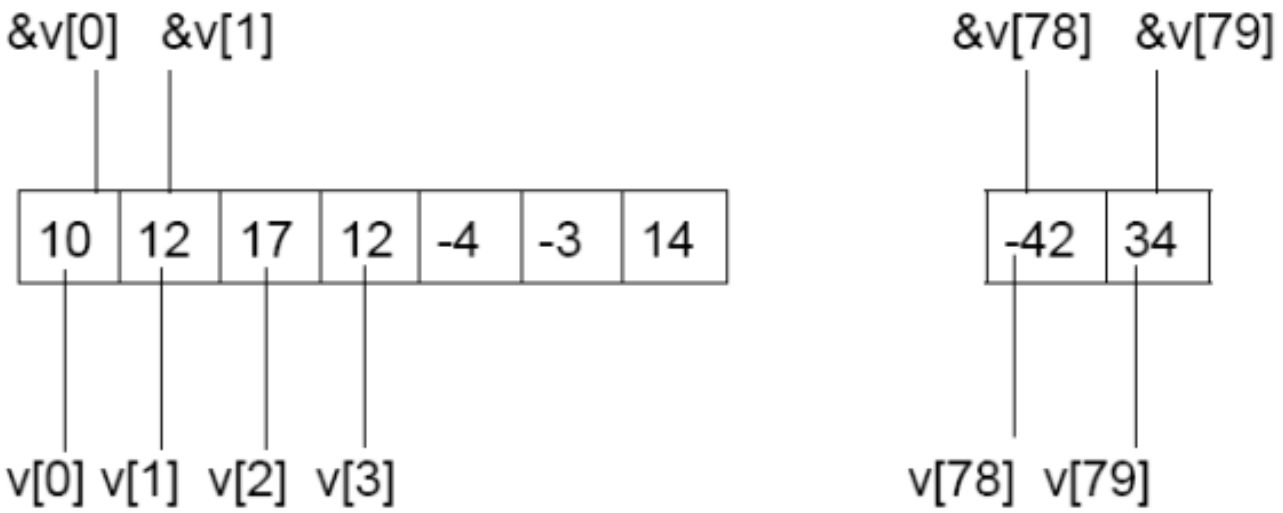
Vetores e ponteiros

Vetores

- Estruturas indexadas que armazenam dados do mesmo tipo

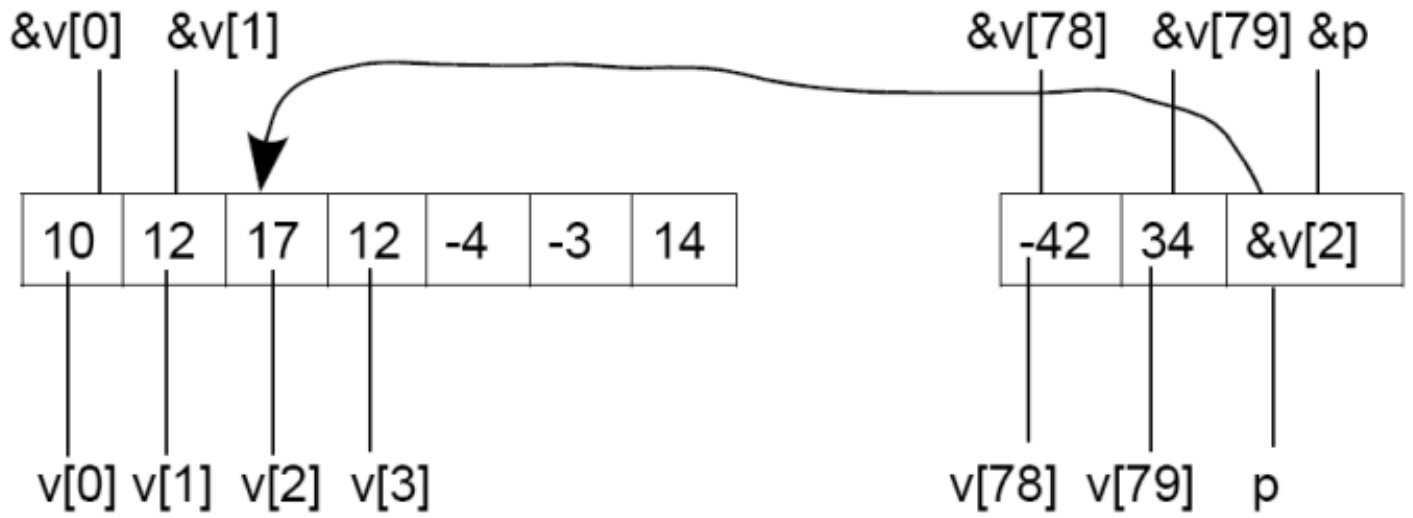


Vetores e ponteiros

- Facilita a manipulação de vetores:

```
int v[80];
int *p;

p = &v[2];
```

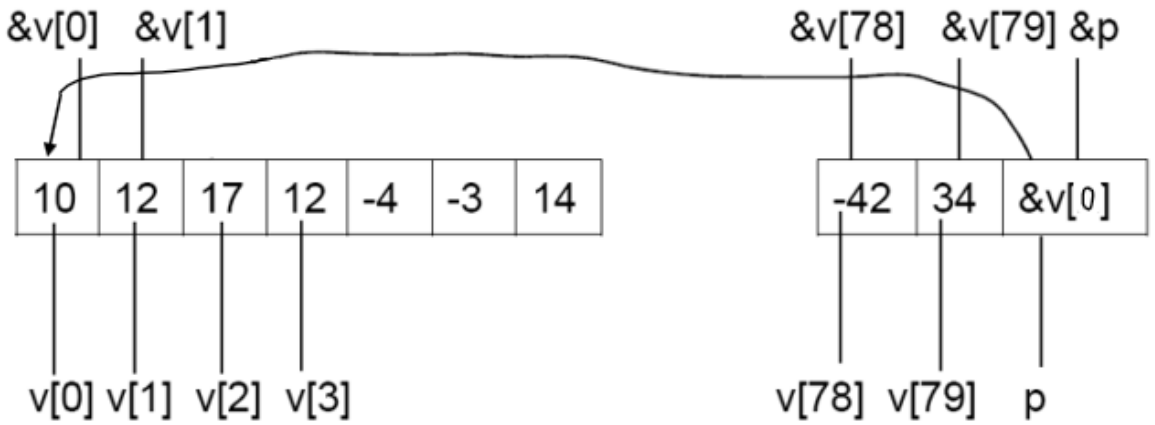


Vetores e ponteiros

