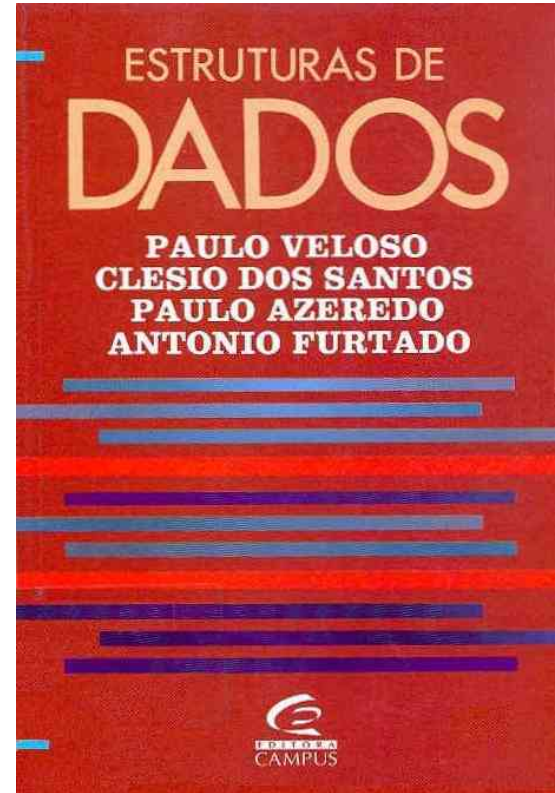


Representação de Dados

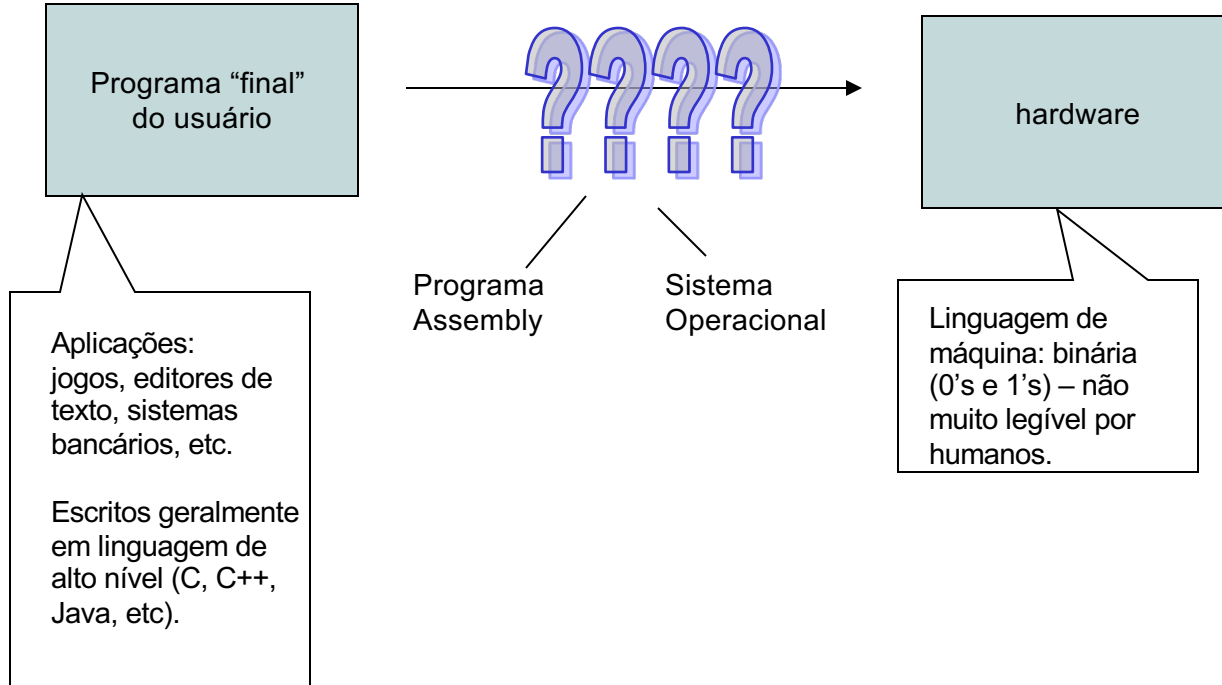
Estrutura de Dados

Prof. Anselmo C. de Paiva

Departamento de Informática – Núcleo de Computação Aplicada NCA-UFMA



Abstrações em Sistema de Computação



Processo geral de um executável

- ▶ Hello world

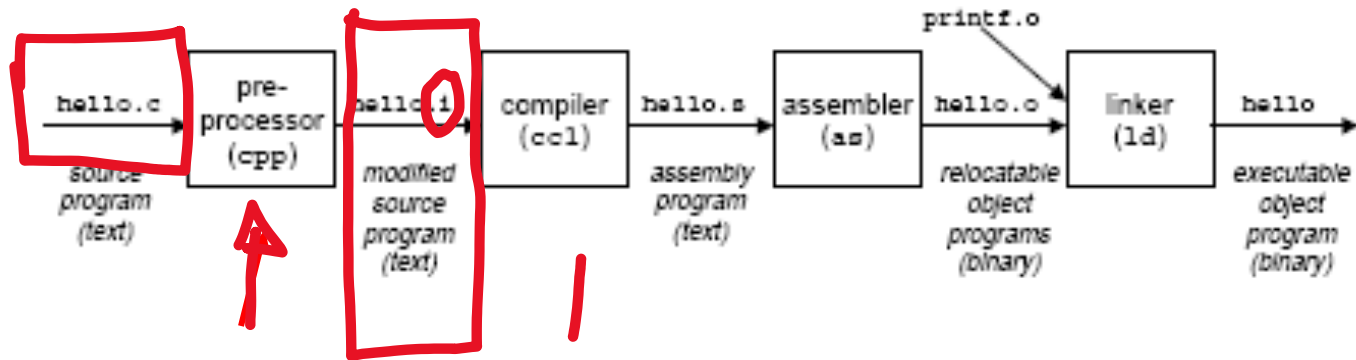
```
1 #include <stdio.h>
2
3 int main()
4 {
5     printf("hello, world\n");
6 }
```

code/intro/hello.c

code/intro/hello.c

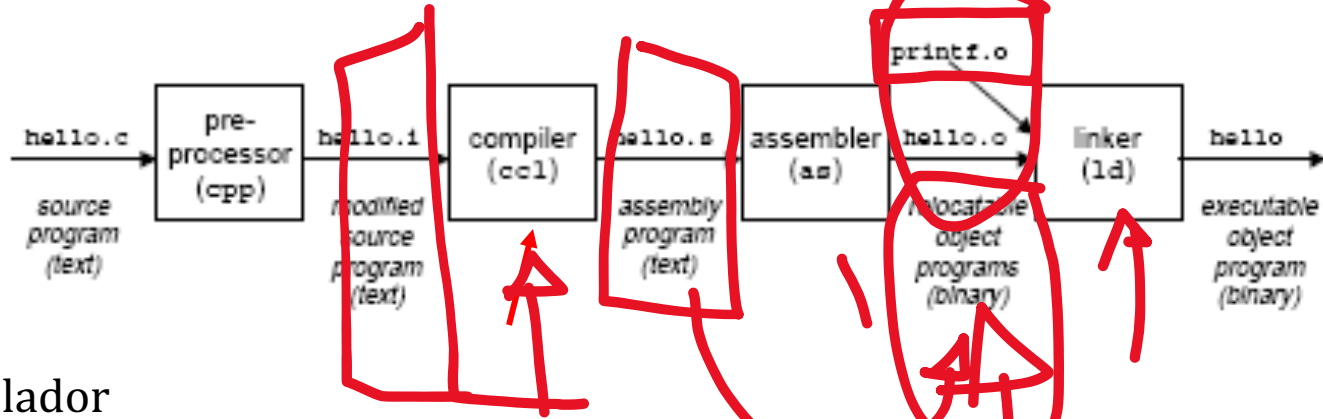
Processo geral de um executável

```
unix> gcc -o hello hello.c
```



