



**BC1424**

**Algoritmos e Estruturas de Dados I**

**Aula 05:**  
**Vetores e Matrizes**

Prof. Jesús P. Mena-Chalco

[jesus.mena@ufabc.edu.br](mailto:jesus.mena@ufabc.edu.br)

1Q-2015

# Alocação estática de memória

5	
6	int V[100];
7	
8	
9	V[0] = 10;
10	V[1] = 20;
11	V[2] = 30;
12	

# Endereços

37FD00

01010111

37FD01

11000011

37FD02

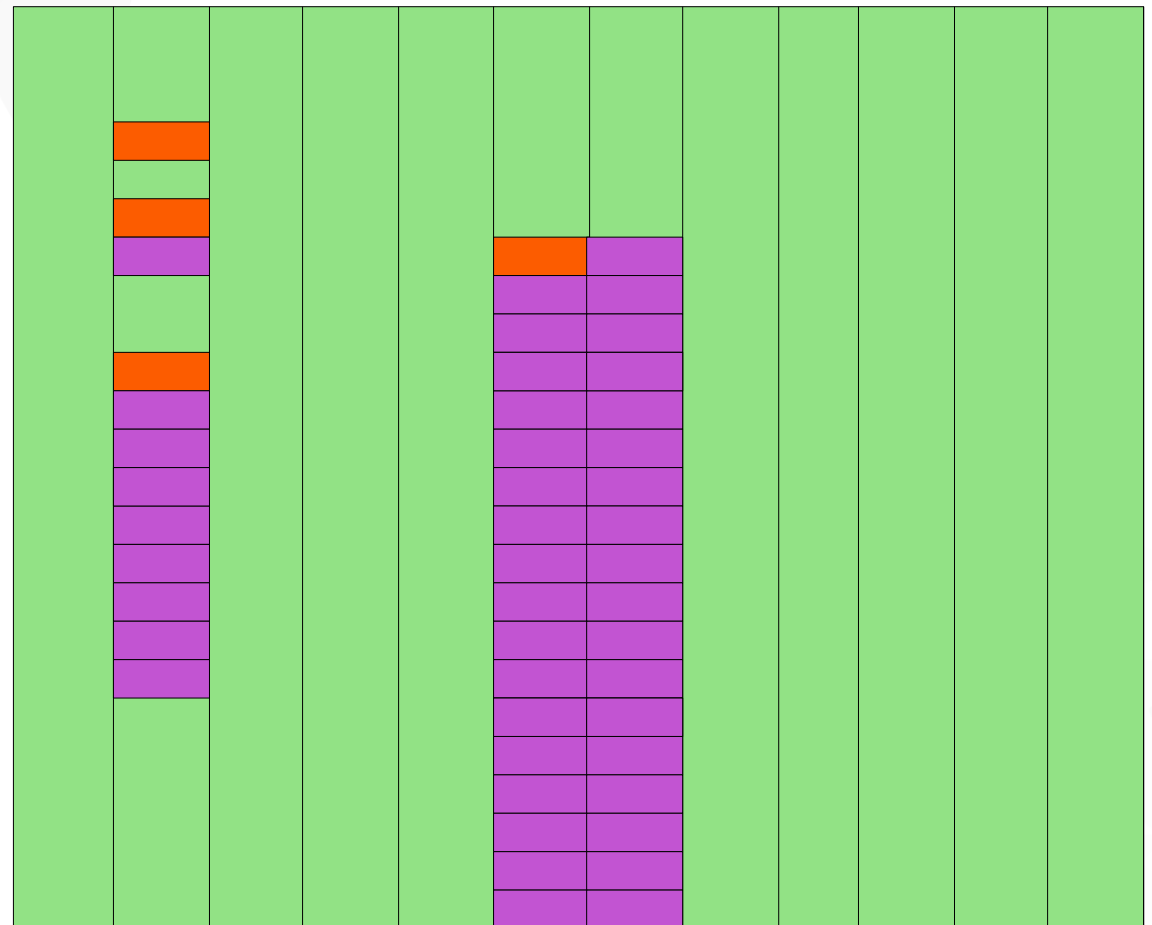
01100100

37FD03

11100010

...

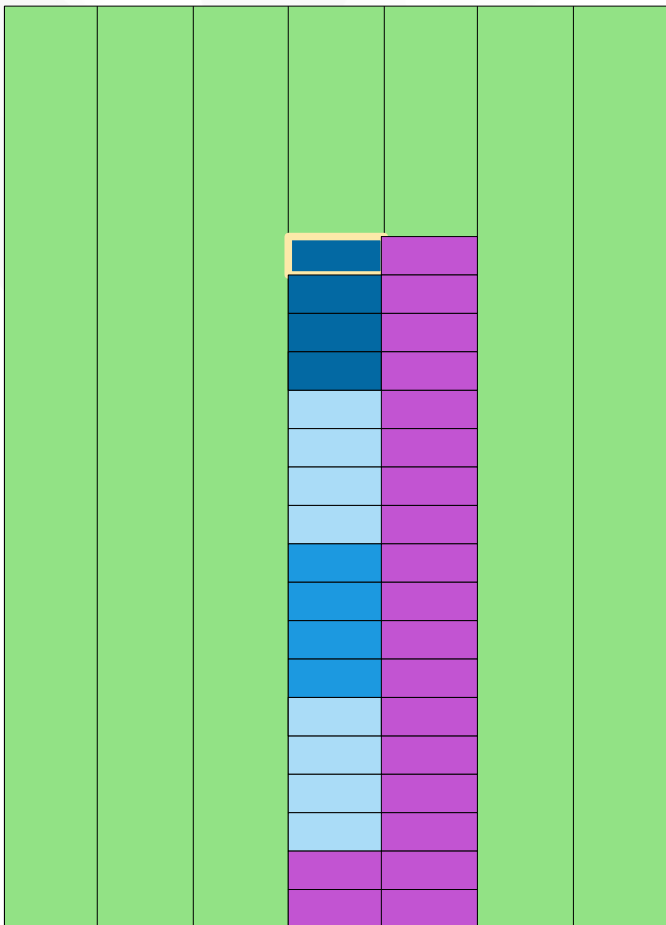
...



