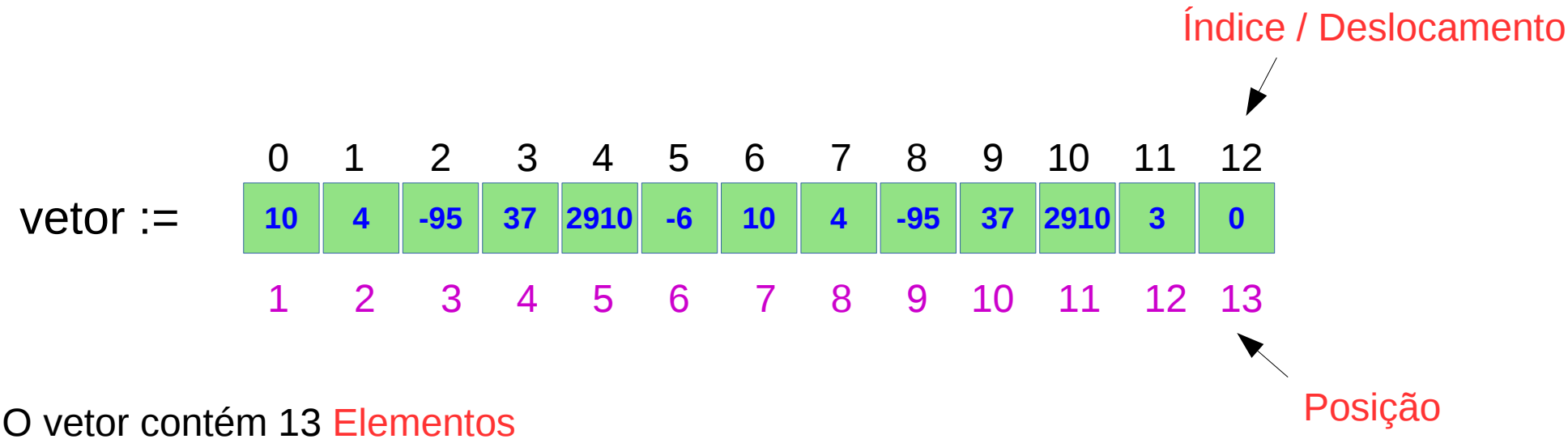

4 bytes

| Category | Types | Size (bits) | Mi |
|---|---|---|---|
| Integer | byte | 8 | -128 |
| | char | 16 | 0 |
| | short | 16 | $-2^{15}$ |
| | int | 32 | $-2^{31}$ |
| | long | 64 | $-2^{63}$ |
| Floating-point | float | 32 | $2^{-149}$ |
| | double | 64 | $2^{-107}$ |

# Vetores

Declaração de uma variável que representa um vetor de 13 inteiros

```
int vetor[13];

vetor[4]  = 2910;
```




O vetor contém 13 Elementos

# Vetores

```c
#include <stdio.h>

void main() {

    int i, vetor[13];

    for (i=0; i<13; i++)
        printf("%d\n", vetor[i]);

}
```

# Vetores

```c
#include <stdio.h>

void main() {

    int i, vetor[13];

    for (i=0; i<13; i++)
        printf("%d\n", vetor[i]);

}
```

```
0    1
1    0
2    4195773
3    0
4    -1939313376
5    32764
6    0
7    0
8    4195696
9    0
10   4195392
11   0
12   -1939313152
```

# Vetores

```
1  #include <stdio.h>
2
3  void main() {
4
5      int i, vetor[13];
6
7      for (i=-5; i<15; i++)
8          printf("%d\n", vetor[i]);
9
10 }
```

# Vetores

```c
#include <stdio.h>

void main() {

    int i, vetor[13];

    for (i=-5; i<15; i++)
        printf("%d\n", vetor[i]);

}
```

```
-5    32741
-4    771953096
-3    32741
-2    0
-1    -1
0     1
1     0
2     4195773
3     0
4     -1699835760
5     32764
6     0
7     0
8     4195696
9     0
10    4195392
11    0
12    -1699835536
13    32764
14    0
```

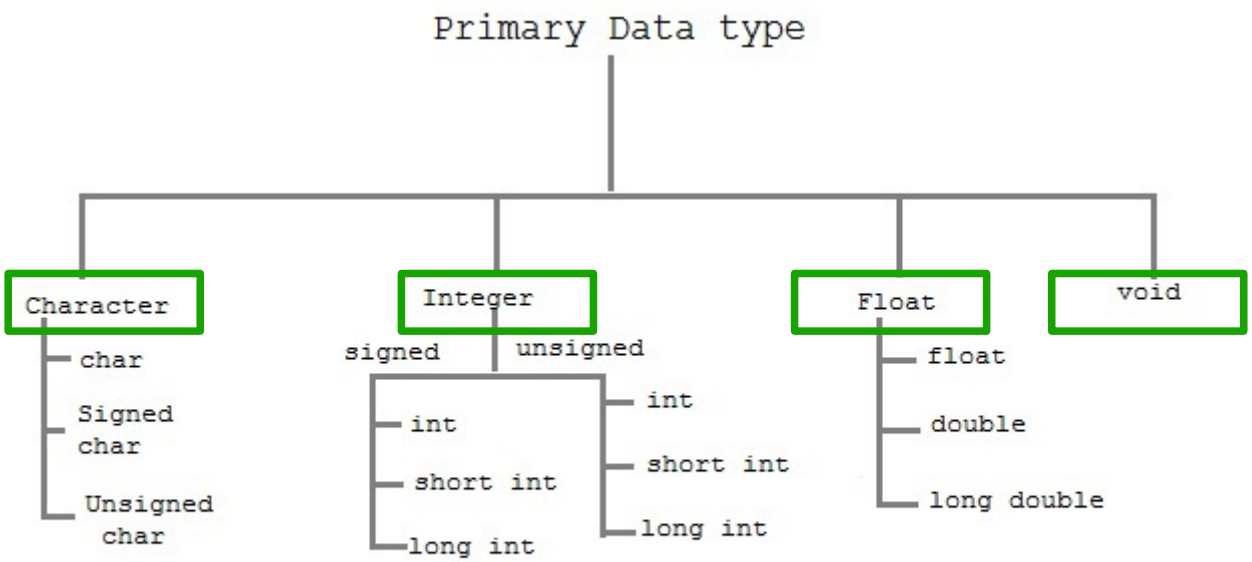
# Em concreto

Um vetor é uma coleção de valores.

**Três importantes características:**

- Os vetores representam um grupo de dados relacionados.
- Todos os dados devem ter o mesmo tipo.
- O tamanho do vetor é definido na sua criação/definição.