- ▶ Modifica programa C de acordo com diretivas #
- ▶ Ex.:
 - ▶ `#include <stdio.h>`
 - ▶ ler o arquivo `stdio.h` e inseri-lo no programa fonte
- ▶ Resultado é um programa expandido em C (.i)

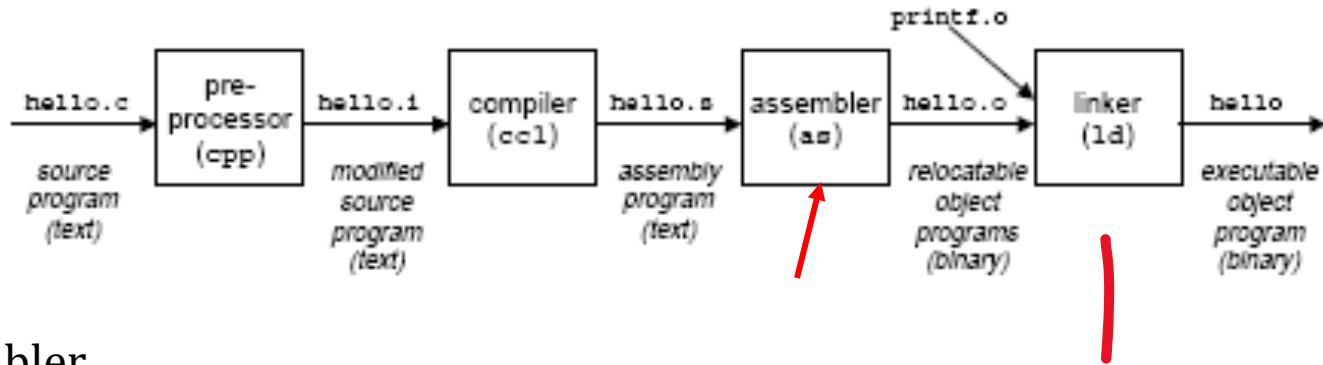
Processo geral de um executável



► Compilador

- traduz o programa .i em programa assembly.
 - formato de saída comum
- programas em várias linguagens são traduzidos
 - para a mesma linguagem Assembly.

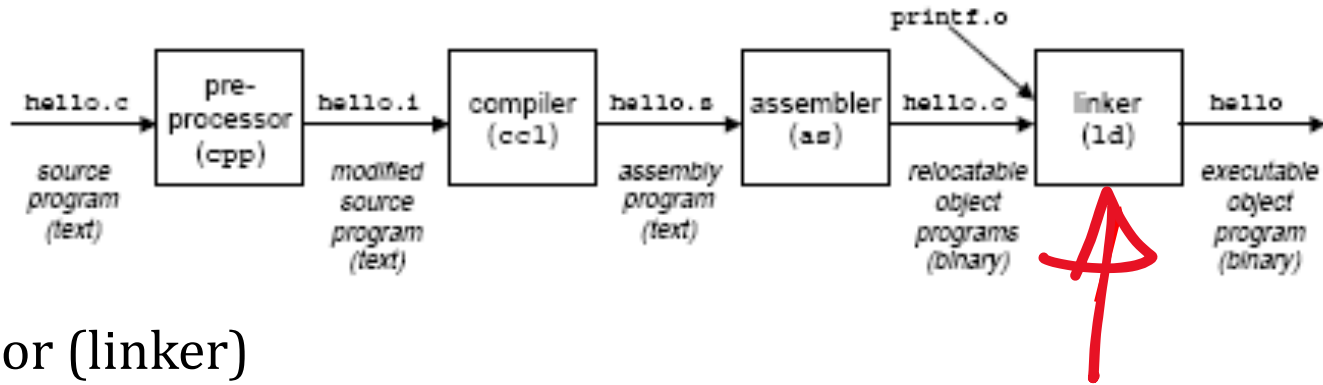
Processo geral de um executável



► Assembler

- transforma o programa assembly em um programa binário em linguagem de máquina
- programa objeto – extensão .o

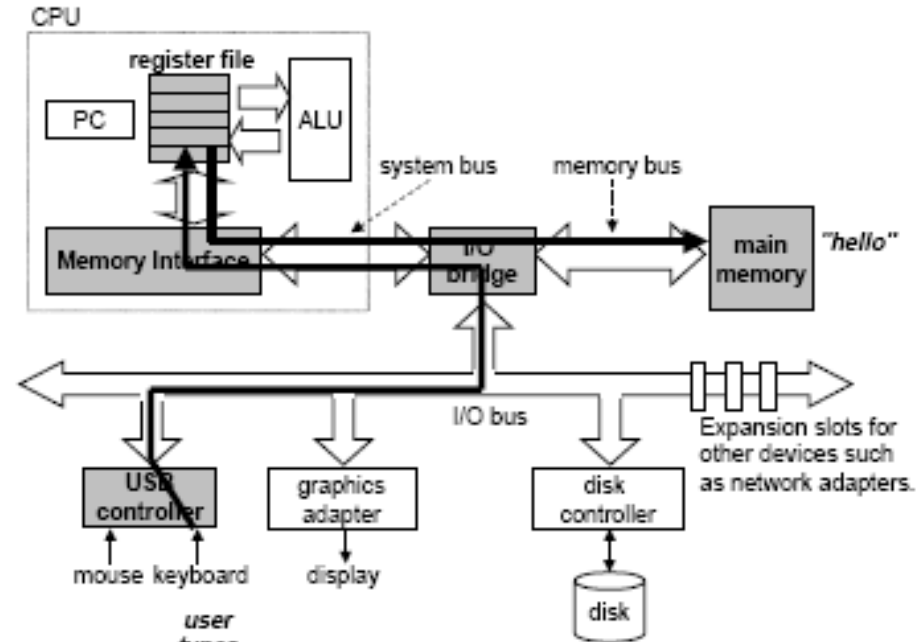
Processo geral de um executável



- ▶ Ligador (linker)
 - ▶ gera programa executável a partir do `.o`
 - ▶ Pode haver funções (por ex., `printf`)
 - ▶ que não estão definidas no programa
 - ▶ Estão em outro `.o` (no caso, `printf.o`).
 - ▶ faz a junção dos programas objetos necessários para gerar o executável.

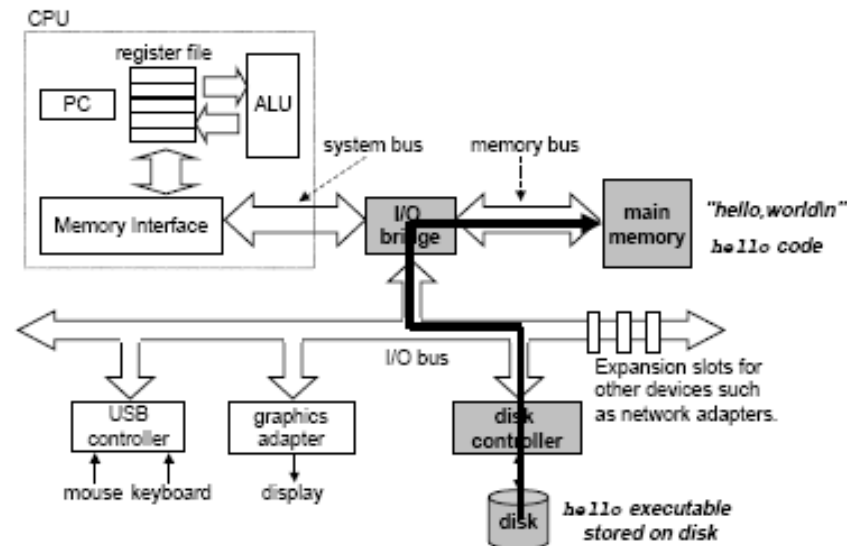
Execução do programa

- ▶ Ao digitar “hello”,
 - ▶ caracteres são enviados para memória principal.
- ▶ Memória
 - ▶ Logicamente: array de bytes (conjuntos de 8 bits)
 - ▶ cada byte tem seu endereço (o índice do array)
 - endereços começam de 0.