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

# Vetores

Os elementos de um vetor são alocados consecutivamente na memória do computador.

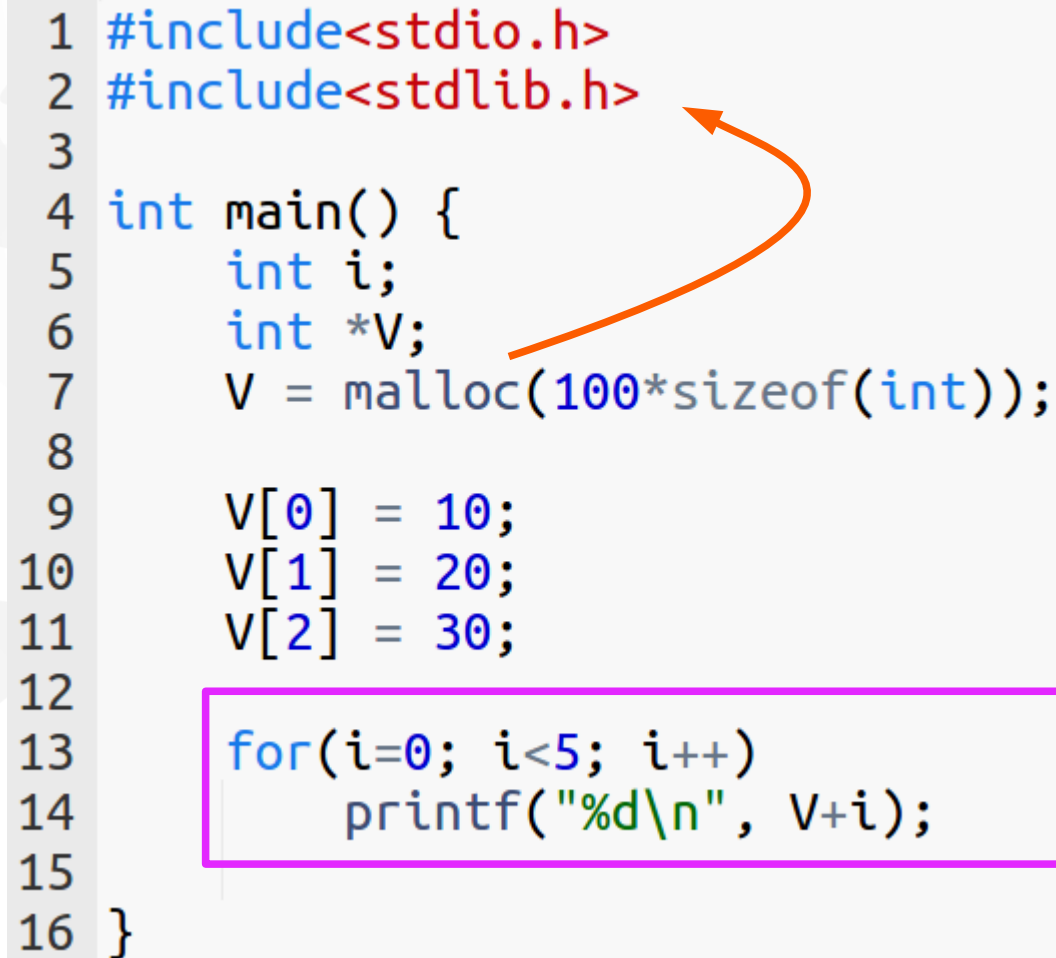


Se cada elemento ocupa **b** bytes, a diferença entre os endereços de dois elementos consecutivos será de **b**.

O compilador C cria a ilusão de que **b** vale 1 qualquer que seja o tipo dos elementos do vetor.

# Alocação memória

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main() {
5     int i;
6     int *V;
7     V = malloc(100*sizeof(int));
8
9     V[0] = 10;
10    V[1] = 20;
11    V[2] = 30;
12
13    for(i=0; i<5; i++)
14        printf("%d\n", V[i]);
15
16 }
```



# Alocação memória

```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main() {
5     int i;
6     int *V;
7     V = malloc(100*sizeof(int));
8
9     V[0] = 10;
10    V[1] = 20;
11    V[2] = 30;
12
13    for(i=0; i<5; i++)
14        printf("%d\n", V+i);
15
16 }
```

**Ideia**  
 $V+i*\text{sizeof}(\text{int})$

.....  
0x17af010  
0x17af014  
0x17af018  
0x17af01c  
0x17af020

# Alocação memória


```
1 #include<stdio.h>
2 #include<stdlib.h>
3
4 int main() {
5     int i;
6     int *V;
7     V = malloc(100*sizeof(int));
8
9     V[0] = 10;
10    V[1] = 20;
11    V[2] = 30;
12
13    for(i=0; i<5; i++)
14        printf("%d\n", *(V+i));    // V[i]
15
16 }
```



10
20
30
0
0

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5
6     int *V = malloc(100*sizeof(int));
7
8     if(V==NULL) {
9         printf("out of memory\n");
10        return 1;
11    }
12
13    *(V+0) = 10;
14    *(V+1) = 20;
15    *(V+99) = 30;
16
17    printf("%d %d %ld\n", V[0], V[99], &V[99]-V+1);
18
19 }
```

Quando não for possível  
Separar memória suficiente  
Um ponteiro NULO é devolvido



10 30 100



```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5
6     int *V = malloc(100*sizeof(int));
7
8     if(V==NULL) {
9         printf("out of memory\n");
10        return 1;
11    }
12
13    *(V+0) = 10;
14    *(V+1) = 20;
15    *(V+99) = 30;
16
17    printf("%d %d %ld\n", V[0], V[99], &V[99]-V+1);
18
19 }
```