- Aqui podemos usar também uma sintaxe especial, para fazer o ponteiro apontar para o início do vetor

```
int v[80];
int *p;

p = v;
```



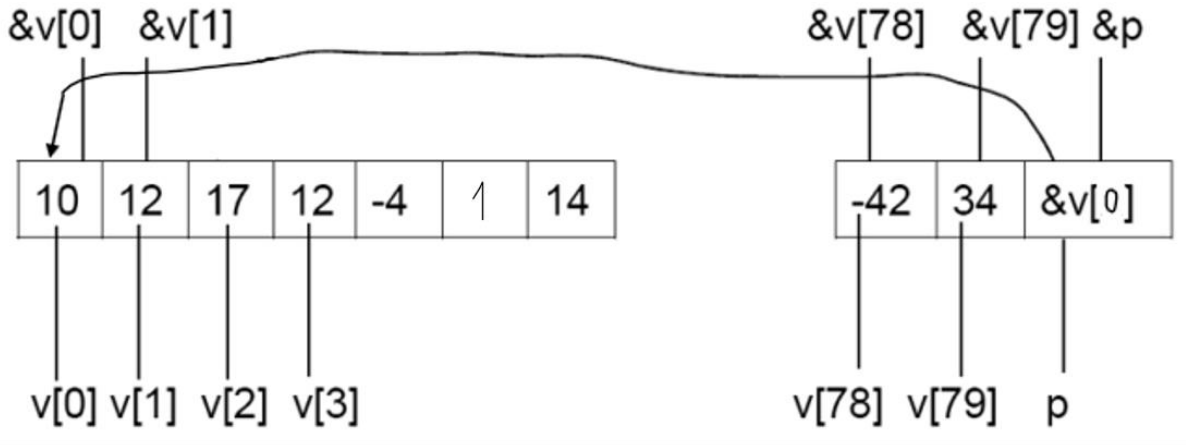
- Equivalente à p = &v[0];

Vetores e ponteiros

- Aqui podemos usar também uma sintaxe especial, para fazer o ponteiro apontar para o início do vetor

```
int v[80];
int *p;

p = v;
p[5] = 1;
```