# Vetores




(*) Fonte: http://www.studytonight.com/c/datatype-in-c.php

# Vetores

```c
#include <stdio.h>

void main() {

    int       v1[10];
    short int v2[10];
    long int  v3[10];
    char      v4[10];
    double    v5[10];

    printf("%ld\n", sizeof(v1));
    printf("%ld\n", sizeof(v2));
    printf("%ld\n", sizeof(v3));
    printf("%ld\n", sizeof(v4));
    printf("%ld\n", sizeof(v5));
}
```

# Vetores

```c
#include <stdio.h>

void main() {

    int       v1[10];
    short int v2[10];
    long int  v3[10];
    char      v4[10];
    double    v5[10];

    printf("%ld\n", sizeof(v1));
    printf("%ld\n", sizeof(v2));
    printf("%ld\n", sizeof(v3));
    printf("%ld\n", sizeof(v4));
    printf("%ld\n", sizeof(v5));
}
```

```
40
20
80
10
80
```

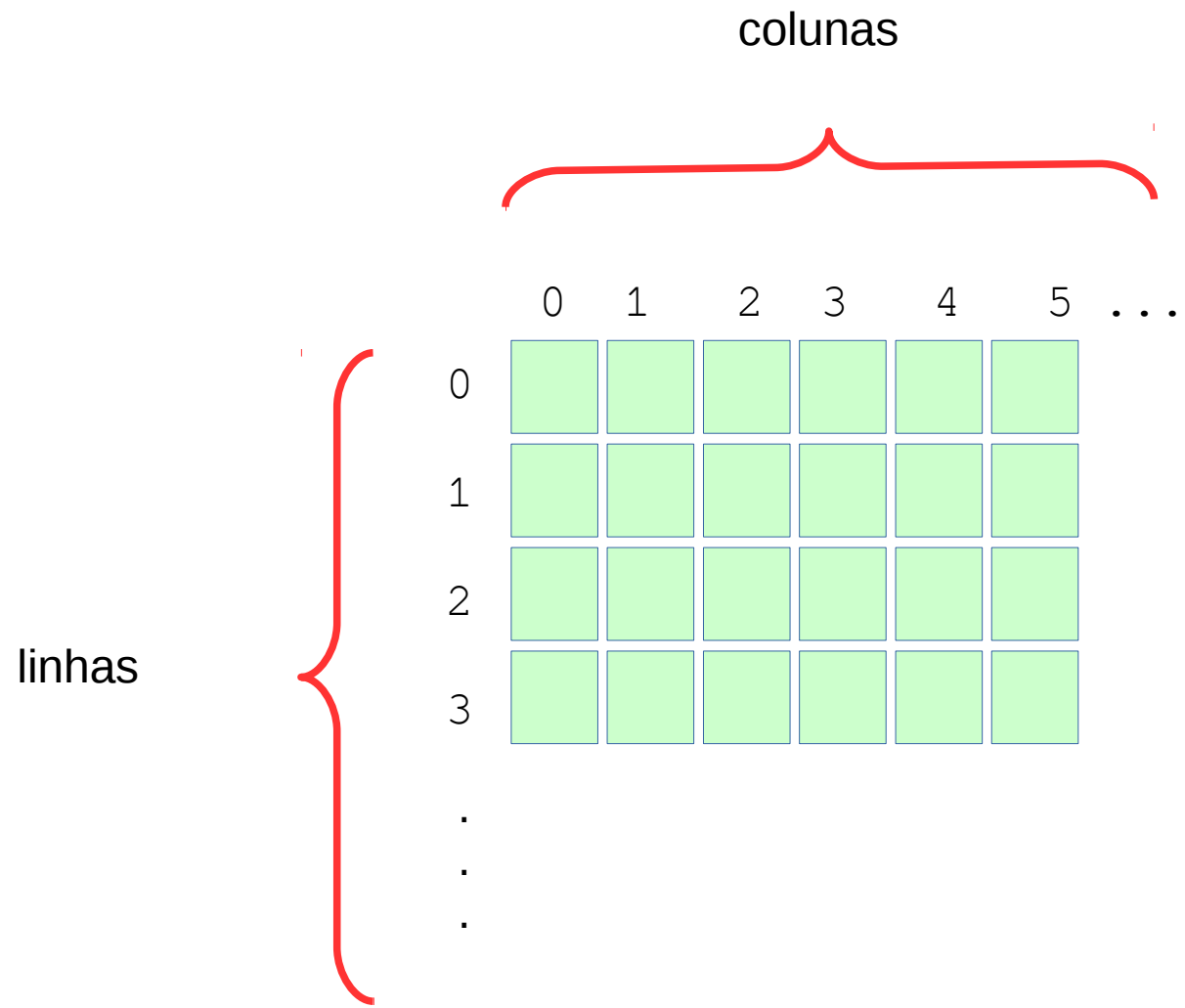
# Vetores

```c
#include <stdio.h>

void main() {

    int       v1[10];
    short int v2[10];
    long int  v3[10];
    char      v4[10];
    double    v5[10];

    printf("%ld\n", sizeof(v1)/sizeof(v1[0]));
    printf("%ld\n", sizeof(v2)/sizeof(v2[0]));
    printf("%ld\n", sizeof(v3)/sizeof(v3[0]));
    printf("%ld\n", sizeof(v4)/sizeof(v4[0]));
    printf("%ld\n", sizeof(v5)/sizeof(v5[0]));
}
```