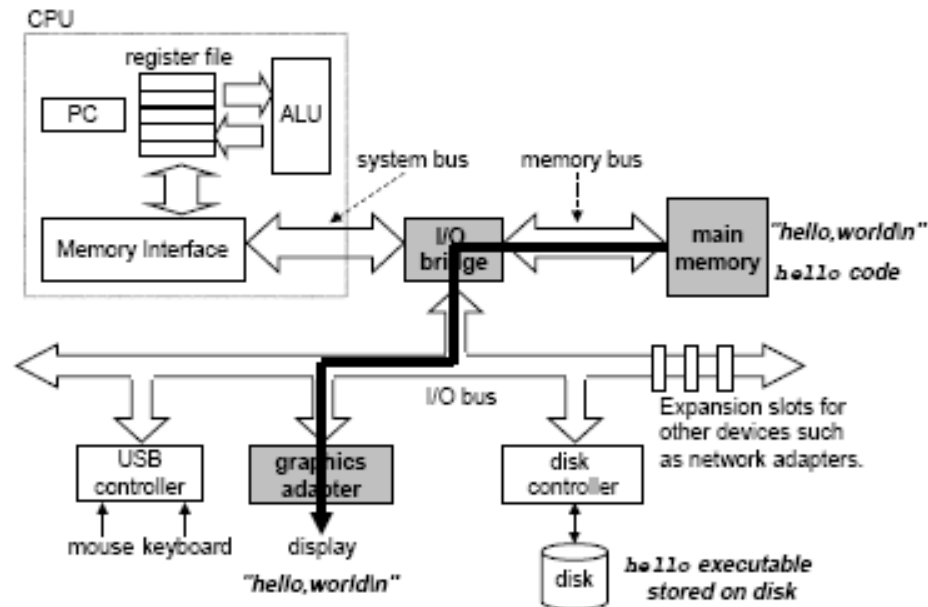
Execução do programa

- ▶ Ao clicar “Enter”,
 - ▶ Copia o código e dados do programa executável `hello.exe` do disco para a memória principal



Execução do programa

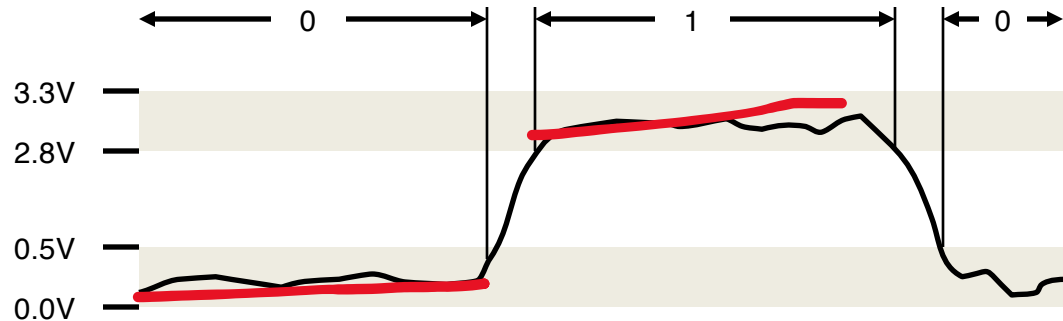
- ▶ Computador aponta para o endereço de memória onde o programa foi escrito
- ▶ Processador executa instruções em linguagem de máquina da “main” do programa



Representação de Dados

Representação binária

- ▶ Exemplo: 1521310 = 11101101101101
- ▶ Vantagens:
 - ▶ Implementação eletrônica
 - ▶ Possibilidade de armazenar elementos com dois estados
 - ▶ Transmissão eletrônica confiável (robustez a ruídos)
 - ▶ Implementação eficiente de operações aritméticas



Byte = 8 bits

► Faixa de valores em diferentes representações:

- Binário: 000000002 - 111111112
- Decimal: 010 - 25510
- Hexadecimal 0016 - FF16
 - Representação na base 16
 - Dígitos são '0' - '9' e 'A' - 'F'
 - Escreva FA1D37B16 em C
 - 0xFA1D37B ou 0xfa1d37b

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Conversão entre bases

- ▶ De base b para base 10
- ▶ De base 10 para base b
 - ▶ Divisões sucessivas por b ; a cada iteração i o resto contém o numeral a_i

```
void num2string(int num, int base, char *b, int size) {
    int div, resto;
    int i = size - 2;
    div = num;
    while (div && i >= 0) {
        resto = div % base;
        if (resto < 10) b[i] = resto + '0';
        else          b[i] = (resto - 10) + 'A';
        div /= base;
        i--;
    }
}

...
char buffer[11]; buffer[10]=0;
num2string(num, base, buffer, 11);
```

A palavra (word)

- ▶ Cada máquina tem seu tamanho de palavra:
 - ▶ = número de bits ocupados por um valor inteiro
 - ▶ = número de bits de endereços
- ▶ Máquinas atuais tem palavra de:
 - ▶ 32 bits (4 bytes)
 - ▶ 64 bits (8 bytes)
- ▶ Tipos de dados ocupam uma fração ou múltiplo do tamanho da palavra
 - ▶ sempre um número inteiro de bytes;

Tamanhos de Tipos em C

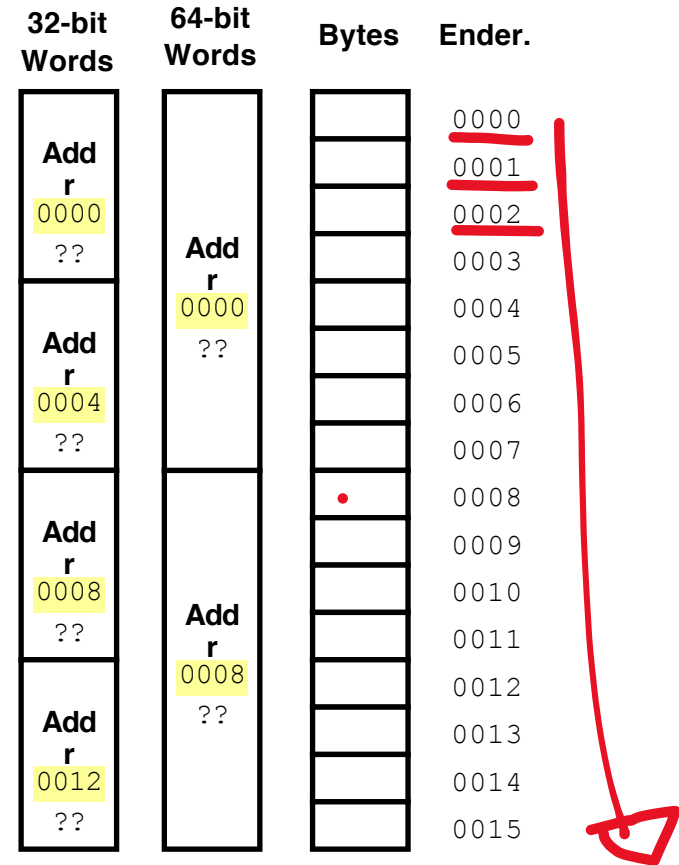
...em bytes

	Alpha(64 bit)	32-bit	IA32
▶ Tipo C			
▶ int	4	4	4
▶ long int	8	4	4
▶ char	1	1	1
▶ short	2	2	2
▶ float	4	4	4
▶ double	8	8	8
▶ long double	8	8	10/12
▶ ponteiro (tipo *)	8	4	4