Quando não for possível  
Separar memória suficiente  
Um ponteiro nulo é devolvido

10 30 100

A diferença de ponteiros  
Devolve um **long** e é  
Permitida e os dois forem do  
Mesmo tipo



# **Alocação dinâmica de vetores**

Os ponteiros  
facilitam a  
alocação dinâmica  
de memória

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main() {
5     int *v;
6     int n, i;
7
8     scanf( "%d", &n);
9
10    v = (int *) malloc( n*sizeof(int) );
11
12    for (i=0; i<n; i++)
13        scanf( "%d", &v[i]);
14
15    for (i=n; i>0; i--)
16        printf( "%d ", v[i-1]);
17
18    free(v);
19 }
```

6  
1  
2  
3  
4  
5  
6  
6 5 4 3 2 1



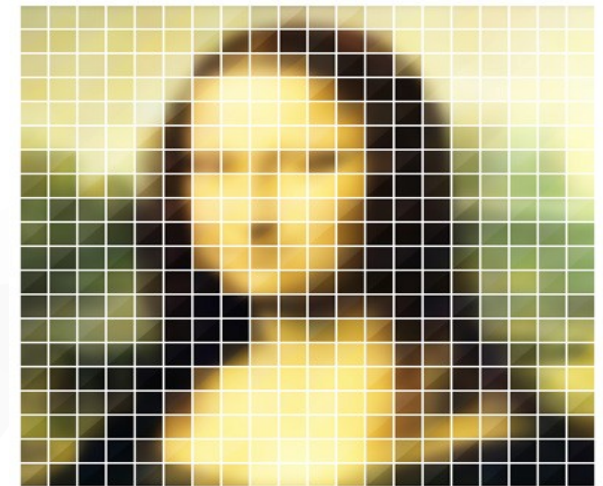
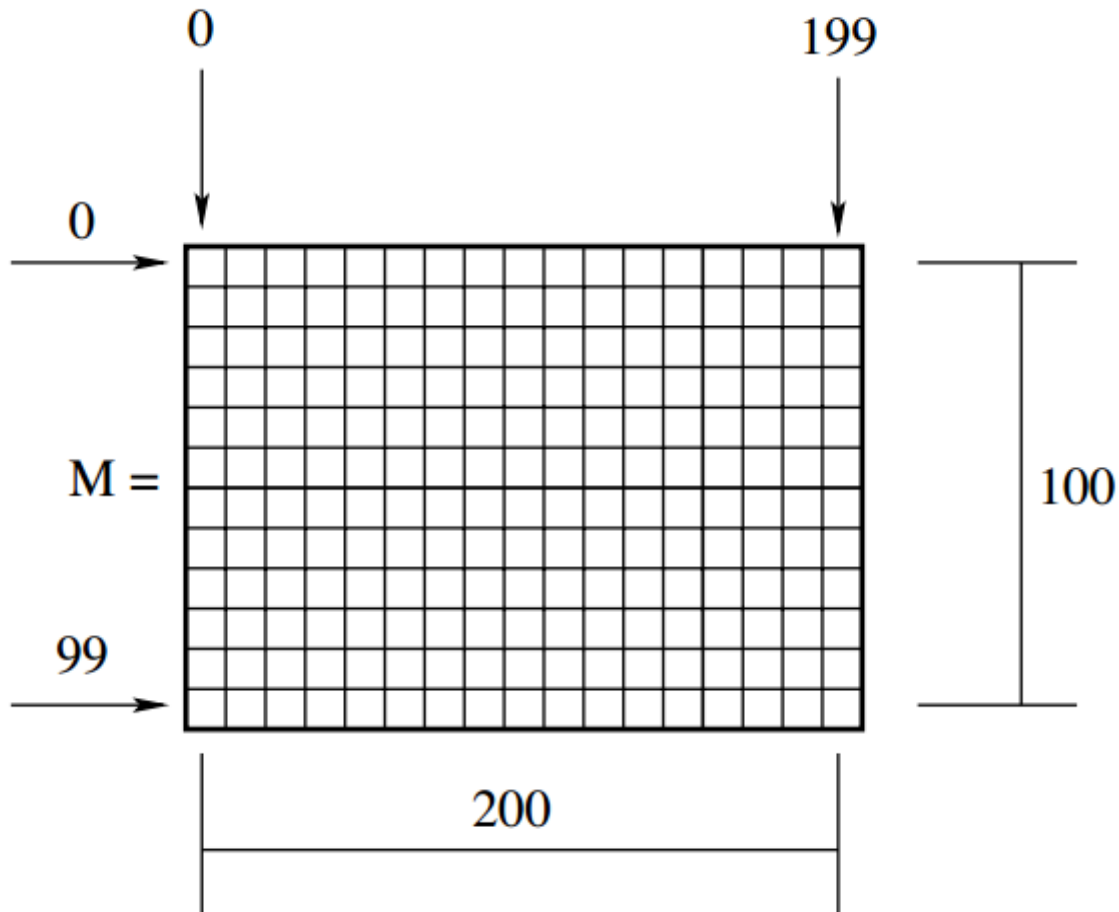
# Matrizes

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

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

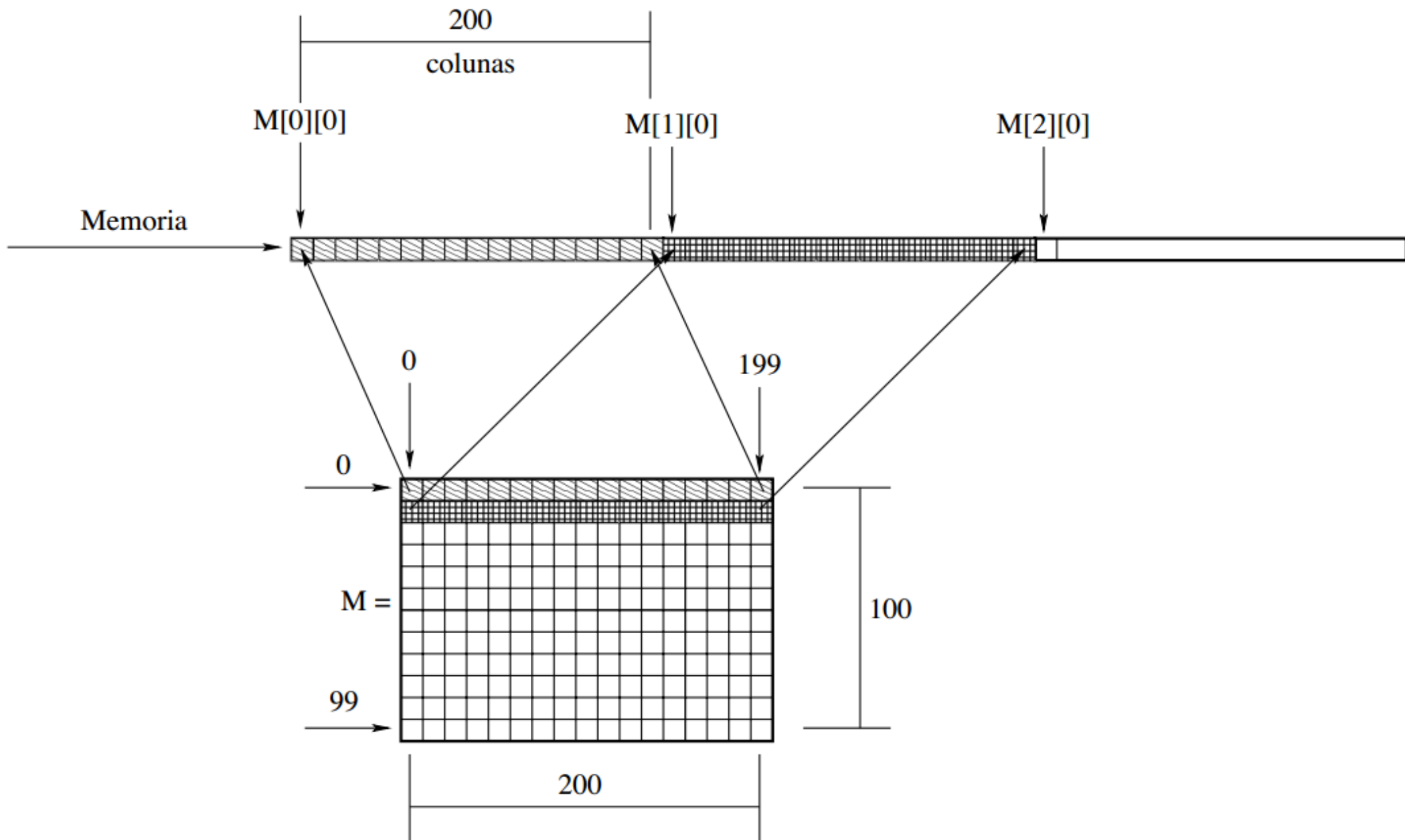
```
int M[100][200];
```

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

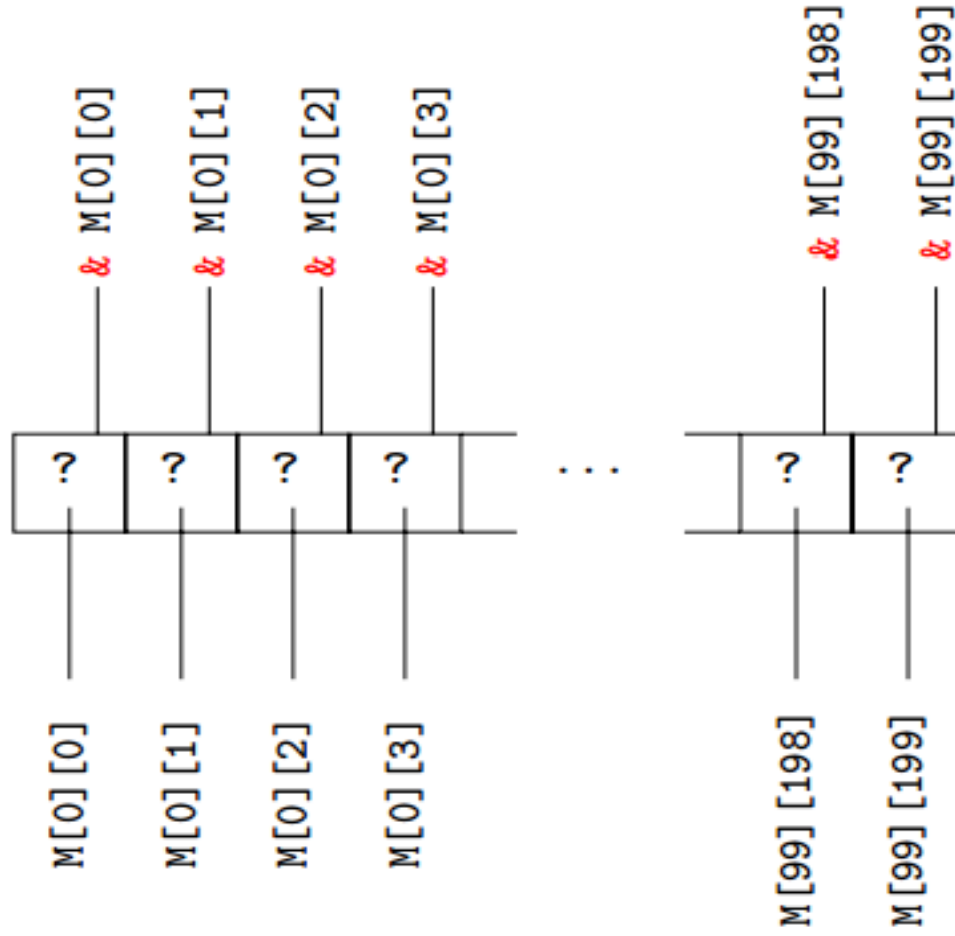


A memória do computador é linear!

## Estrutura da matriz na memória do computador

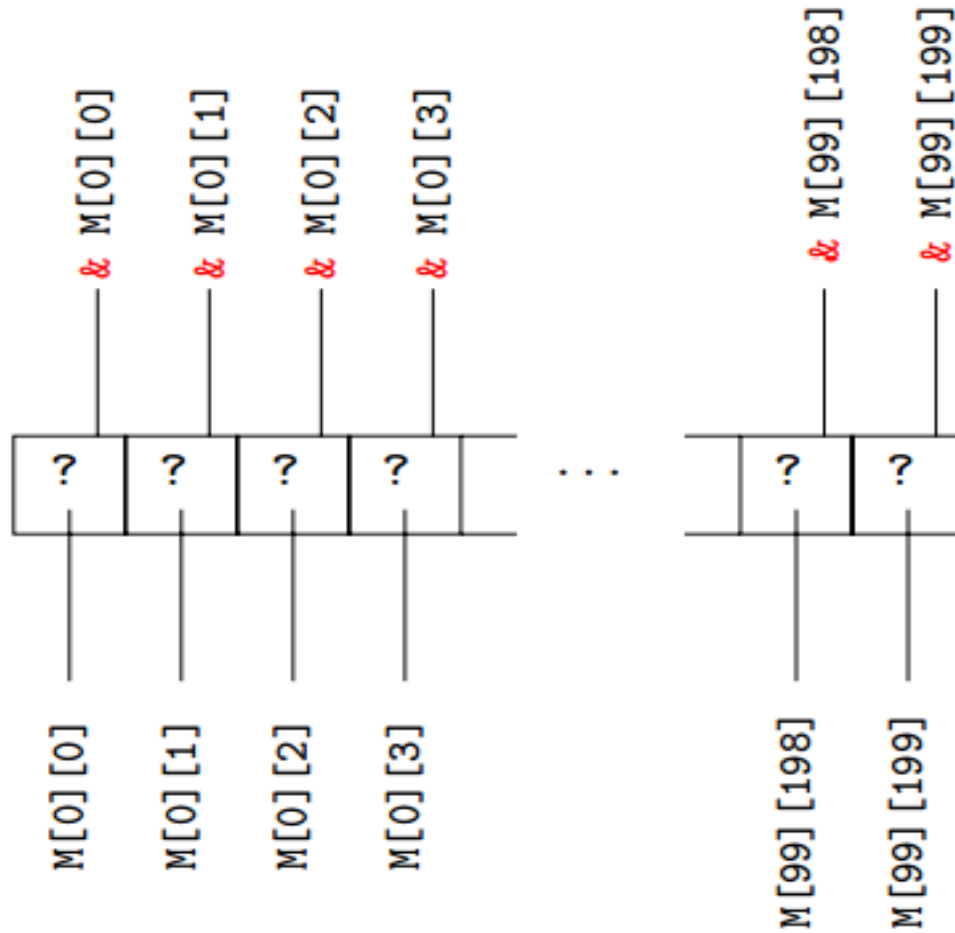