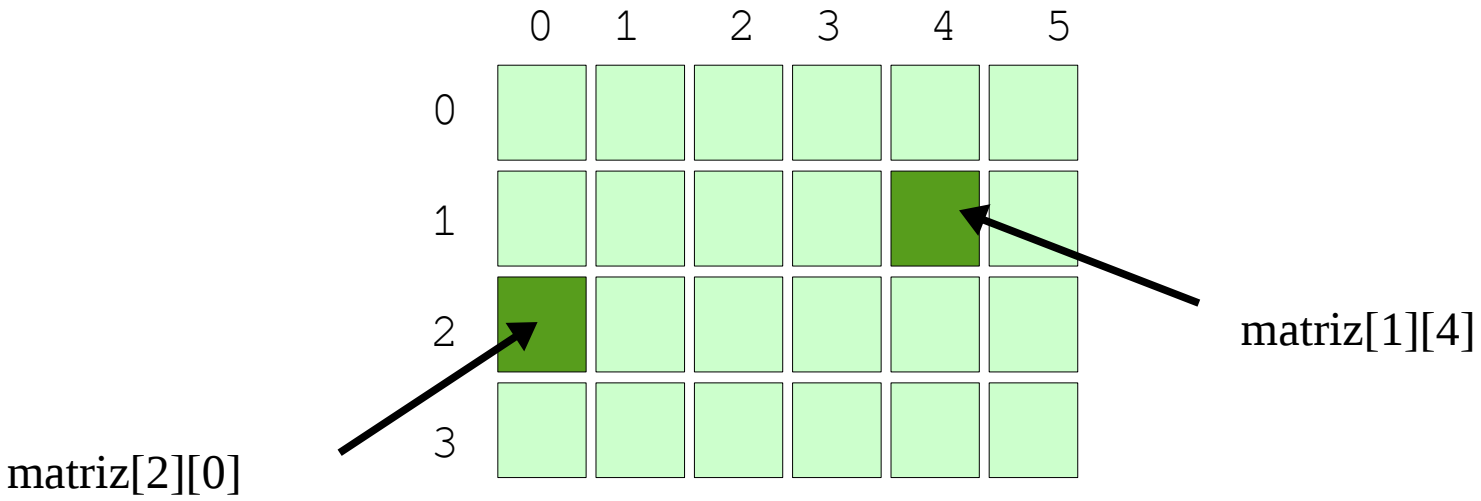
```
10
10
10
10
10
```

# Matrizes

# Matriz bidimensional



# Matriz bidimensional



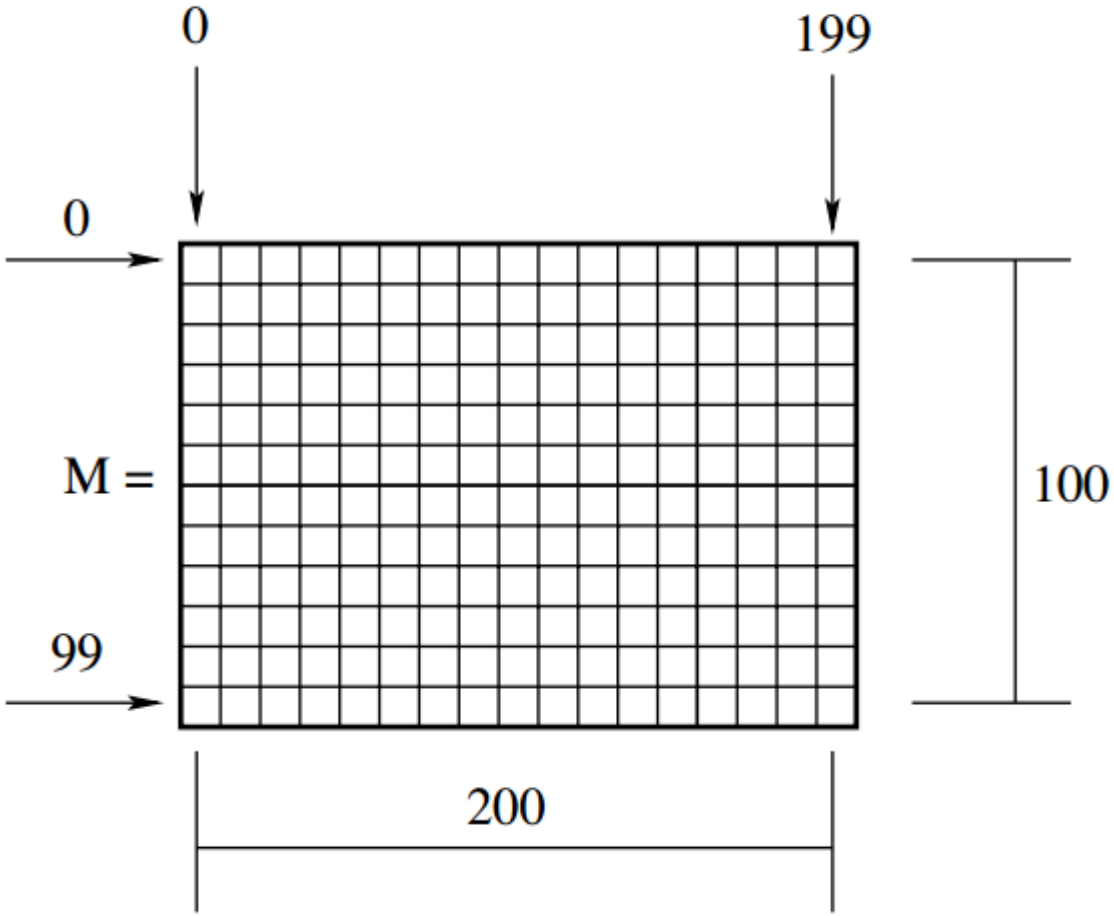
# Matriz bidimensional

```c
#include <stdio.h>

void main() {

    int i, j;
    int matriz[5][7] = {0};

    for (i=0; i<5; i++) {
        for (j=0; j<7; j++) {
            printf("%d", matriz[i][j]);
        }
        printf("\n");
    }

}
```

```
0000000
0000000
0000000
0000000
0000000
```

```
int M[100][200];
```

**Declara uma matriz M de 100 linhas com 200 colunas (20mil inteiros)**



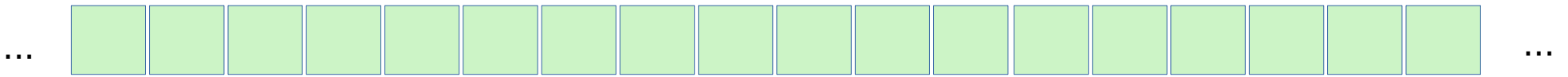

A memória do computador é linear!