Obs: `sizeof(T)` retorna o número de bytes usado por tipo T

Memória orientada a palavras

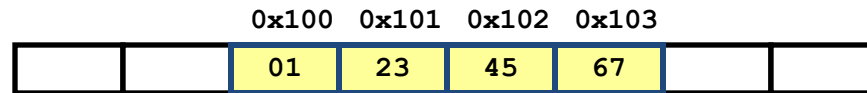
- ▶ Memória
 - ▶ vetor de bytes
 - ▶ cada byte possui um endereço
 - ▶ acesso ocorre palavra a palavra
- ▶ Endereço
 - ▶ primeiro byte da palavra
- ▶ Endereços de palavras subsequentes
 - ▶ diferem de 4 em máquina de 32 bits



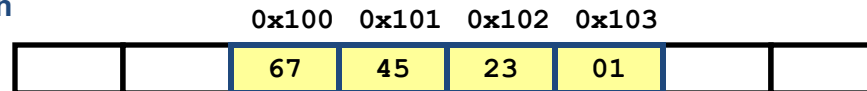
Ordenação de bytes

- ▶ Como organizar os bytes de uma palavra (4 bytes)?
 - ▶ Big Endian (computadores Sun, Mac)
 - ▶ byte menos significativo com maior endereço
 - ▶ Little Endian (computadores Alpha PCs)
 - ▶ Byte menos significativo no menor endereço
 - ▶ Exemplo
 - ▶ variável y tem valor 0x01234567 (hexa)
 - ▶ Endereço &y é 0x100

Big Endian



Little Endian



BE

0	000 0
1	000 1
2	0010
3	0011

00	00
01	00
<hr/>	
10	01

Ordenação de Bytes

- ▶ Transparente para o programador C
- ▶ Relevante quando analisamos o código de máquina
- ▶ Exemplo (little endian):

01 05 64 94 04 08

add \$0x8049464 %eax

- ▶ Importante também para a transmissão de dados entre máquinas big e little endian



Verificar se a máquina é big ou little endian

```
#include <stdio.h>
typedef unsigned char *byte_ptr;

void mostra (byte_ptr inicio, int tam) {
    int i;
    for (i=0;i<tam;i++)
        printf("%.2x", inicio[i]);
    printf("\n");
}

void mostra_int (int num) {
    mostra((byte_ptr) &num, sizeof(int));
}
```

Caracteres

- ▶ Usa-se uma tabela para mapear inteiros (não negativos) em símbolos

- ▶ Tabelas mais comuns:

- ▶ ASCII – codificação em 8 bits

'a'	97	0x61
'z'	122	0x7A
'A'	65	0x41

- ▶ UNICODE – codificação em 16 bits

- ▶ Em C, tipo `char` define apenas 1 byte
- ▶ Pode-se manipular um `char` como um `int`
 - ▶ Exemplo: `if (c > '0') val = c - '0';`

Vetores e Matrizes

▶ Vetor

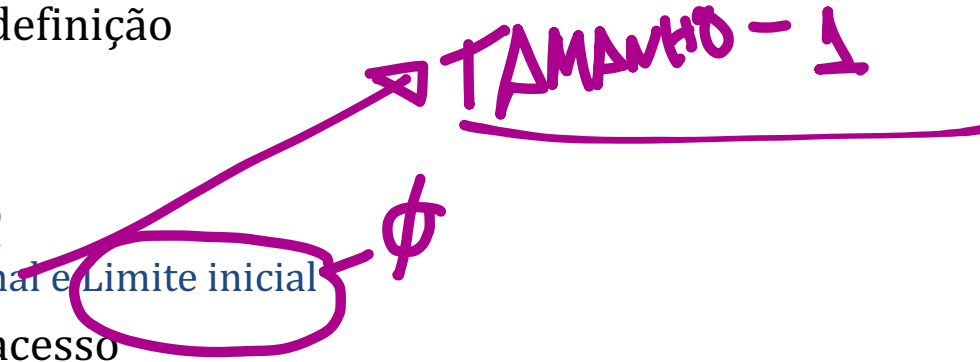
- ▶ Conjunto finito e limitado de elementos homogêneos.

▶ Forma de definição

- ▶ Nome
- ▶ Tipo
- ▶ Tamanho
- ▶ Limite final e Limite inicial

▶ Forma de acesso

- ▶ Armazenamento e recuperação de qualquer posição dentro do vetor em qualquer tempo.



Vetores e Matrizes

▶ Vetor – unidimensional

▶ Matriz – bidimensional

▶ Volume - tridimensional

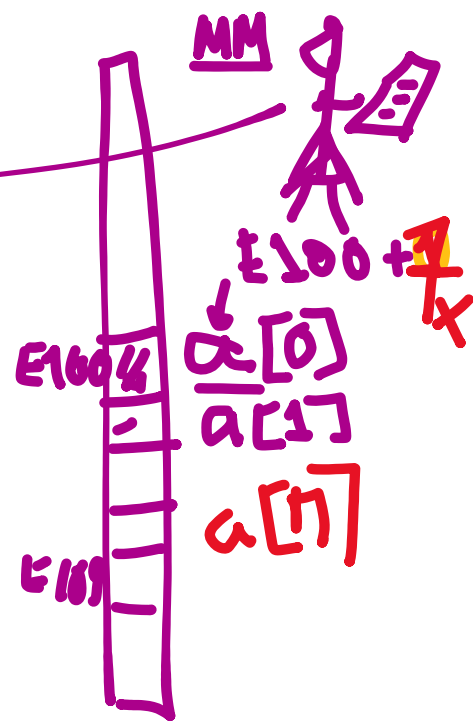
Vetores e Matrizes

Implementação de vetores

Em C: `int a[100];`

- ▶ Reserva n posições sucessivas de memória
- ▶ Cada posição contém um único elemento
- ▶ O endereço da primeira posição é a base a
- ▶ Intervalo de 0 a $(n-1)$
- ▶ Forma geral:

`a[pos]` → conteúdo $\{ \text{base}(a) + \text{pos} * \text{size} - t \}$



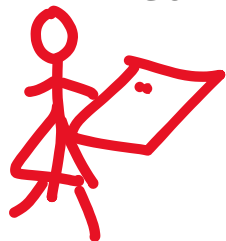
Todos os elementos possuem o mesmo tamanho, facilitando as operações básicas e a geração de código compacto e veloz.

Matrizes

- ▶ Matrizes
 - ▶ Nome usual para definir vetores n-dimensionais.
- ▶ Implementação de matrizes
 - ▶ $a[i][j] := @base(a) + i * numcol + j$
- ▶ Exemplo:
 - ▶ Matriz A [3] [5]

Matrizes

Como



$E^{2 \times 3}$

$E^{3 \times 7}$

$E^{4 \times 2}$

00	01	02	03	04	10	11	12	13	14	20	21	22	23	24
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

Linha 0

Linha 1

Linha 2

$m[2][3]$



$m^{E^{2 \times 3}}$

$m[0][0]$

0	→
1	→
2	→

linhas

$(0,0)$	$(0,1)$	$(0,2)$...
---------	---------	---------	-----

09

$(1,0)$	$(1,1)$	$(1,2)$...
---------	---------	---------	-----

11

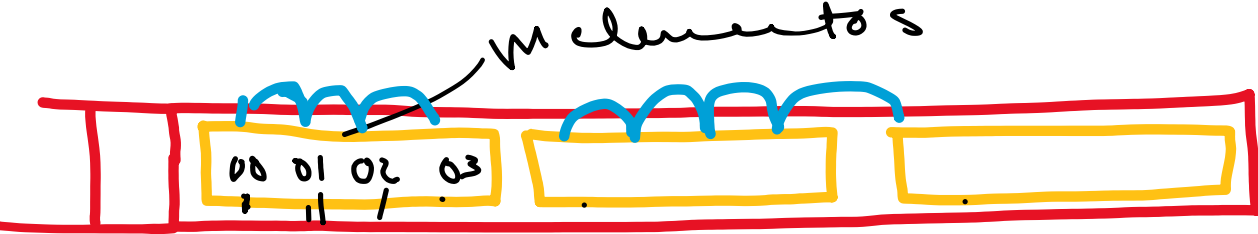
$(2,0)$	$(2,1)$	$(2,2)$...
---------	---------	---------	-----