## Disposição dos 20mil elementos da matriz M na memória



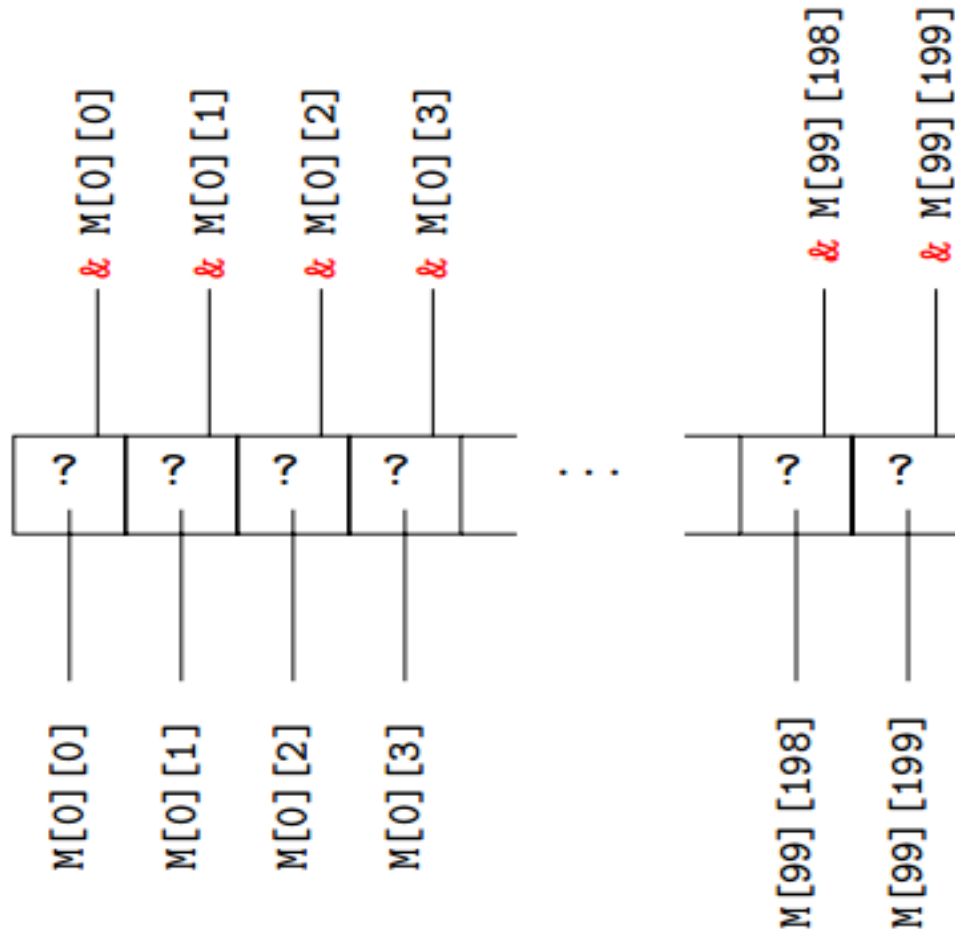


## Disposição dos 20mil elementos da matriz M na memória



Qual o endereço de M[0][78]?  
(tendo como base M[0][0])

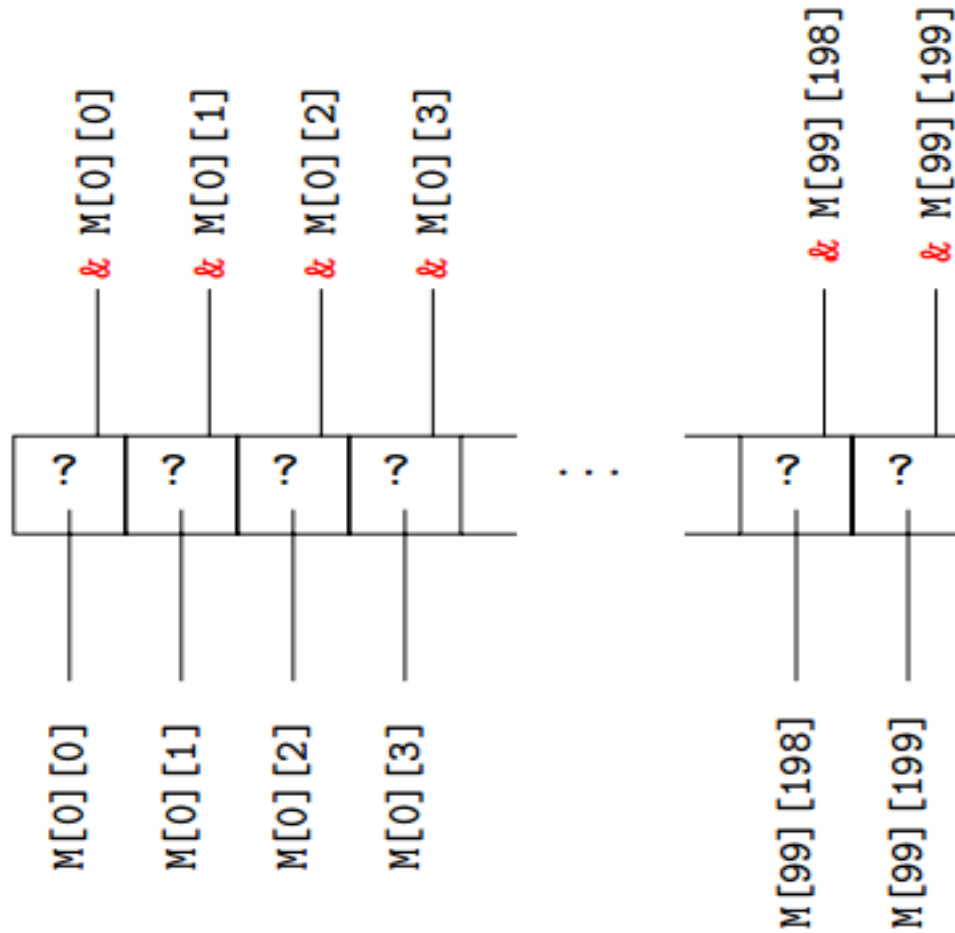
## Disposição dos 20mil elementos da matriz M na memória



Qual o endereço de  $M[0][78]$ ?  
(tendo como base  $M[0][0]$ )

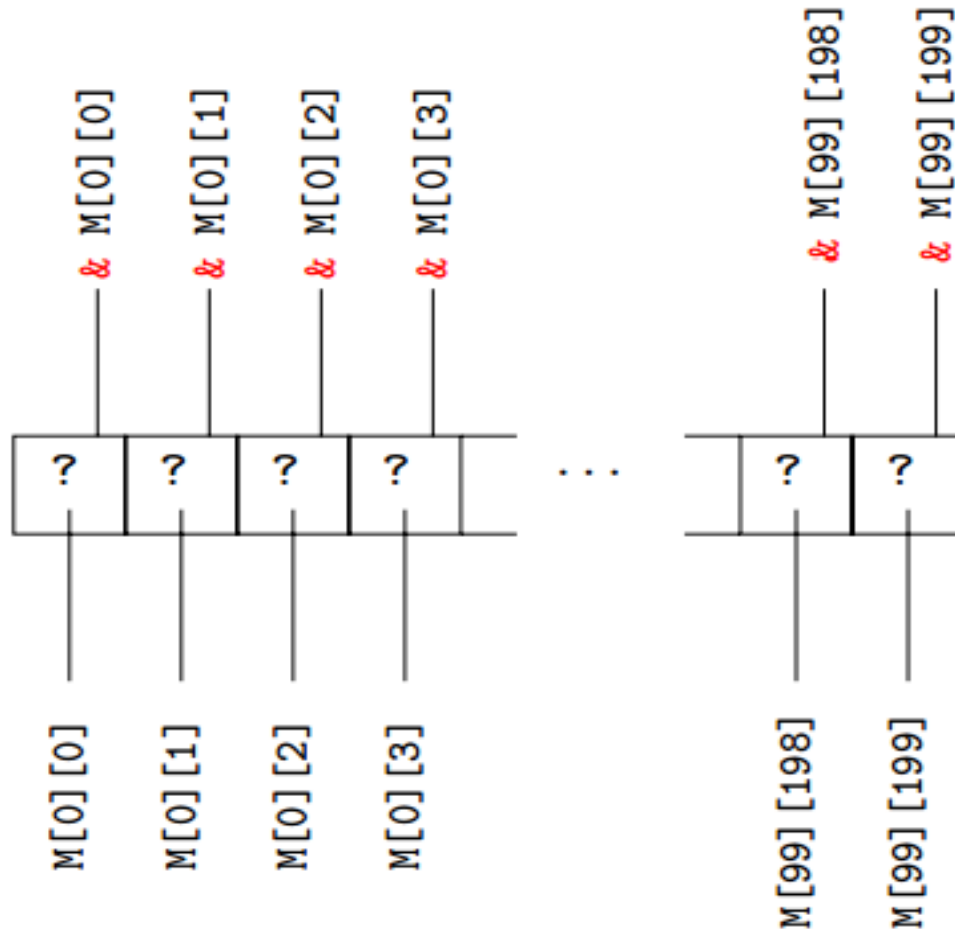
$\&M[0][0] + 78$

## Disposição dos 20mil elementos da matriz M na memória



Qual o endereço de M[78][21]?  