# Memória (hardware)

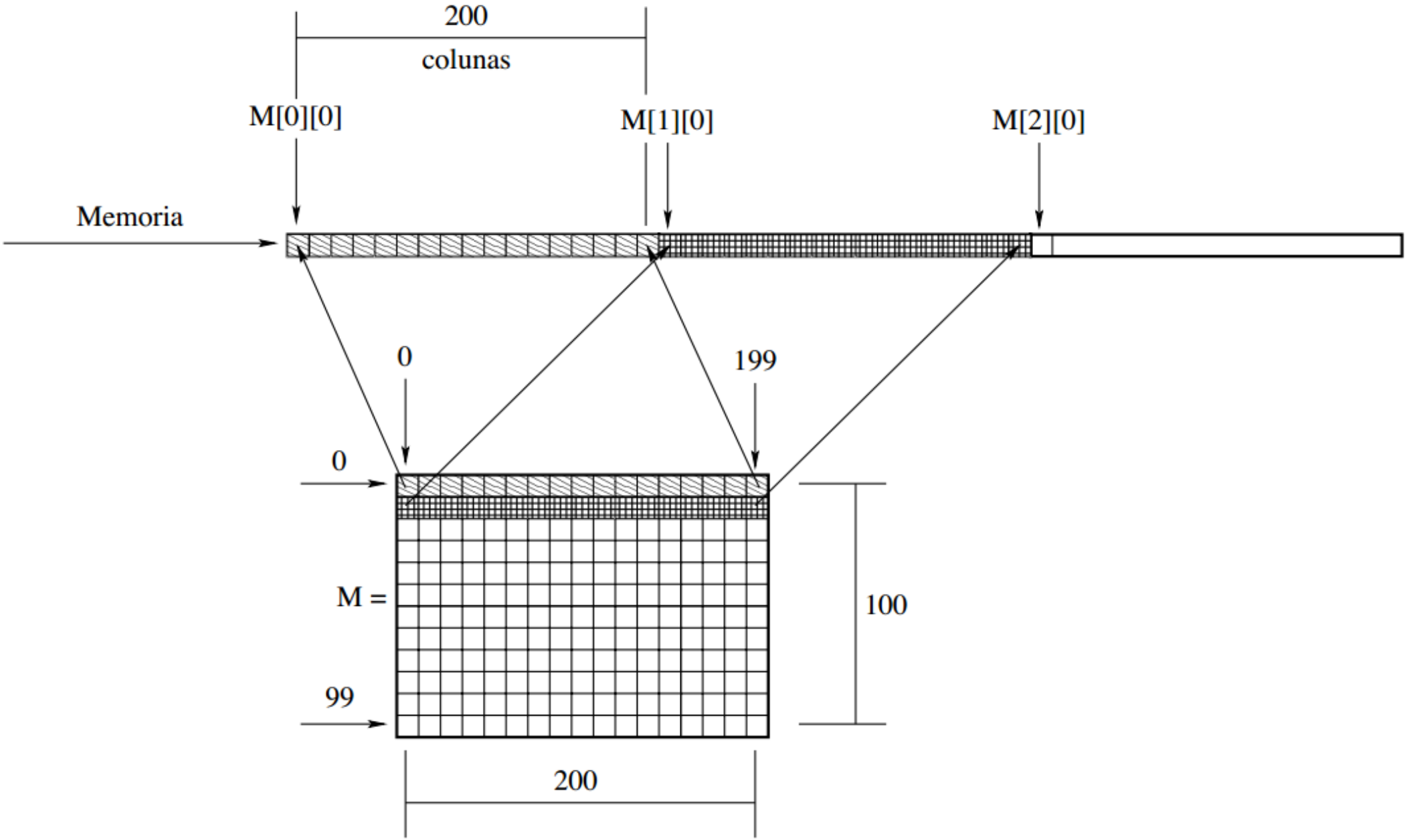
RAM

HDD

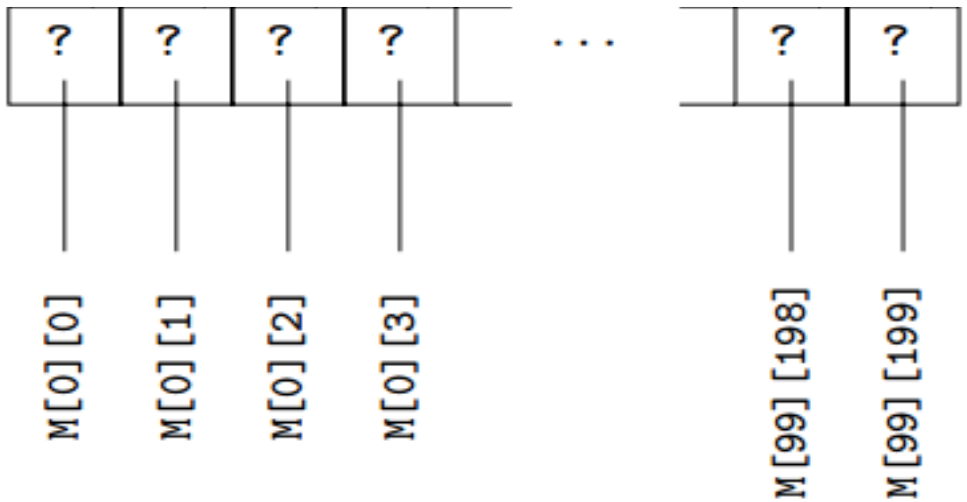
SSD



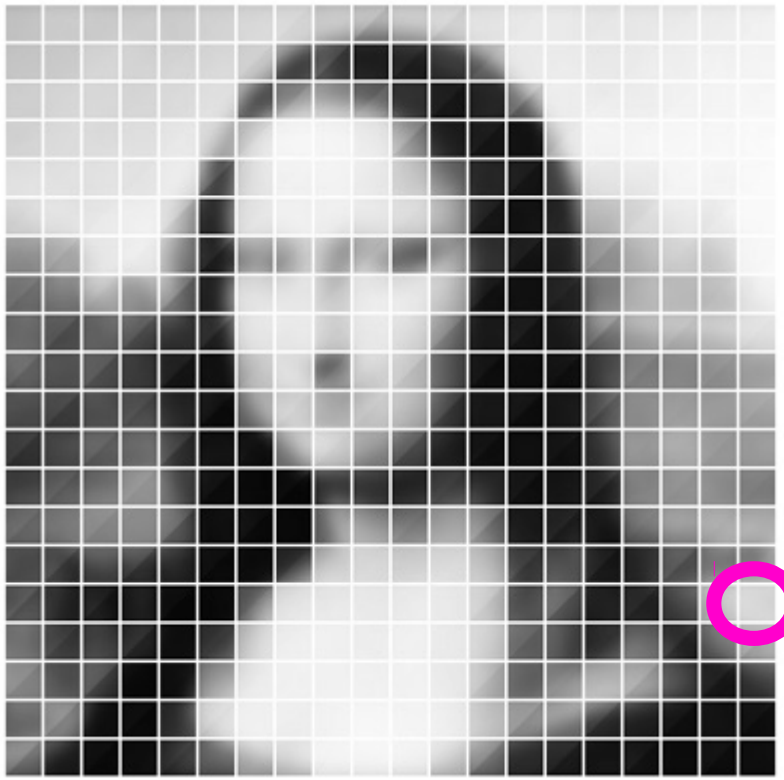
# Estrutura da matriz na memória do computador



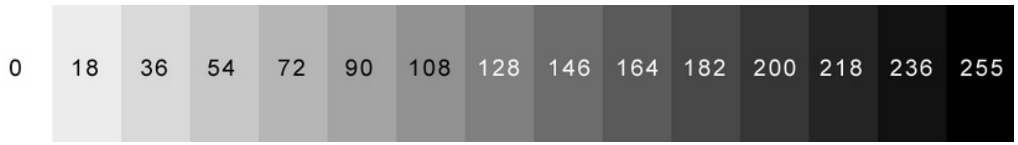
# Disposição dos 20mil elementos da matriz M na memória