29

$m[0][1]$

$m[0][2]$

$m[2][0]$



$$A_{n \times m}$$

$$3 \times 4$$

$$\rightarrow \Delta LINHA i \rightarrow POS 1^o - da i + j$$

10

1º ELEMENTO DA LINHA 0

$$E0 + 0 \times m$$

$$1 - E0 + m \times 1$$

$$2 - (E0 + m) + m = E0 + 2 \times m$$

$$LINHA i - E0 + i \times m$$

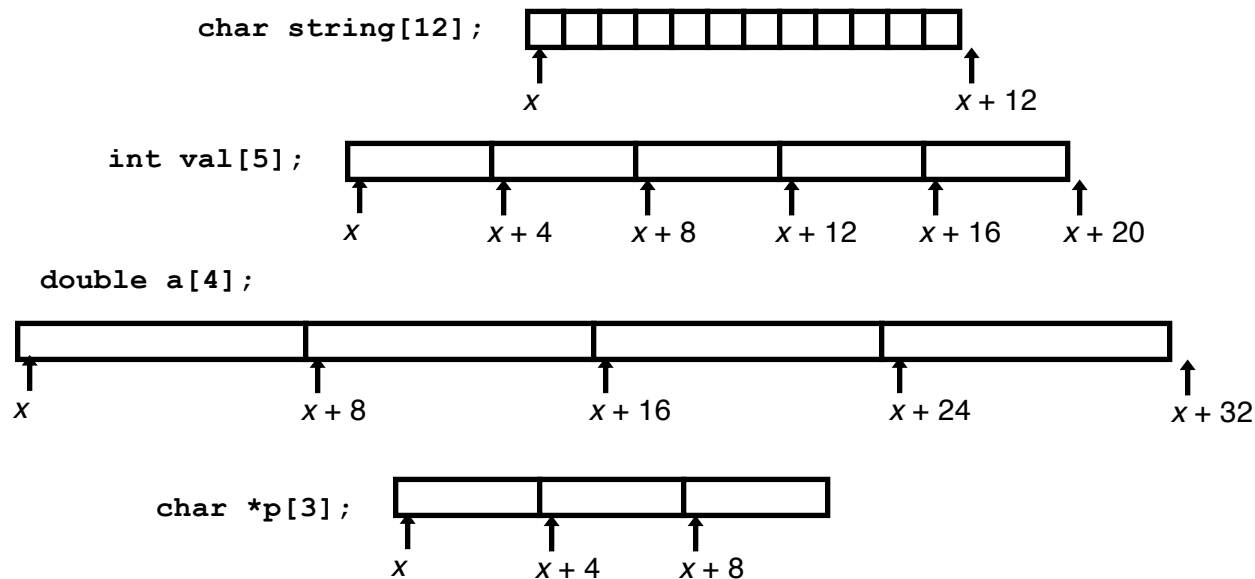
$$\text{Elemento } (i, j) = E0 + i \times m + j$$

Representação de tipos estruturados: Vetores (arrays)

- ▶ Alocação contígua na memória
- ▶ Primeiro elemento (a[0])
 - ▶ menor endereço de memória
- ▶ Tipo a[tam] ocupa
 - ▶ sizeof(Tipo) * tam bytes
- ▶ Nome do array equivale a um ponteiro
 - ▶ valor deste ponteiro não pode ser alterado)
- ▶ Exemplo:
 - ▶ int a[tam],
 - ▶ a é ponteiro constante tipo int *, e seu valor é &a[0]
- ▶ Nomes de arrays passados como parâmetros são endereços (do array)

Alocação para Arrays

- ▶ Seja T A[L];
 - ▶ Array de tipo T e comprimento L
 - ▶ Alocação contígua de $L * \text{sizeof}(T)$ bytes
- ▶ Exemplos:



Vetor

void f() {
 int v[10];
}

▶ Declaração estática

- ▶ int v[10];
- ▶ tamanho definido antes da execução do programa
- ▶ v armazena o endereço de memória ocupado pelo primeiro elemento do vetor
 - ▶ *v e v[0]
 - ▶ *(v+1) e v[1]
- ▶ Escopo de declaração local
- ▶ Declarado dentro da função
 - ▶ Não pode ser acessado fora da função
 - ▶ Memória do vetor é liberada ao final da função

▶ Declaração Dinâmica

- ▶ int *v;
- ▶ v = (int*) malloc (n*sizeof(int));
- ▶ Tamanho pode ser definido em tempo de execução do programa
- ▶ Variável ponteiro aponta para a primeira posição do vetor
- ▶ Memória do vetor permanece alocada até que seja liberada explicitamente por free()

PONTEIROS

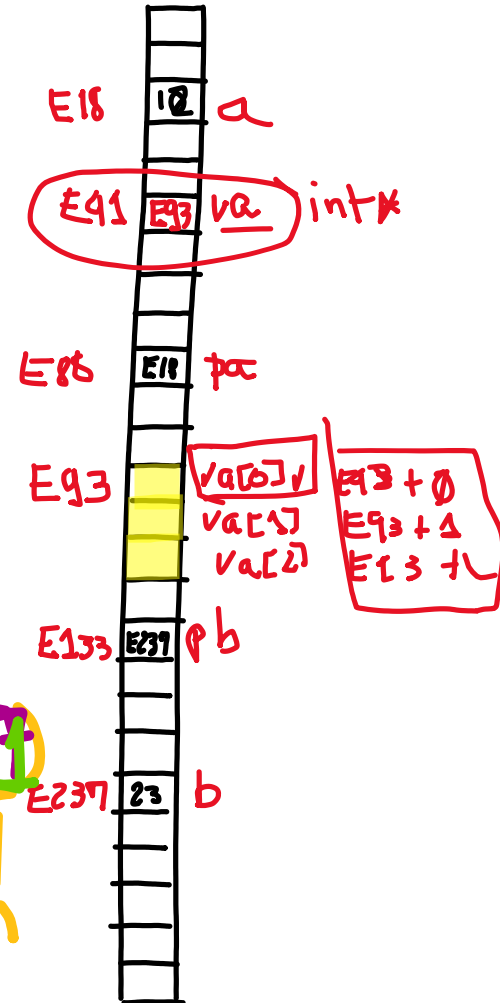
```
int a;
int b;
int *pa;
int *pb;
int va[3];
float *fuik;
```

```
a = 10;
b = 23;
pa = &a;
pb = &b;
```

```
*pa = 12;
pa = 12;
```

tipo	nome	Valor	Valor do valor
int	a	12	
int	b	23	
Int *	pa	E18	12
Int *	pb	E237	23

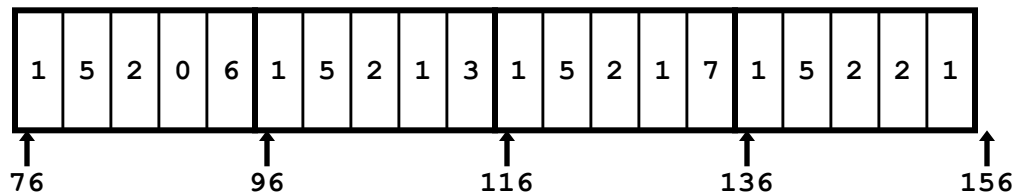
fuik = (float *) malloc(sizeof(float) * 3)
 VAR TIPO TIPO h
 float void*