(tendo como base M[0][0])

## Disposição dos 20mil elementos da matriz M na memória



Qual o endereço de  $M[78][21]$ ?  
(tendo como base  $M[0][0]$ )

$$\&M[0][0] + (78 \cdot 200 + 21)$$

# Índices

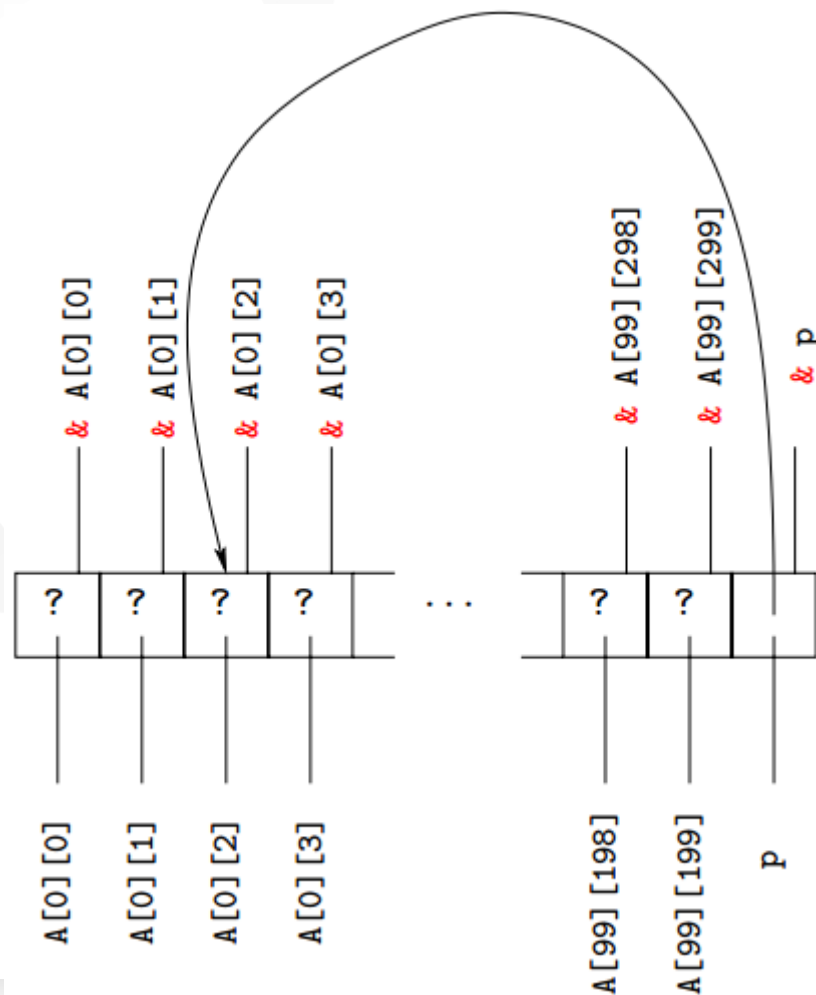
- Na linguagem C **não existe verificação de índices fora da matriz/vetor.**  
Quem deve controlar o uso correto dos índices é o programador.
- O acesso utilizando um índice errado pode ocasionar o acesso de outra variável na memória.
  - Se o acesso à memória é indevido você recebe a mensagem “segmentation fault”.

# Matrizes