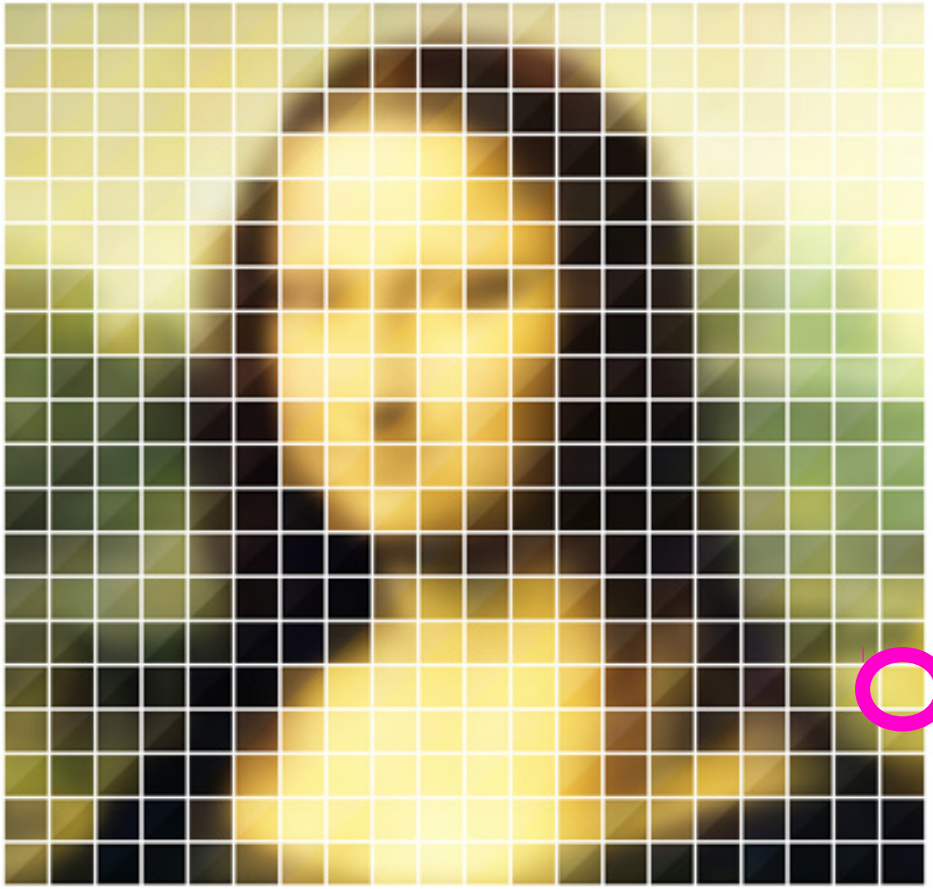
# Matriz bidimensional (imagem em níveis de cinza)