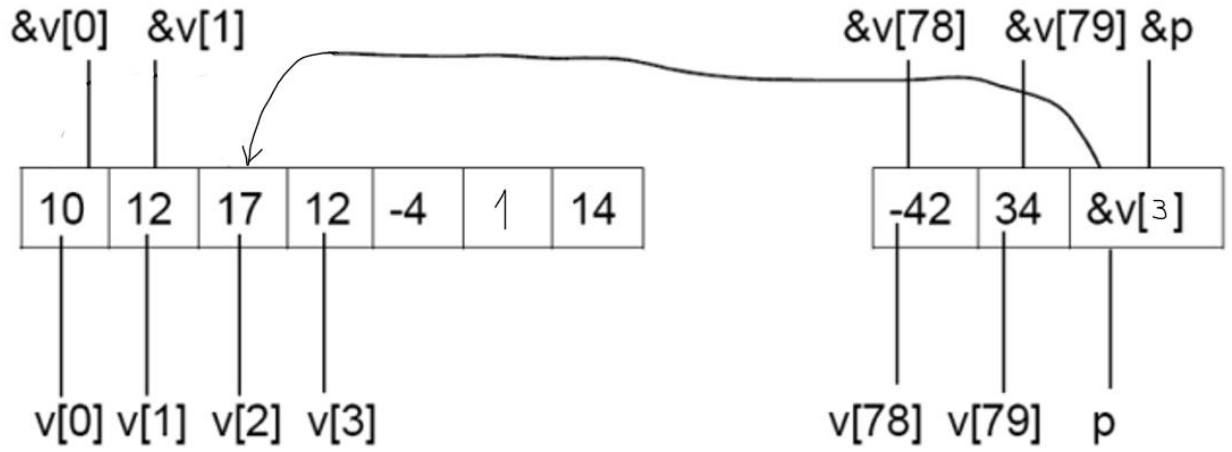
- Equivalente à v[5] = 1;

Vetores e ponteiros

- Aqui podemos usar também uma sintaxe especial, para fazer o ponteiro apontar para o início do vetor

```
int v[80];
int *p;

p = &v[3];
```



- Então, p[1] é o elemento v[4], p[2] o v[5] e assim por diante..

Operações

```
int *p, *q, n, v[50];
float *x, y[20];
```

- Se p = &v[4] ,
  - (p + 1) é &v[5] ; (p + 2) é &v[6] e assim por diante
  - *p é v[4], portanto se v[4] = = 3, *p = = 3.
  - *(p+1) é v[5] ..
- Se x = &y[3]
  - (x+1) = = &y[4].. (x+i) = = &y[3+1]

Operações

```
int *p, *q, n, v[50];
float *x, y[20];
```

- Somar ou subtrair um inteiro de um ponteiro

```
p = &v[22]; q = &v[30];
p = p - 4; q++;
*(p+2) = 3; *q = 4;
```

Operações

```
int *p, *q, n, v[50];
float *x, y[20];
```

- Somar ou subtrair um inteiro de um ponteiro

```
p = &v[22]; q = &v[30];
p = p - 4; q++;
*(p+2) = 3; *q = 4;
```


v[20] = 3
v[31] = 4

Vetores como parâmetros

```
# include <math.h>
float modulo (float v[], int n) {
  int i;
  float r = 0;
  for (i=0; i<n; i++) {
    r = r + v[i]*v[i];
  }
  r = sqrt (r);
  return r;
}
```

Vetores como parâmetros

```
# include <math.h>
float modulo (float v[], int n) {
  int i;
  float r = 0;
  for (i=0; i<n; i++) {
    r = r + v[i]*v[i];
  }
  r = sqrt (r);
  return r;
}
```

```
float modulo (float *p , int n) {
  int i;
  float r = 0;
  for (i=0; i<n; i++) {
    r = r + p[i]*p[i];
  }
  r = sqrt (r);
  return r;
}
```

float v[] é a mesma coisa que *p → v é um ponteiro!

Vetores como parâmetros

O parâmetro v da função modulo aponta para a variável x[0] da função main.

Então v[i] na função modulo é exatamente x[i] da função main.

v aponta para x[0]. Então v[i] é x[i]

```
1   # include <stdio.h>
2   # include <math.h>
3
4
5   float modulo (float v[], int n) {
6     int i;
7     float r = 0;
8     for (i=0; i<n; i++) {
9       r = r + v[i]*v[i];
10    }
11    r = sqrt (r);
12    return r;
13  }
14
15  int main () {
16    float x[100], comprimento;
17    int m;
18
19    m = 3;
20    x[0] = 2; x[1] = -3, x[2] = 4;
21
22
23    comprimento =  modulo (x, m);
24
25    printf ("Comprimento = %f\n", comprimento);
26
27    return 0;
28  }
```

Vetores como parâmetros

O parâmetro v da função modulo aponta para a variável x[0] da função main.