```
int A [100][300];  
int *p ;           // ponteiro para inteiro  
p = &A[0][2];      // p aponta para a A[0][2]
```

# Matrizes

```
int A [100][300];  
int *p ;           // ponteiro para inteiro  
  
p = &A[0][2];      // p aponta para a A[0][2]
```



```
int A[100][300];  
int *p, *q;  
  
p = &A[0][0];  
q = A[0];  
  
printf("%p\n%p", p, q);
```

```
0x7fff68497970  
0x7fff68497970
```



```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main()
5 {
6     int A[100][300];
7     int *p = &A[0][0];
8     int i, j;
9
10    for(i=0; i<100; i++)
11        for(j=0; j<300; j++)
12            A[i][j] = i+j;
13
14    printf("A[2][51]    = %d\n", p[2*300+51]);
15    printf("A[99][200] = %d\n", p[99*300+200]);
16    printf("A[99][200] = %d\n", *(p+99*300+200));
17 }
```

```
A[2][51]    = 53
A[99][200]  = 299
A[99][200]  = 299
```

Uma matriz não pode ser considerada um ponteiro para ponteiro


# Função com matriz como parâmetro

- O nome de **uma matriz** dentro do parâmetro de uma função é utilizado **como sendo um ponteiro para o primeiro elemento da matriz** que está sendo usada na hora de usar a função.
  - Não é alocada memória (novamente) para um vetor passado por parâmetro para uma função.

# Função com matriz como parâmetro

## PRÁTICA

Macro!