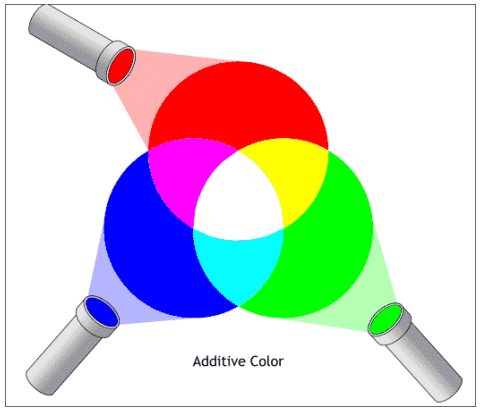
Nível=18



# Matriz tridimensional (imagem em RGB)

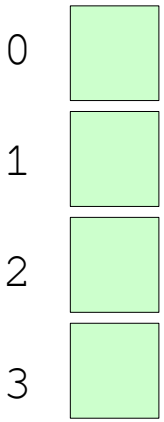


{Red, Green, Blue}

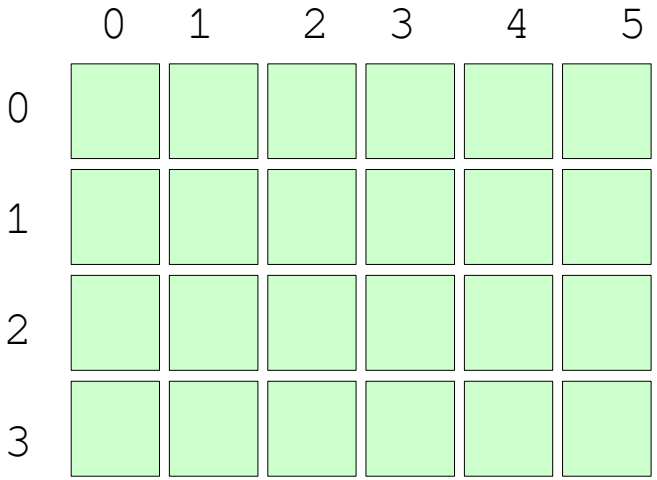


# Matrizes

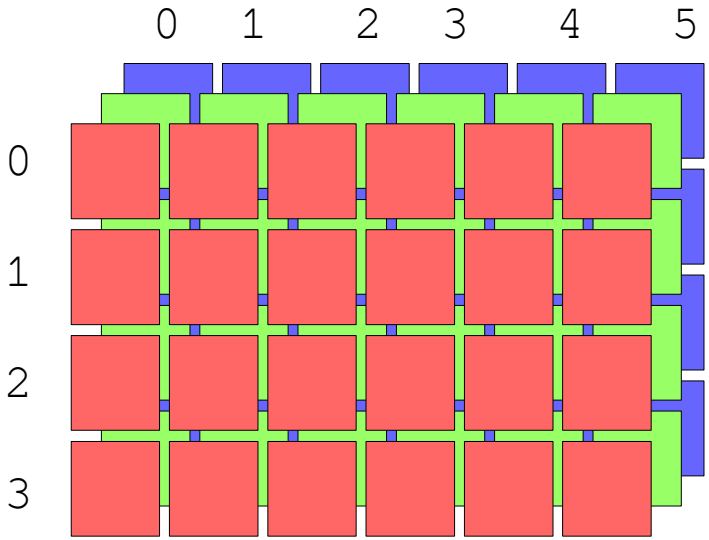
**Matriz unidimensional (vetor/Array)**




**Matriz bidimensional (2D)**

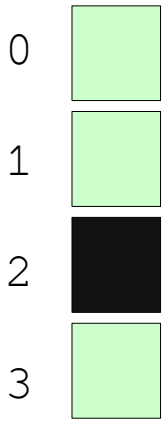



**Matriz tridimensional (3D)**



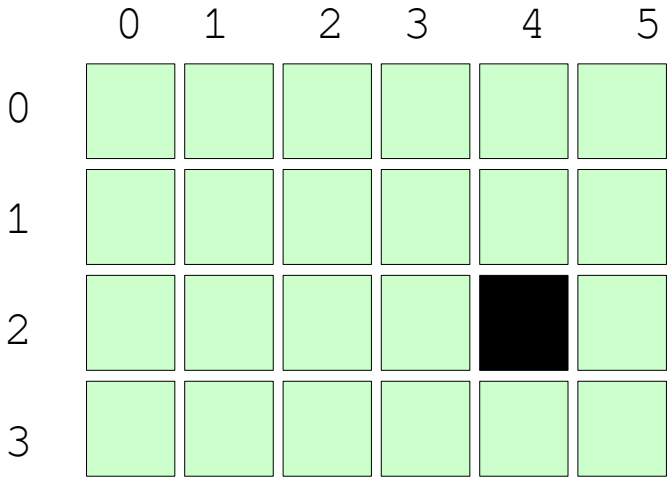
# Matrizes

## Matriz unidimensional (vetor/Array)



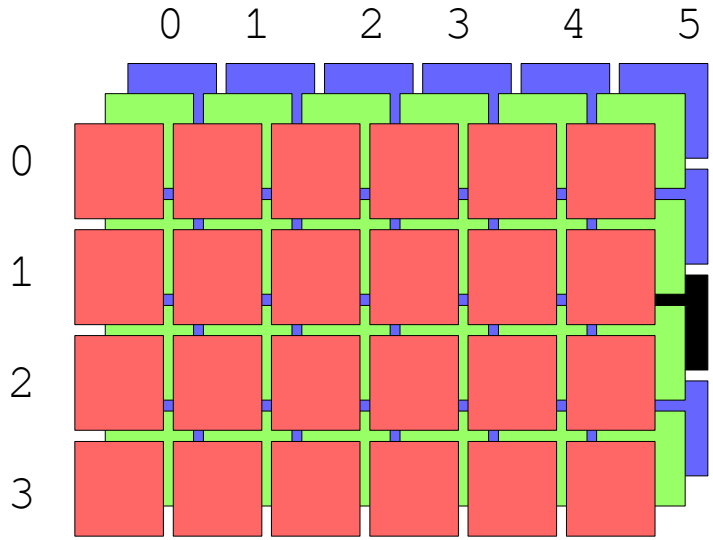

`M[2]`

## Matriz bidimensional (2D)




`M[2][4]`

## Matriz tridimensional (3D)



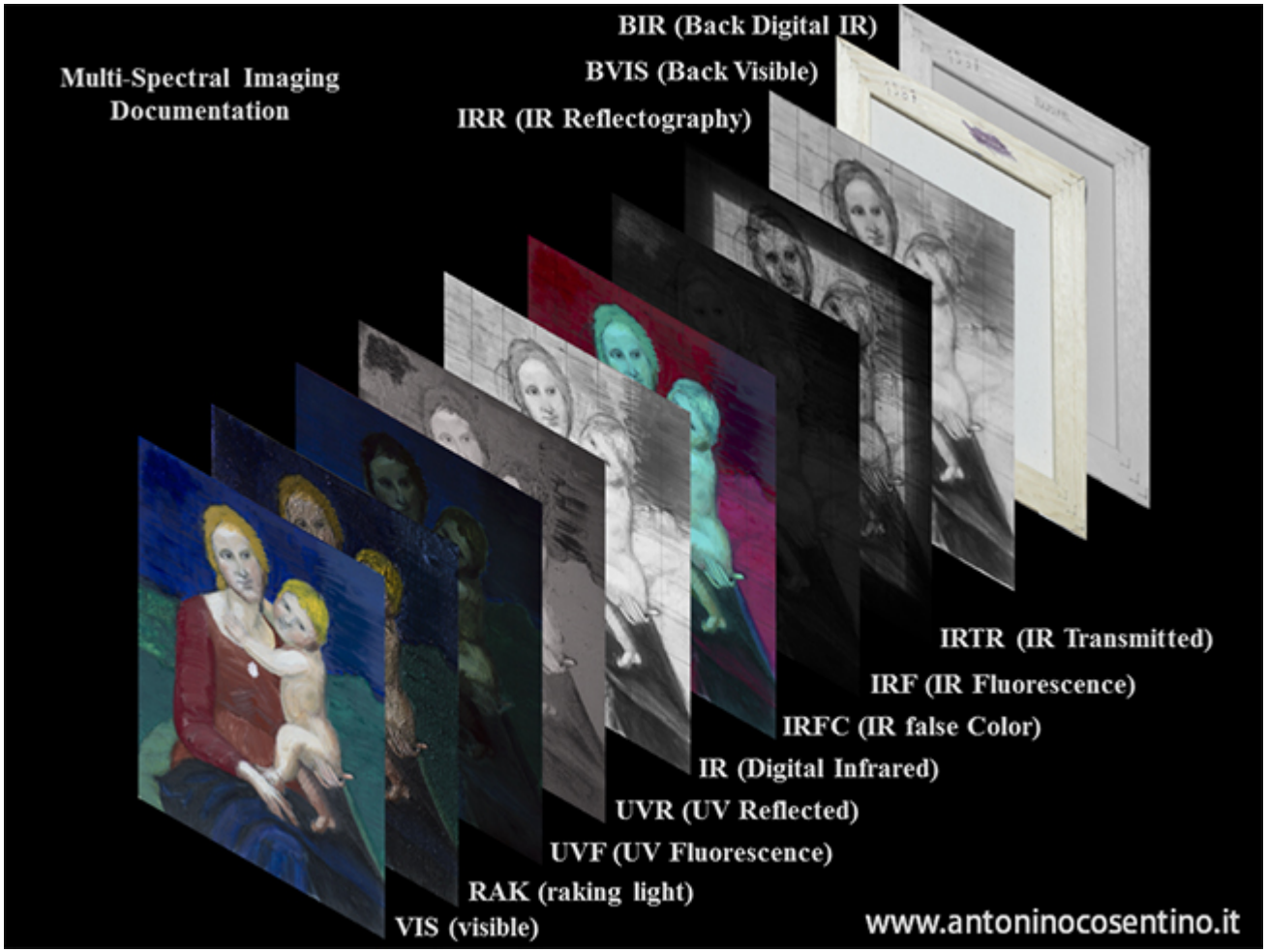

`M[2][5][2]`

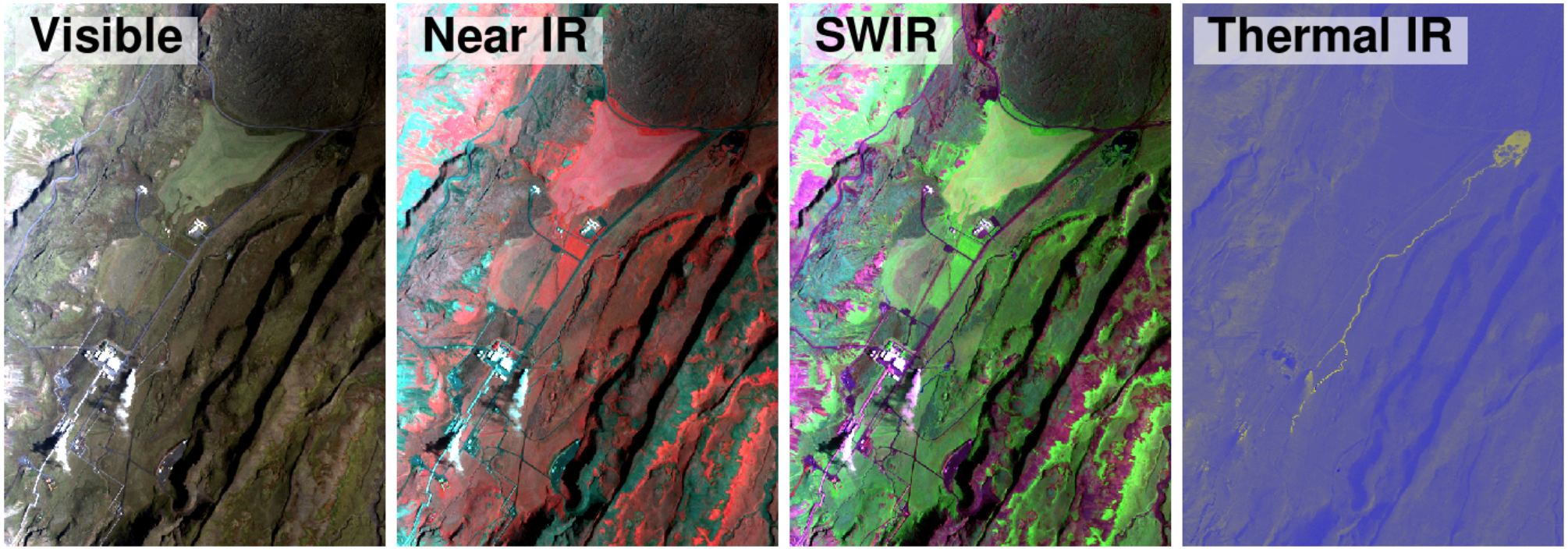
```
int M[100][200][3];
```

# Dimensões?

```c
#include <stdio.h>

void main() {

    int M[100][200][3];

    printf("%ld\n", sizeof(M)/sizeof(M[0]));
    printf("%ld\n", sizeof(M[0])/sizeof(M[0][0]));
    printf("%ld\n", sizeof(M[0][0])/sizeof(M[0][0][0]));

}
```

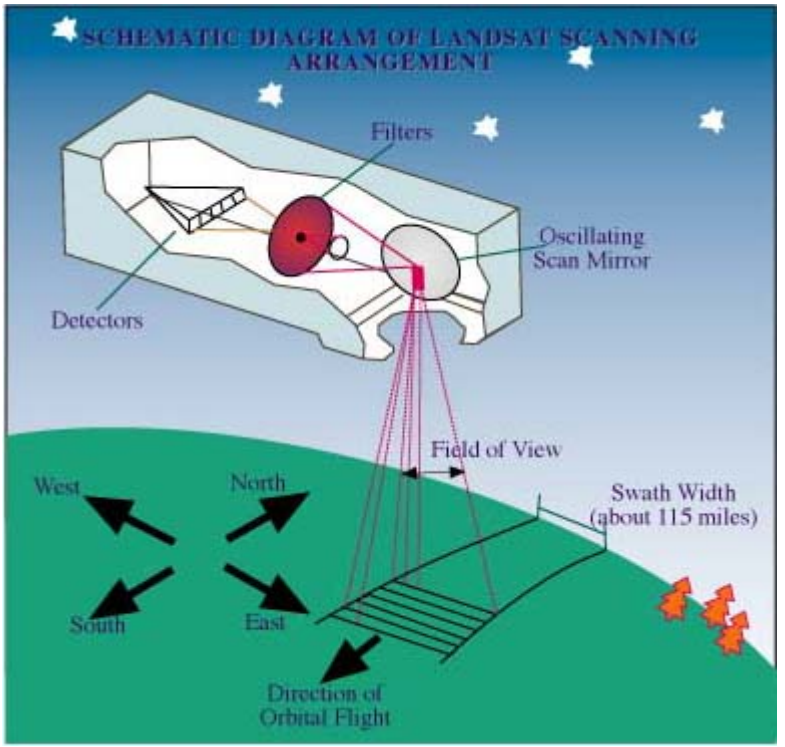
```
100
200
3
```


Multi-Spectral Imaging
Documentation
BIR (Back Digital IR)
BVIS (Back Visible)
IRR (IR Reflectography)
IRTR (IR Transmitted)
IRF (IR Fluorescence)
IRFC (IR false Color)
IR (Digital Infrared)
UVR (UV Reflected)
UVF (UV Fluorescence)
RAK (raking light)
VIS (visible)