Então v[i] na função modulo é exatamente x[i] da função main.

u aponta para v[0].

```
1   # define MAX 200
2
3
4   float f (float u[]) {
5     float s;
6     /* declaração da função f */
7     ...
8     u[i] = 4;
9     ...
10    return s;
11  }
12
13  int main () {
14    float a, v[MAX]; /* declaração da variável a e vetor v */
15    ...
16    /* outras coisas do programa */
17
18
19    a = f (v);   /* observe que o vetor é passado apenas pelo nome */
20
21    ...
22
23    return 0;
24  }
```

Problema

Faça uma função que recebe dois vetores de tamanho n e retorna o seu produto escalar.

O protótipo dessa função seria:

```
float ProdutoEscalar (float u[], float v[], int n);
```

Problema

Faça uma função que recebe dois vetores de tamanho n e retorna o seu produto escalar.

O protótipo dessa função seria:

```
float ProdutoEscalar (float u[], float v[], int n);
```

A função recebe como parâmetros os vetores u e v, e um inteiro n. Uma possível solução para esta função seria:

```
float ProdutoEscalar (float u[], float v[], int n) {
  int i;
  float res = 0;
  for (i=0; i<n; i++)
  res = res + u[i] * v[i];
  return res;
}
```

Passagem de Parâmetros

- Em C, por padrão, o que existe é a passagem de parâmetro por valor, ou seja, é sempre criada uma variável como uma cópia.

- Logo, deve-se implementar a passagem por referência utilizando-se de ponteiros.

- Cuidado com a alocação de memória!

Passagem de Parâmetros – Valor vs. Referência

```
void SomaUm(int x, int *y) {
	x = x + 1;
	*y = (*y) + 1;
	printf("Funcao SomaUm: %d %d\n", x, *y);
}

int main() {
	int a = 0, b = 0;
	SomaUm(a, &b);
	printf("Programa principal: %d %d\n", a, b);
	return 0;
}
```

Passagem de Parâmetros – Valor vs. Referência

```
void SomaUm(int x, int *y) {
        x = x + 1;
        *y = (*y) + 1;
        printf("Funcao SomaUm: %d %d\n", x, *y);
}

int main() {
        int a = 0, b = 0;
        SomaUm(a, &b);
        printf("Programa principal: %d %d\n", a, b);
        return 0;
}
```

Funcao SomaUm: 1 1
Programa principal: 0 1

Exemplo 01

Faça um programa que leia um valor n, crie dinamicamente um vetor de n elementos inteiros e passe esse vetor para uma função que deverá preenchê-lo com valores digitados pelo usuário. A leitura do vetor deverá ser feita via scanf. O programa deve mostrar o vetor preenchido.

Importante: não se esqueça de liberar a memória alocada para o vetor.

Exemplo 02

Escreva uma função que receba como parâmetro um vetor de inteiros (passagem por referência) e o tamanho do vetor (passagem por valor). A função deve retornar o maior elemento do vetor.

a) Escreva uma solução iterativa usando os índices do vetor.

b) Escreva uma solução iterativa usando o ponteiro do vetor.

Exemplo 02

Escreva uma função que receba como parâmetro um vetor de inteiros (passagem por referência) e o tamanho do vetor (passagem por valor). A função deve retornar o maior elemento do vetor.

a) Escreva uma solução iterativa usando os índices do vetor.

b) Escreva uma solução iterativa usando o ponteiro do vetor.

c) Escreva uma solução recursiva.

```c
int maiorElementoIndice(int vetor[], int tamanho) {
    if (tamanho <= 0) return -1;

    int maior = vetor[0];
    for (int i = 1; i < tamanho; i++) {
        if (vetor[i] > maior) {
            maior = vetor[i];
        }
    }
    return maior;
}
```

```c
int maiorElementoIndice(int vetor[], int tamanho) {
    if (tamanho <= 0) return -1;

    int maior = vetor[0];
    for (int i = 1; i < tamanho; i++) {
        if (vetor[i] > maior) {
            maior = vetor[i];
        }
    }
    return maior;
}
```

```c
int maiorElementoPonteiro(int *vetor, int tamanho) {
    if (tamanho <= 0) return -1;

    int maior = *vetor; // primeiro elemento
    for (int i = 1; i < tamanho; i++) {
        if (*(vetor + i) > maior) {
            maior = *(vetor + i);
        }
    }
    return maior;
}
```

```c
int maiorElementoIndice(int vetor[], int tamanho) {
    if (tamanho <= 0) return -1;

    int maior = vetor[0];
    for (int i = 1; i < tamanho; i++) {
        if (vetor[i] > maior) {
            maior = vetor[i];
        }
    }
    return maior;
}
```

```c
int maiorElementoPonteiro(int *vetor, int tamanho) {
    if (tamanho <= 0) return -1;

    int maior = *vetor; // primeiro elemento
    for (int i = 1; i < tamanho; i++) {
        if (*(vetor + i) > maior) {
            maior = *(vetor + i);
        }
    }
    return maior;
}
```

```c
#include <stdio.h>

int main() {
    int numeros[] = {3, 7, 2, 9, 5};
    int tamanho = 5;

    int maior1 = maiorElementoIndice(numeros, tamanho);
    int maior2 = maiorElementoPonteiro(numeros, tamanho);

    printf("Maior (índice): %d\n", maior1);
    printf("Maior (ponteiro): %d\n", maior2);

    return 0;
}
```