```
1 #include <stdio.h>
2 #define MAX 100
3
4 void f ( int M[MAX][MAX] ) {
5     M[2][3] = 4;
6 }
7
8
9 int main ( )
10 {
11     int A[MAX][MAX];
12
13     A[2][3] = 5;
14
15     f(A) ;
16
17     printf("%d", A[2][3]);
18 }
```

# Exercício

Escreva um programa que leia um número inteiro positivo **n** seguido de **n** números inteiros e imprima esses **n** números em ordem invertida.

Por exemplo, ao receber

5 222 333 444 555 6666

o seu programa deve imprimir

6666 555 444 333 222

```
6
5 222 333 444 555 6666
6666 555 444 333 222 5
```

**Seu programa não deve impor limitações sobre o valor de n**

Os ponteiros  
facilitam a  
alocação dinâmica  
de memória

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main ( ) {
5     int n, i;
6     scanf("%d", &n);
7
8     int *p = (int *) malloc(n*sizeof(int));
9
10    for(i=0; i<n; i++)
11        scanf("%d", p+i);
12
13    for(i=n-1; i>=0; i--)
14        printf("%d ", *(p+i));
15
16    free(p);
17 }
```

```
6
5 222 333 444 555 6666
6666 555 444 333 222 5
```

# Alocação de memória: estática VS Dinâmica

Na execução, **um programa é um processo.**

**Um processo ocupa parte da memória principal, reservada para:**

- As instruções, e
- A pilha

## **INSTRUÇÕES**

Armazena o código compilado (na linguagem máquina)

[~bytes]

## **PILHA**

Armazena as variáveis ao longo da execução do programa.

## Processo na memória

### INSTRUÇÕES

Armazena o código compilado (na linguagem máquina)

[~bytes]

### PILHA

Armazena as variáveis ao longo da execução do programa.

### HEAP

Espaço de memória principal gerenciado pelo SO.



## Processo na memória

### INSTRUÇÕES

Armazena o código compilado (na linguagem máquina)

[~bytes]

### PILHA

Armazena as variáveis ao longo da execução do programa.

Alocação estática

### HEAP

Espaço de memória principal gerenciado pelo SO.

Alocação dinâmica

## Processo na memória

### INSTRUÇÕES

Armazena o código compilado (na linguagem máquina)

[~bytes]

### PILHA

Armazena as variáveis ao longo da execução do programa.

Alocação estática

```
int x;  
double M[10][20];
```

### HEAP

Espaço de memória principal gerenciado pelo SO.

Alocação dinâmica

```
double M = malloc(...);
```

## Processo na memória

### INSTRUÇÕES

Armazena o código compilado (na linguagem máquina)

[~bytes]

### PILHA

Armazena as variáveis ao longo da execução do programa.

Alocação estática

```
int x;  
double M[10][20];
```

### HEAP


Espaço de memória principal gerenciado pelo SO.

Alocação dinâmica

```
double M = malloc(...);
```

Aqui é recomendável desalocar a memória  
`free()`


# Is Fibo

 Authored by [shashank21j](#) on Dec 05 2013

[Problem](#)[Submissions](#)[Leaderboard](#)[Discussions](#)[Editorial !\[\]\(c694a3ff3b077d76910920a6a1593ab4\_img.jpg\)](#)



Submitted a few seconds ago • Score: 11.11

Status: **Wrong Answer**

# 0  0.02s : Success  (sample)



# 1  0.02s : Success 



# 2  0.02s : Success 



# 3  0.02s : Success 



# 4  0.02s : Success 

# 5  0.02s : Success 

# 6  0.02s : Wrong Answer 

# 7  0.02s : Wrong Answer 

# 8  0.02s : Wrong Answer 

# 9  0.02s : Wrong Answer 

## Submitted Code

Language: C

 [Open in editor](#)

```
1 #include <stdio.h>
2 #include <string.h>
3 #include <math.h>
4 #include <stdlib.h>
5
6 char* isFibo(int n) {
7     int fib1 = 0;
8     int fib2 = 1;
9     int fib = 1;
10
11     while (fib<=n) {
12         fib = fib1 + fib2;
13         fib1 = fib2;
14         fib2 = fib;
15         if (fib==n)
16             return "IsFibo";
17     }
18     return "IsNotFibo";
19 }
```

1 1  
2 1  
3 2  
4 3  
5 5  
6 8  
7 13  
8 21  
9 34  
10 55  
11 89  
12 144  
13 233  
14 377  
15 610  
16 987  
17 1597  
18 2584  
19 4181  
20 6765  
21 10946  
22 17711  
23 28657  
24 46368  
25 75025  
26 121393  
27 196418  
28 317811  
29 514229  
30 832040

31 1346269  
32 2178309  
33 3524578  
34 5702887  
35 9227465  
36 14930352  
37 24157817  
38 39088169  
39 63245986  
40 102334155  
41 165580141  
42 267914296  
43 433494437  
44 701408733  
45 1134903170  
46 1836311903  
47 2971215073  
48 4807526976  
49 7778742049  
50 12586269025  
51 20365011074  
52 32951280099  
53 53316291173  
54 86267571272  
55 139583862445  
56 225851433717  
57 365435296162  
58 591286729879  
59 956722026041  
60 1548008755920

13 digitos

# Lista 03: <https://www.hackerrank.com>

- **Filling Jars**
- **Alternating Characters**
- **Two Strings**

Será utilizado um programa de detecção de plágio em todas as submissões!  
Plágio → reprovação

**Data:** 08/Março (domingo) até às 23h50.

**Envio:** Através do Tidia.

## Arquivos:

Para cada exercício-problema deverá submeter:

- O código fonte: nome do arquivo → RA\_nomeDoProblema.c
- O comprovante de aceitação (screenshot) → RA\_nomeDoProblema.pdf

Exemplo: 10123456\_solveMeFirst.c  
10123456\_solveMeFirst.pdf