Matrizes

```
#define PCOUNT 4  
int a[5][PCOUNT] =  
    {{1, 5, 2, 0, 6},  
     {1, 5, 2, 1, 3 },  
     {1, 5, 2, 1, 7 },  
     {1, 5, 2, 2, 1 }};
```

`int a[5][4];`



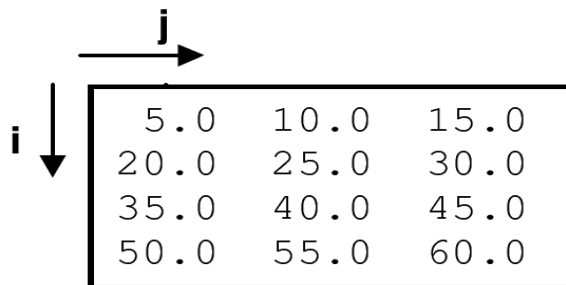
- ▶ Declaração “`int a[4][5]`”
 - ▶ Variável `a` denota array de 4 elementos
 - ▶ Alocados continuamente
 - ▶ Cada elemento é um vetor de 5 `int`'s
 - ▶ Alocados continuamente
 - ▶ Ordenação é por linha da matriz

Matrizes

▶ Conjuntos bidimensionais declarados estaticamente

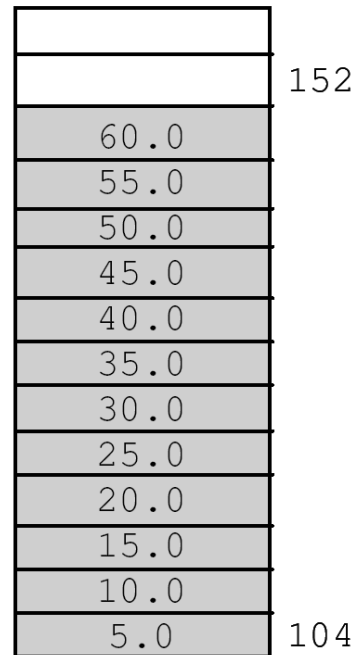
- ▶ float mat[4][3];
- ▶ acesso de elemento: mat[2][0]
- ▶ float mat[4][3] = {{5.0,10.0,15.0},
{20.0,25.0,30.0},
{35.0,40.0,45.0},
{50.0,55.0,60.0}};
- ▶ Iteração típica sobre todos os elementos:

```
for (i=0;i<lin;i++){  
    for(j=0;j<col;j++){
```



A diagram showing a 2D array with 4 rows and 3 columns. The row index is labeled 'i' with a downward arrow, and the column index is labeled 'j' with a rightward arrow. The array contains the following values:

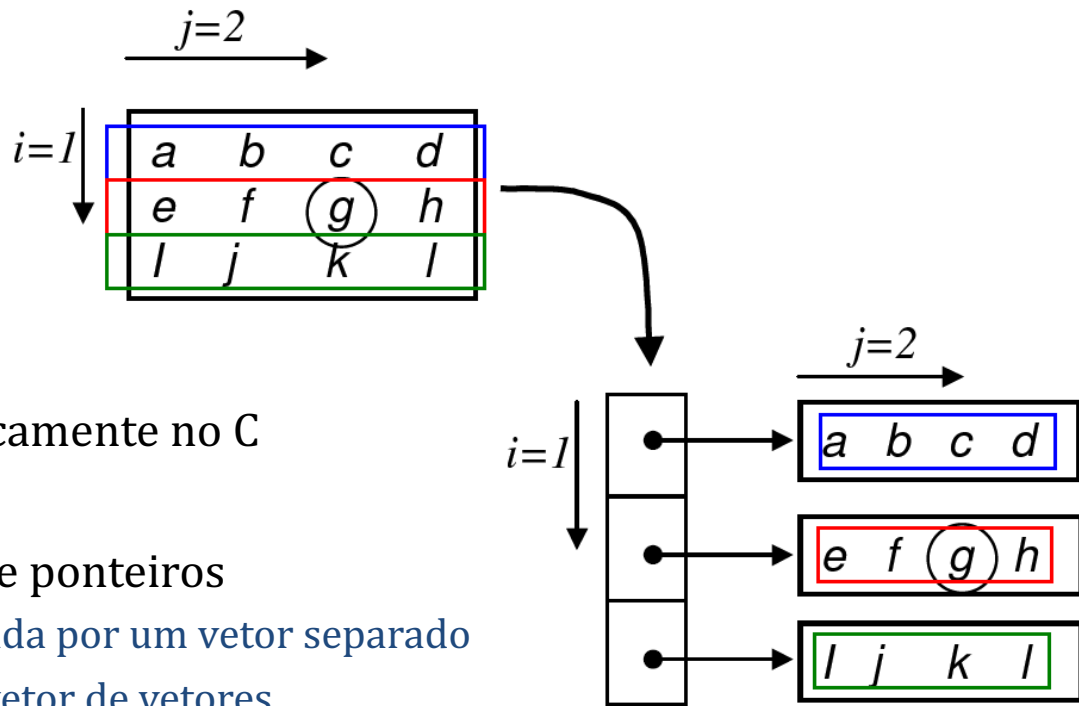
5.0	10.0	15.0
20.0	25.0	30.0
35.0	40.0	45.0
50.0	55.0	60.0



A diagram showing a 1D array representing the 2D array in row-major order. The array has 12 elements, with the first two being empty and the last one being 5.0. The address 152 is indicated at the top right, and 104 is indicated at the bottom right.

60.0
55.0
50.0
45.0
40.0
35.0
30.0
25.0
20.0
15.0
10.0
5.0

Matrizes Dinâmicas



- ▶ não podem ser alocados dinamicamente no C
- ▶ necessárias abstrações
- ▶ Matriz representada por vetor de ponteiros
 - ▶ Cada linha da matriz é representada por um vetor separado
 - ▶ A matriz é representada por um vetor de vetores
 - ▶ Vetor de ponteiros
 - Elemento do vetor armazena o endereço de memória do primeiro elemento de cada linha

PONTEIROS

— ·

int **mat;