Exemplo 03

Problema de avaliação de um polinômio: escreva uma função que receba um vetor com os coeficientes $(a_0, a_1, \ldots, a_n)$, receba o valor de x e retorne o valor do polinônio para um valor n recebido.

$$P(n) = a_0 + a_1 x^1 + a_2 x^2 + \ \ldots + a_n x^n$$

a) Solução iterativa.

b) Solução recursiva.

Exmplo 04

Escreva uma função em C que receba dois vetores como parâmetros e verifica se o segundo vetor ocorre dentro do primeiro.

Matrizes e ponteiros

Matrizes

- Estruturas indexadas (em forma matricial) utilizadas para armazenar valores de um mesmo tipo.

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

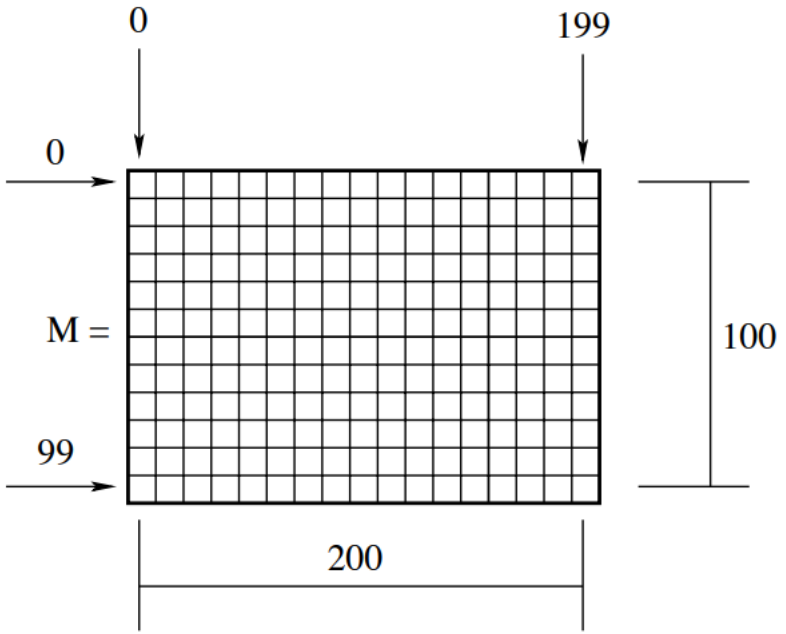
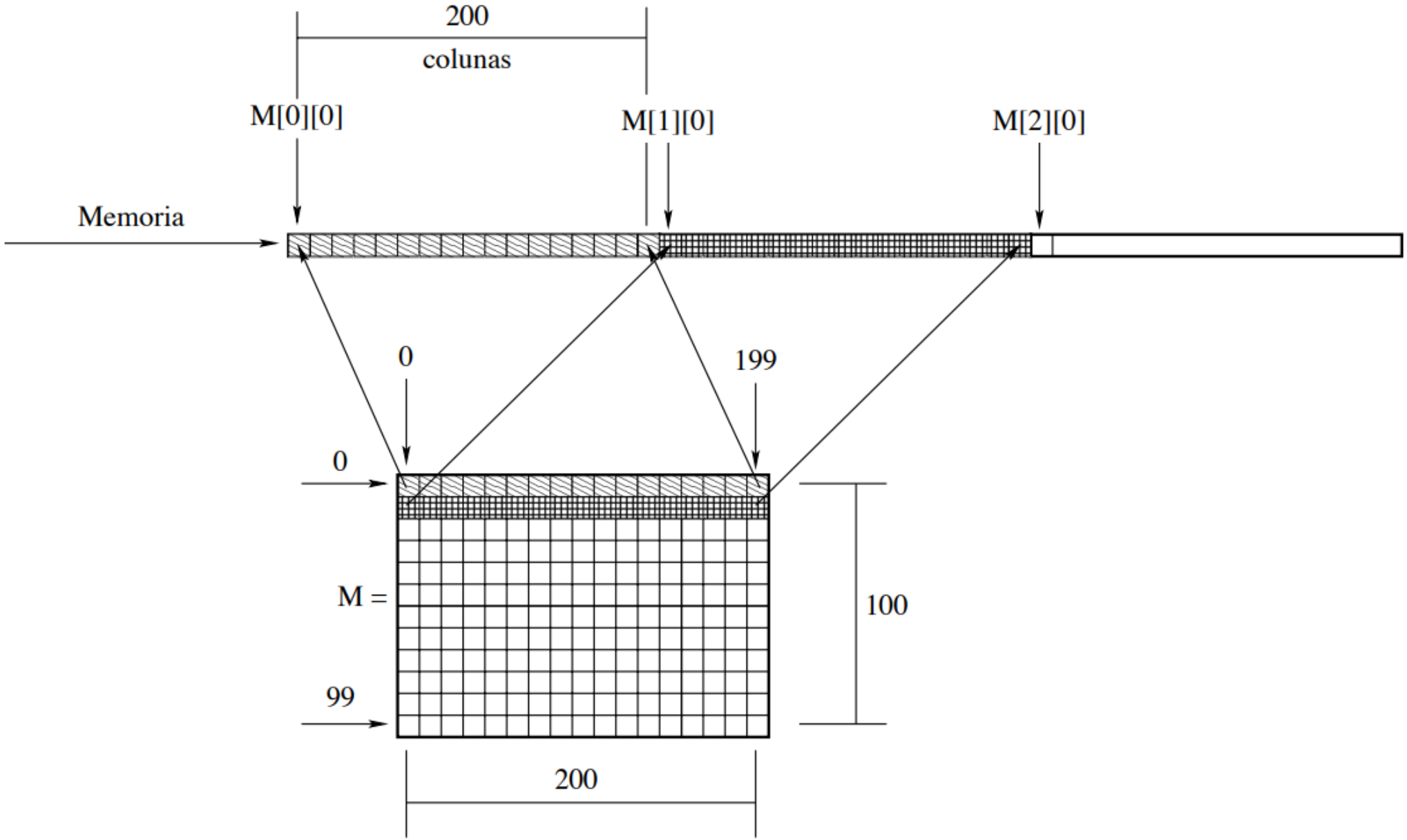