www.antoninocosentino.it




Processamento de imagens multiespectrais (imagens do mesmo objeto, mas capturadas com diferentes comprimentos de onda **eletromagnéticas**)
Os aparelhos consideram diferentes comprimentos de onda para cada camada.

http://all-geo.org/volcan01010/2013/01/processing-arsf-remote-sensing-data-with-open-source-gis-tools/




How do spaceborne sensors work?
http://geosun.sjsu.edu/paula/285/285/as_sem.htm

# Atividade em aula

# Lista 1

**Disciplina: Programação Estruturada**
**Turmas: A1 e A2 – Noturno**

Prof. Dr. Jesús P. Mena-Chalco
Assistente: Rafael J. P. Damaceno

Universidade Federal do ABC

**Lista 1 - Deadline: 04/10 (23h50)**

Nesta lista serão trabalhos os tópicos vetores e matrizes. Usaremos a Plataforma URI para a avaliação de todos os problemas da lista: https://www.urionlinejudge.com.br

**Vetores**

1. Problema 1172. Substituição em Vetor I
2. Problema 1174. Seleção em Vetor I
3. Problema 1178. Preenchimento de Vetor III

**Matrizes**

4. Problema 1181. Linha na Matriz
5. Problema 1182. Coluna na Matriz
6. Problema 1184. Abaixo da Diagonal Principal
7. Problema 1190. Área Direita

**Observações:**

- A linguagem de programação considerada é, exclusivamente, C ou C++.
- Será utilizado um programa especializado para detecção de plágio em todas as submissões.