—

mat[0] int *

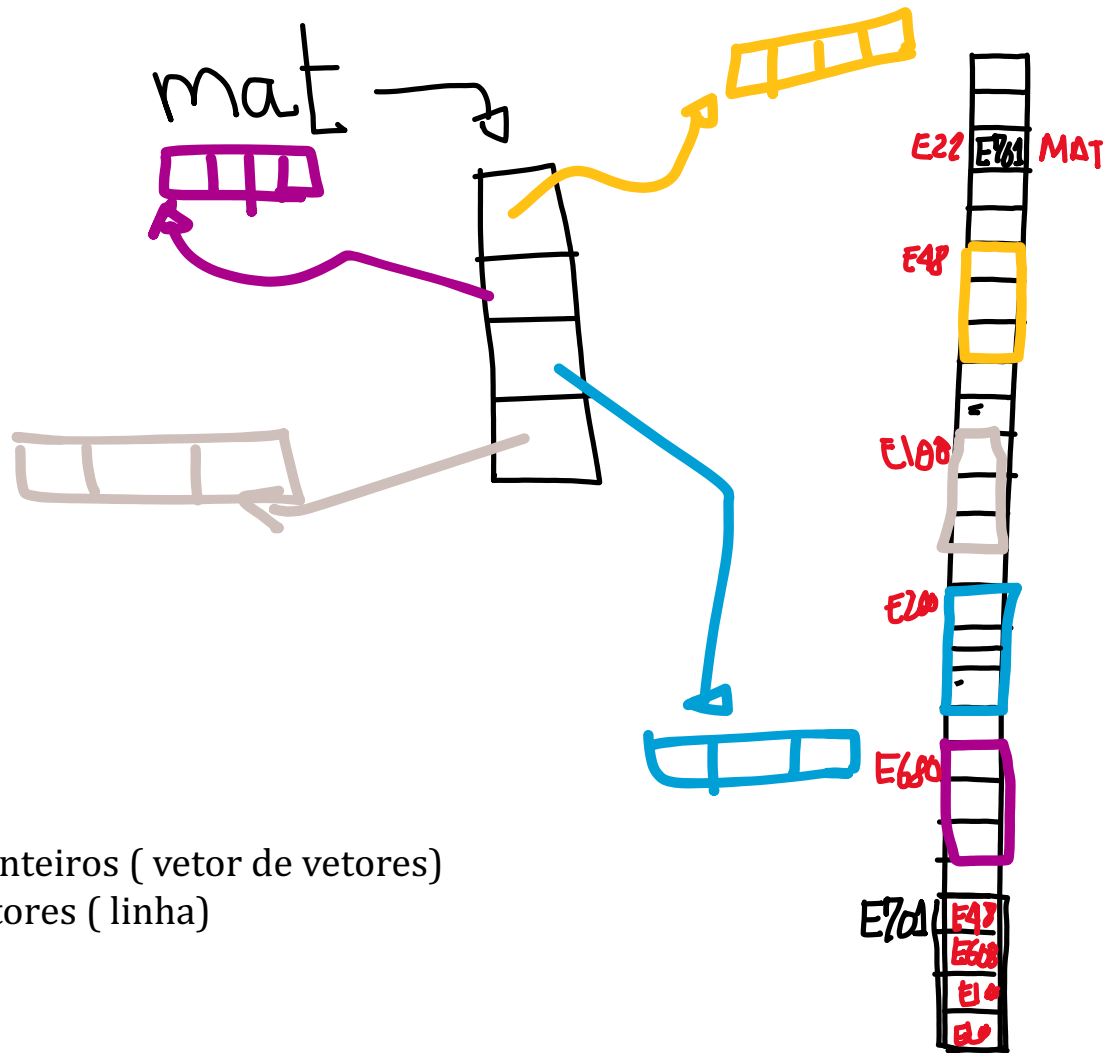
Mat tipo int **

Mat [2][3]. Int

Aloacao em dois passo

Primeiro ; aoca um vetor de ponteiros (vetor de vetores)

Segundo aloca cada um dos vetores (linha)



Matriz representada por vetor de ponteiros

▶ Alocar

```
▶ int i;  
▶ float **mat; /*vetor de ponteiros*/  
▶ mat = (float **) malloc (n*sizeof(float *));  
▶ if ( mat != NULL ) {  
▶     for (i=0; i<n; i++) {  
▶         mat[i] = (float *) malloc (m*sizeof(float));  
▶     }  
▶ }
```

▶ Liberar o espaço de memória alocado

```
▶ for (i=0; i<m; i++) {  
▶     free (mat[i]);  
▶ }  
▶ free (mat);
```

- ▶ A lista de exercício deve ser respondida manuscrita em papel e escaneada (fotografada), gerando um único arquivo pdf.

- ▶ 1. Faça um algoritmo que recebe um vetor de n números reais e uma matriz $m \times p$ de números reais, e imprime em quais posições da matriz (linha e coluna) o valor é igual a um dos valores do vetor (informe também a posição do vetor).
 - ▶ a. Considere a matriz alocada como vetor de vetores (matriz 2D)
`int CmpVectorMatrix (int n, int m, int p, float *vet, float **mat)`
 - ▶ b. Considere a matriz alocada em um vetor unidimensional
`int *CmpVectorMatrix (int n, int m, int p, float *vet, float *mat)`

- ▶ 2. Faça um algoritmo que recebe os inteiros n , m , p e q , e duas matrizes ma e mb e retorna o produto entre as duas matrizes.
 - ▶ a. Considere as matrizes alocadas como vetor de vetores (matriz 2D)
`int **MultMatrix (int n, int m, int p, int q, float **ma, float **mb)`
 - ▶ b. Considere as matrizes alocadas em um vetor unidimensional
`int *MultMatrix (int n, int m, int p, int q, float *ma, float *mb)`

```

int CmpVectorMatrix ( int n, int m, int p, float *vet, float **mat) {
    int i, l, c, elm;
    if ( vet != NULL && mat != NULL) {
        if ( n > 0 && m > 0 && p > 0 ) {
            for (i=0; i<n; i++ ) {
                elm = vet[i];
                for (l = 0; l< m; l++) {
                    for ( c=0; c<p; c++) {
                        if (elm == mat[l][c]) {
                            printf ( “elemento %d na posição %d do vetor é igual a elemento na posição
                                [%d][%d] da matriz”, elm, i, l, c);
                        }
                    }
                }
            }
        }
    }
    return TRUE;
}

```

```

float **MultMatrix ( int n, int m, int p, int q, float **ma, float **mb) {
    int i, j, k;
    float **mc;
    if ( ma != NULL && mb != NULL ) {
        if ( n > 0 && m > 0 && p > 0 && q > 0 ) {
            if ( m == p ) {
                mc = (float *) malloc ( sizeof ( float *)*n);
                if ( mc != NULL ) {
                    for ( i=0; i<n; i++ ) {
                        mc[i] = ( float *) malloc (sizeof(float)*q);
                        if ( mc[i] == NULL ) {
                            for (j=i-1; j<=0; j--) {
                                free(mc[j]);
                            }
                            return NULL;
                        }
                    }
                }
            }
            for ( i=0; i<n; i++ ) {
                for (j=0; j<q; j++ ) {
                    mc [i][j] = 0.0;
                    for (k= 0; k<m; k++){
                        mc[i][j] += ma[i][k]*mb[k][j]
                    }
                }
            }
            return mc;
        }
    }
    return NULL;
}

```

Armazenamento de Matrizes em vetor

- ▶ Útil em aplicações de processamento de imagens e visualização volumétrica

- ▶ Matriz 2D

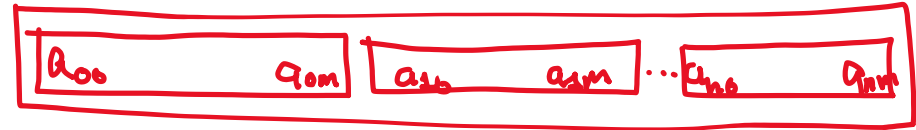
- ▶ $K = @base(a) + i * numcol + j$

- ▶ Matriz 3D

- ▶ $L = @base(a) + k * numcol * numlinha + i * numcol + j$

$$A = \begin{bmatrix} a_{00} & \dots & a_{0m} \\ \vdots & & \vdots \\ a_{n0} & \dots & a_{nm} \end{bmatrix}$$

✓



MATRIZ A
↳ vetor ✓
(i, j) MATRIZ
↳ k novo for
 $k = (k * m + j)$

1. elemento da linha 0 — 0

$$m=4$$

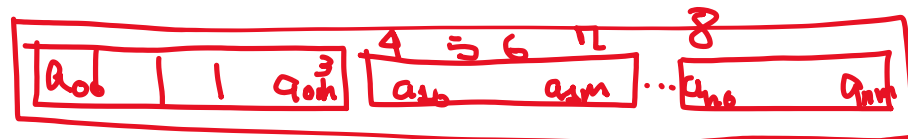
1. " " " 1 — m

" " " 2 — $m+m-2m$

i — $i \times m$

$$j=0 \quad 1 \quad 2$$

$$+0 \quad +1 \quad +2$$



$$A = \begin{bmatrix} a_{00} & \dots & a_{0m} \\ \vdots & & \vdots \\ a_{n0} & \dots & a_{nm} \end{bmatrix}$$