Figura 1: Estrutura de uma matriz int M[100][200].

Matrizes

Como uma matriz fica armazenada na memória?

R: cada linha da matriz uma em seguida da outra


200
colunas
M[0][0]
M[1][0]
M[2][0]
Memoria
0
199
0
M =
99
100
200

Matrizes

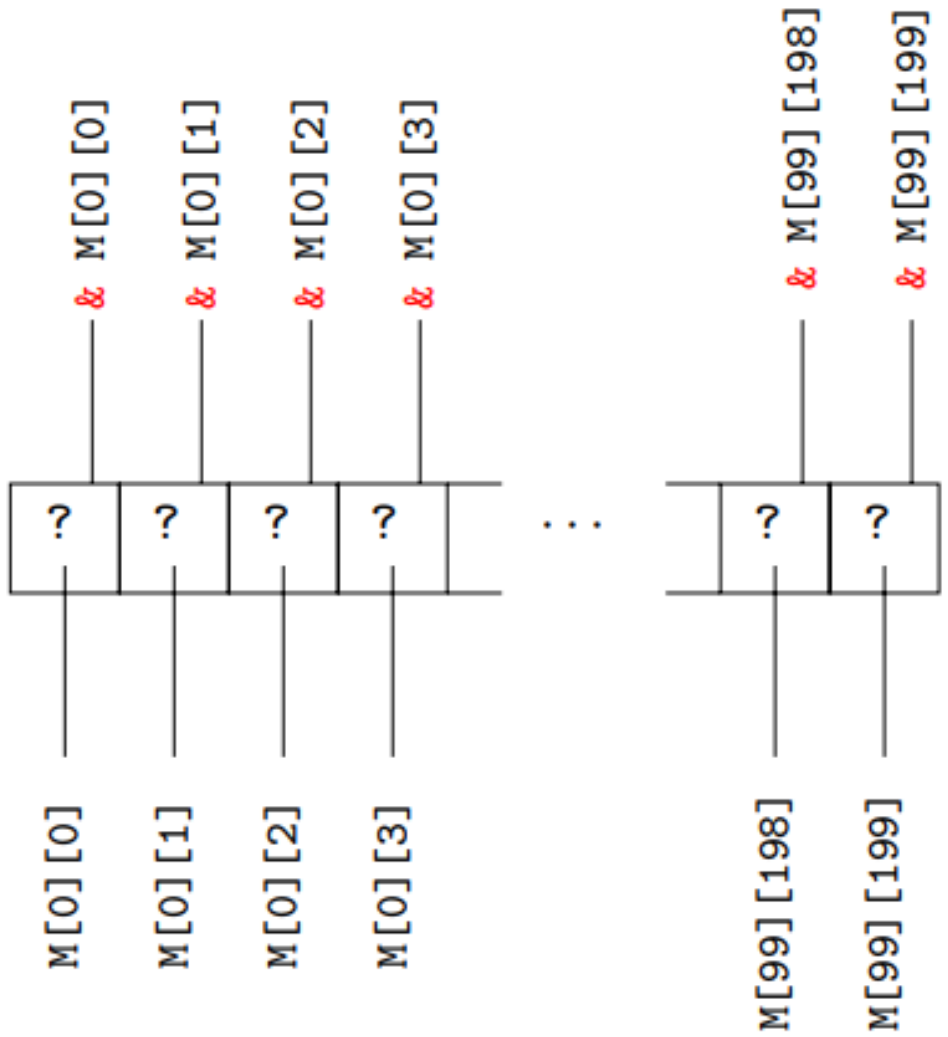
- Isso significa, para a matriz int M[100][200]:

M[0][0], M[0][1]...,M[0][199],M[1][0],M[1][1],.....M[1][199]....

1ª linha

2ª linha

Matrizes



Disposição dos 20.000 elementos da matriz `M` na memória.

Matrizes

- Vamos supor, para efeitos didáticos, que o endereço de memória de &M[0][0] seja 10 e que os endereços seguintes sejam os números consecutivos..
- &M[0][1] = 11... &M[99][199] = 20.0009

- Ou seja, conhecendo o 1º endereço, consigo determinar o restante

Matrizes

- Vamos supor, para efeitos didáticos, que o endereço de memória de &M[0][0] seja 10 e que os endereços seguintes sejam os números consecutivos..
- &M[0][1] = 11... &M[99][199] = 20.0009

- Ou seja, conhecendo o 1º endereço, consigo determinar o restante
- Exemplo: &M[0][78] = M[0][0] + 78 = 88

Matrizes

- Vamos supor, para efeitos didáticos, que o endereço de memória de &M[0][0] seja 10 e que os endereços seguintes sejam os números consecutivos..
- &M[0][1] = 11... &M[99][199] = 20.0009

- Ou seja, conhecendo o 1º endereço, consigo determinar o restante
- E qual seria o endereço de M[78][21] ?

Matrizes

- Para determinar o endereç de memória a um determinado elemnto M[i][j], é feito internamente uma conta de endereçamento:

- M[i][j] = &M[0][0] + i.c + j , onde c é o número de colunas

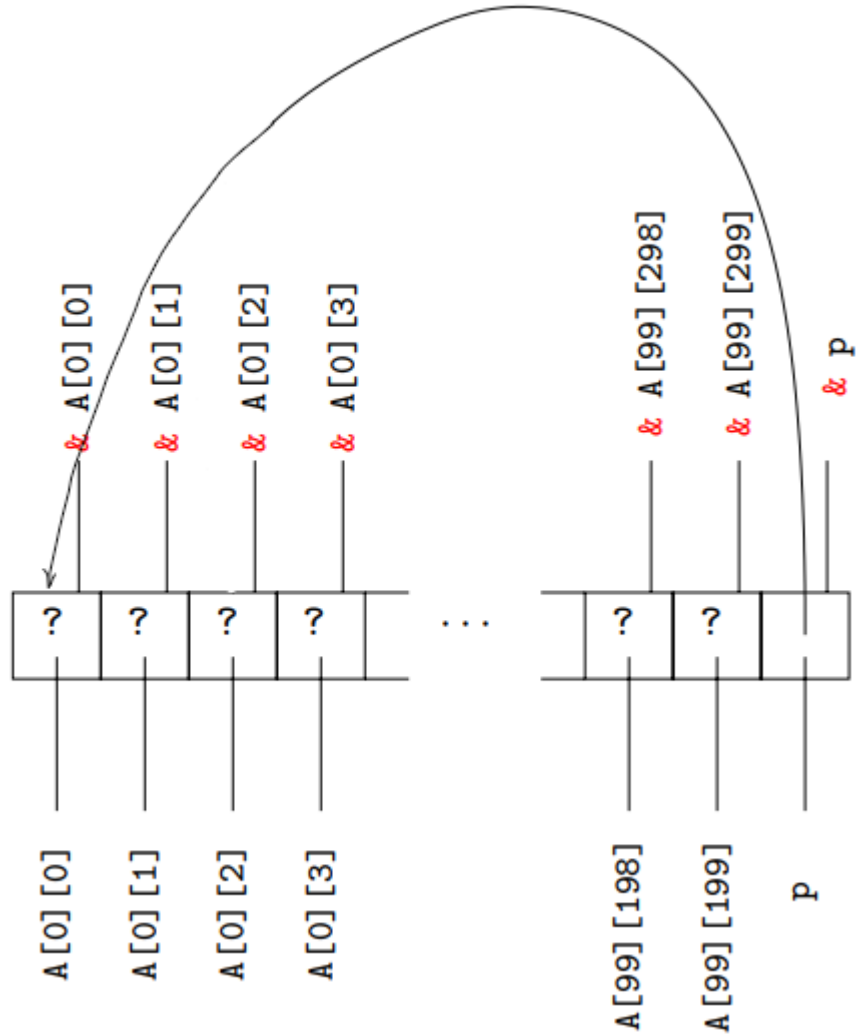
A gente não vai fazer essa conta, o compilador já faz. Mas perceba que precisamos fornecer pra ele o número de colunas.