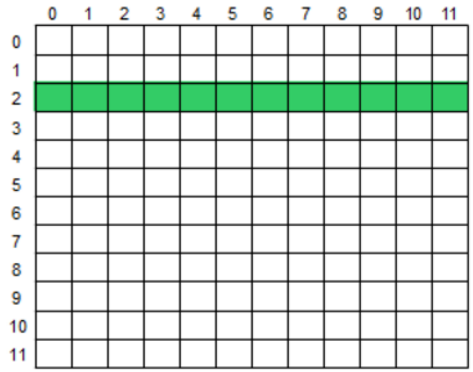
URI Online Judge | 1181

# Linha na Matriz

Por Neilor Tonin, URI Brasil

**Timelimit: 1**

Neste problema você deve ler um número, indicando uma linha da matriz na qual uma operação deve ser realizada, um caractere maiúsculo, indicando a operação que será realizada, e todos os elementos de uma matriz **M**[12][12]. Em seguida, calcule e mostre a soma ou a média dos elementos que estão na área verde da matriz, conforme for o caso. A imagem abaixo ilustra o caso da entrada do valor 2 para a linha da matriz, demonstrando os elementos que deverão ser considerados na operação.




## Entrada

A primeira linha de entrada contem um número **L** (0 ≤ **L** ≤ 11) indicando a linha que será considerada para operação. A segunda linha de entrada contém um único caractere Maiúsculo **T** ('S' ou 'M'), indicando a operação (Soma ou Média) que deverá ser realizada com os elementos da matriz. Seguem os 144 valores de ponto flutuante que compõem a matriz, sendo que a mesma é preenchida linha por linha, da linha 0 até a linha 11, sempre da esquerda para a direita.

## Saída

Imprima o resultado solicitado (a soma ou média), com 1 casa após o ponto decimal.

| Exemplo de Entrada | Exemplo de Saída |
|---|---|
| 2<br>S<br>0.0<br>-3.5<br>2.5<br>4.1<br>... | 12.6 |

1181.c

```c
#include <stdio.h>

int main() {

    int linha, i, j;
    double m[12][12], soma = 0.0;
    char operacao;
    scanf("%d %c", &linha, &operacao);
    for (i = 0; i < 12; i++){
        for (j = 0; j < 12; j++){
            scanf("%lf", &m[i][j]);
            if (linha == i)
                soma += m[i][j];
        }
    }

    if (operacao == 'S')
        printf("%.1lf\n", soma);
    else if (operacao == 'M')
        printf("%.1lf\n", soma/12.0);

    return 0;
}
```

Exemplo de solução para o **Problema 1181** (Linha na Matriz).

# Atividade em aula

# Questão 1 - a

```
int v1[] = {10, 2, 3, 100};
int v2[] = {7, 10, 7, 7, 80};
```

```
int funcao1( int a[], int n ) {
    int i;
    int s = 0;

    for (i=0; i<n; i++) {
        s += a[i];
    }

    return a[--i];
}
```

Devolve o último elemento do vetor.

```
printf("%d\n", funcao1(v1, 4));
100


printf("%d\n", funcao1(v2, 5));
80
```

# Questão 1 - b

```
int v1[] = {10, 2, 3, 100};
int v2[] = {7, 10, 7, 7, 80};
```

```
int funcao2( int a[], int n ) {
    int i;
    int s = 0;

    for (i=0; i<n; i++) {
        s += a[i]-a[i];
    }

    return s;
}
```

Devolve 0.

```
printf("%d\n", funcao2(v1, 4));
0

printf("%d\n", funcao2(v2, 5));
0
```

# Questão 1 - c

```
int v1[] = {10, 2, 3, 100};
int v2[] = {7, 10, 7, 7, 80};
```

```
int funcao3( int a[], int n ) {
    int i, j, cont = 0;

    for (i=0; i<n-1; i++) {
        for (j=i+1; j<n; j++) {
            if (a[i]==a[j]) {
                cont++;
            }
        }

    }

    return cont;
}
```

Devolve a quantidade de elementos iguais no vetor.

```
printf("%d\n", funcao3(v1, 4));
0


printf("%d\n", funcao3(v2, 5));
3
```

# Questão 2

```
int M[3][4] = { {-1, -1, -1, -1},  {0, 0, 0, 10},  {10, 10, 10, 10} } ;
```

```
int funcao4( int n, int m, int M[n][m] ) {
    int i, j, cont=0;

    for (i=0; i<n; i++) {
        for (j=0; j<m; j++) {
            if (M[i][j]>0)
                cont++;
            if (M[i][j]<0)
                cont--;
        }
    }

    if (cont>0)
        return 1;
    else
        return 0;
}
```

Verifica se a quantidade de elementos negativos é igual à quantidade de números positivos.

Se for verdadeiro, devolve 1, caso contrário, devolve 0.

```
printf("%d\n", funcao4(3, 4, M));
1
```

# Questão 3

**Indique se são verdadeiras ou faltas as seguintes afirmações**
(resposta correta +0,5, incorreta -0,5)

(a) [ ] Uma função em C pode devolver simultaneamente mais do que um valor.

(b) [ ] Uma função em C tem que devolver sempre um inteiro.

(c) [ ] Os parâmetros das funções podem ser do tipo `void`.

(d) [ ] A instrução `return`, termina a execução de uma função apenas se for a última instrução da função em que se encontra

(e) [ ] A instrução `return`, quando executada dentro da função `main`, termina o programa.

(f) [ ] O nome de uma função é opcional.

# Questão 3

**Indique se são verdadeiras ou faltas as seguintes afirmações**
(resposta correta +0,5, incorreta -0,5)

(a) [ **F** ] Uma função em C pode devolver simultaneamente mais do que um valor.

*Em C, uma função pode devolver apenas 1 valor (tipo básico ou ponteiro)*

(b) [ **F** ] Uma função em C tem que devolver sempre um inteiro.

*Pode devolver qualquer tipo de dado básico ou ponteiro*

(c) [ **F** ] Os parâmetros das funções podem ser do tipo `void`.

*Não. Cada parâmetro deve ter um tipo de dado associado*

(d) [ **F** ] A instrução `return`, termina a execução de uma função apenas se for a última instrução da função em que se encontra

(e) [ **V** ] A instrução `return`, quando executada dentro da função `main`, termina o programa.

(f) [ **F** ] O nome de uma função é opcional.