MATRIX A

vector v

(i, j) MATRIX n rows m columns

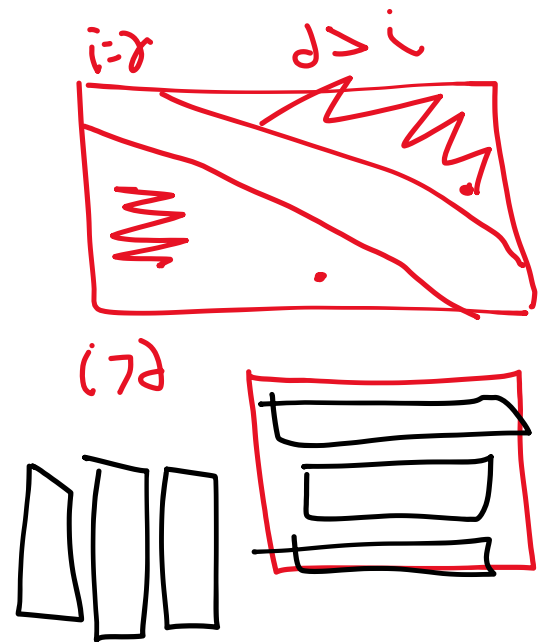
k row vector

$$k = (km + j)$$

```
float *AlocaLeMatrizEmVetor ( int n, int m )
{
    float *vm;
    if ( n> 0 && m >0 ) {
        vm = (float *) malloc (sizeof(float ) * n *m );
        if ( vm != NULL ) {
            for ( i=0; i<n; i++) {
                for (j=0; j<m; j++ ) {
                    printf("digite o elemento da linha %d coluna %d ", i, j);
                    // scanf ( "%f", &(matriza[i][j]));
                    k = i*m + j;
                    scanf ( "%f", &( vm[k]));
                }
            }
            return vm;
        }
    }
}
```

```
int *RetornaElementosdaLinha ( int *vm, int n, int m, int l )
{
    int *elms;
    if ( vm != NULL ){
        if (n>0 && m>0 ) {
            elms = (int *) malloc (sizeof( int) *m );
            if ( elms != NULL ) {
                for (j=0; j<m; j++ ) {
                    // elms[j] = matriz[l][j]
                    k = l*m + j;
                    elms[j] = vm [k]
                }
                return elms;
            }
        }
    }
    return NULL;
}
```

- ▶ Considere as matrizes ma e mb armazenadas em vetor e seus tamanhos $n \times m$ e $p \times q$ e faça as seguintes funções:
- ▶ A) retorna o resultado da multiplicação de ma por mb se for possível em um vetor
 - ▶ `int *MultMat (int * vma, int *vmb, int n, int m, int p, int q)`
- ▶ B) retorna a matriz transposta de ma
 - ▶ `int *TranspostadeMat (int * vma, int n, int m)`
- ▶ C) retorna a soma dos elementos da linha l da matriz ma
 - ▶ `int *SomaDaDiagonal (int * vma, int n, int m)`
- ▶ D) retorna TRUE se ma é simétrica e FALSE caso contrario
 - ▶ `Int ESimetrica (int * vma, int n, int m)`
- ▶ E) retorna os elementos do triângulo superior da matriz ma
 - ▶ `int *TRainguloSuperiodemat (int * vma, int n, int m)`
- ▶ F) retorna a soma dos elementos da diagonal principal ($i=j$)
 - ▶ `int *ElementosDiagonal (int * vma, int n, int m)`



Matrizes representadas por vetor simples

- ▶ Matriz é representada por vetor unidimensional
- ▶ Primeiras posições do vetor armazenam os elementos da primeira linha
- ▶ Exige disciplina para acessar os elementos

▶ `mat[i][j]`

▶ mapeado para `v[i*n + j]`

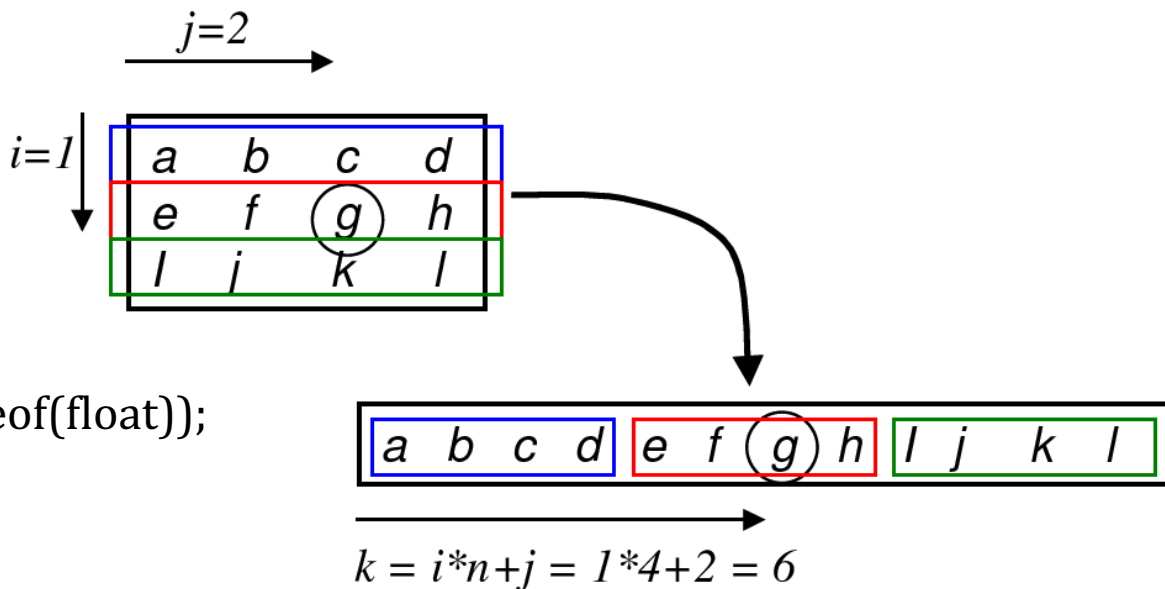
▶ `m` e `n`

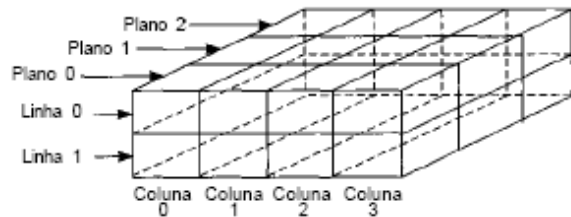
▶ dimensões da matriz

▶ Vetor com $m*n$ elementos

`float *mat;`

`mat = (float*) malloc(m*n*sizeof(float));`





(a)

Cabeçalho		0	2
		0	1
		0	3
Plano 0	Linha 0	$b[0]$	$[0]$ $[0]$
		$b[0]$	$[0]$ $[1]$
		$b[0]$	$[0]$ $[2]$
	Linha 1	$b[0]$	$[0]$ $[3]$
		$b[0]$	$[1]$ $[0]$
		$b[0]$	$[1]$ $[1]$
Plano 1	Linha 0	$b[0]$	$[1]$ $[2]$
		$b[0]$	$[1]$ $[3]$
		$b[1]$	$[0]$ $[0]$
	Linha 1	$b[1]$	$[0]$ $[1]$
		$b[1]$	$[0]$ $[2]$
		$b[1]$	$[0]$ $[3]$
Plano 2	Linha 0	$b[1]$	$[1]$ $[0]$
		$b[1]$	$[1]$ $[1]$
		$b[1]$	$[1]$ $[2]$
	Linha 1	$b[1]$	$[1]$ $[3]$
		$b[2]$	$[0]$ $[0]$
		$b[2]$	$[0]$ $[1]$
Plano 3	Linha 0	$b[2]$	$[0]$ $[2]$
		$b[2]$	$[0]$ $[3]$
		$b[2]$	$[1]$ $[0]$
	Linha 1	$b[2]$	$[1]$ $[1]$
		$b[2]$	$[1]$ $[2]$
		$b[2]$	$[1]$ $[3]$

← base (b)

(b)