Matrizes e ponteiros

```
int A[100][300];
int *p; /* ponteiro para inteiro */

p = &A[0][0];
```

O ponteiro pode ir percorrendo a "matriz" a medida em que anda com o endereço



Matrizes e ponteiros

```
int A[100][300], i, j;
int *p; /* ponteiro para inteiro */
i = 3; j = 4;
p = A[0]; /* p aponta para o 1o elemento da 1a linha de A */
          /* Equivale a fazer p = &A[0][0] */
p[i*300+j] = 4; /* equivale a fazer M[i][j] = 4 */
```

"anda" i vezes a quantidade de colunas, e soma o j

Matrizes como Parâmetros de Funções

```
# include <math.h>
float soma_diagonal (float B[300][300], int n) {
  int i;
  float r = 0;
  for (i=0; i<n; i++) {
    r = r + B[i][i];
  }
  return r;
}
```

O parâmetro B da função **soma_diagonal** aponta para a variável C[0][0] da função main.

Então B[i][i] na função **soma_diagonal** **é** exatamente C[i][i] da função main.

B aponta para C[0][0]. Então B[i][i] é C[i][i]

```
1   # include <stdio.h>
2   # include <math.h>
3
4
5   float soma_diagonal (float B[300][300], int n) {
6     int i;
7     float r = 0;
8     for (i=0; i<n; i++) {
9       r = r + B[i][i];
10    }
11    return r;
12  }
13
14  int main () {
15    float C[300][300], soma;
16    int m;
17
18    m = 3;
19    C[0][0] = 2; C[0][1] = -2, C[0][2] = 4;
20    C[1][0] = 3; C[1][1] = -1, C[1][2] = 7;
21    C[2][0] = 5; C[2][1] = -3, C[2][2] = 3;
22
23
24    soma = soma_diagonal (C, m);
25
26    printf ("Soma = %f\n", soma);
27
28    return 0;
29  }
```

Exemplo

```
1   # include <stdio.h>
2
3   # define MAX 100
4
5   int f (int M[MAX][MAX], int n) {
6     n = 10;
7     M[2][3] = 4;
8     return 0;
9   }
10
11  int main () {
12    int A[MAX][MAX], m, a;
13    m = 15;
14    A[2][3] = 5;
15    a = f (A, m);
16    return 0;
17  }
```

Faça uma função que recebe como entrada um inteiro n, uma matriz inteira $A_{n \times n}$ e devolve três inteiros: k, Lin e Col. O inteiro k é um maior elemento de A e é igual a $A[Lin][Col]$.

Obs.: Se o elemento máximo ocorrer mais de uma vez, indique em Lin e Col qualquer uma das possíveis posições.

Exemplo: se $A = \begin{pmatrix} 3 & 7 & 1 \\ 1 & 2 & 8 \\ 5 & 3 & 4 \end{pmatrix}$ então $\begin{cases} k = 8 \\ Lin = 1 \\ Col = 2 \end{cases}$

```
# define MAX 100

void maior (int A[MAX][MAX], int n, int *k, int *Lin, int *Col);
```

```c
# define MAX 100

void maior (int A[MAX][MAX], int n, int *k, int *Lin, int *Col) {
  int i, j;

  /* percorrimento da matriz A por linha */
  for (i=0; i<n; i++) {
    for (j=0; j<n; j++) {
      if (A[i][j] > *k) {
        *k = A[i][j];
        *Lin = i;
        *Col = j;
      }
    }
  }

}
```