

# Vetores (arrays)

Prof. Bruno Travençolo

# Relembrando..

- Faça um programa para ler 5 números e mostrar o resultado da soma desses números

```
int main()
{
    double val1, val2, val3, val4, val5, soma;

    printf("\nDigite o 1o. numero: ");
    scanf("%lf", &val1);

    printf("\nDigite o 2o. numero: ");
    scanf("%lf", &val2);

    printf("\nDigite o 3o. numero: ");
    scanf("%lf", &val3);

    printf("\nDigite o 4o. numero: ");
    scanf("%lf", &val4);

    printf("\nDigite o 5o. numero: ");
    scanf("%lf", &val5);

    soma = val1 + val2 + val3 + val4 + val5;

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

É se for para somar  
100 números?  
Usar 100 variáveis?

## ► Relembrando

```
int main()
{
    double val, soma;
    int contagem;

    // inicializando o valor de soma
    soma = 0;

    // inicializando o contador
    contagem = 1;

    while (contagem <= 5){
        printf("\nDigite o %do. numero: ", contagem);
        scanf("%lf", &val);
        soma = soma + val;
        contagem = contagem + 1;
    }

    printf("\nO resultado da soma eh: %.2f", soma);
    return 0;
}
```

Mantenho a variável `val` para leitura dos dados

Crio a variável `contagem` para funcionar como contador

Mantenho a variável `soma` para atuar com acumulador

Acumulo `soma` a cada passo do loop

Incremento `contagem` a cada passo do loop

# Mudando o problema

---

- ▶ Faça um programa para ler 5 números e **mostrar**, após a leitura de todos os números, **os números lidos juntamente com resultado da soma** desses números
- ▶ Na solução anterior, a cada passo do loop o valor lido era sobrescrito pelo próximo passo

```
while (contagem <= 5){  
    printf("\nDigite o %do. numero: ", contagem);  
    scanf("%lf", &val); // sobrescreve  
    soma = soma + val;  
    contagem = contagem + 1;  
}
```



# Mudando o problema

---

- ▶ Faça um programa para ler 5 números e **mostrar**, após a leitura de todos os números, **os números lidos juntamente com resultado da soma** desses números
- ▶ Solução: armazenar todos os valores em variáveis



# Solução??

---

```
int main()
{
    double val1, val2, val3, val4, val5, soma;

    printf("\nDigite o 1o. numero: ");
    scanf("%lf", &val1);

    printf("\nDigite o 2o. numero: ");
    scanf("%lf", &val2);

    printf("\nDigite o 3o. numero: ");
    scanf("%lf", &val3);

    printf("\nDigite o 4o. numero: ");
    scanf("%lf", &val4);

    printf("\nDigite o 5o. numero: ");
    scanf("%lf", &val5);

    soma = val1 + val2 + val3 + val4 + val5;

    printf("\nValores lidos: %.2f; %.2f; %.2f; %.2f; %.2f", val1, val2, val3, val4, val5);
    printf("\nO resultado da soma eh: %.2f", soma);

    return 0;
}
```

---



# Solução??

---

```
int main()
{
    double val1, val2, val3, val4, val5, soma;

    printf("\nDigite o 1o. numero: ");
    scanf("%lf", &val1);

    printf("\nDigite o 2o. numero: ");
    scanf("%lf", &val2);

    printf("\nDigite o 3o. numero: ");
    scanf("%lf", &val3);

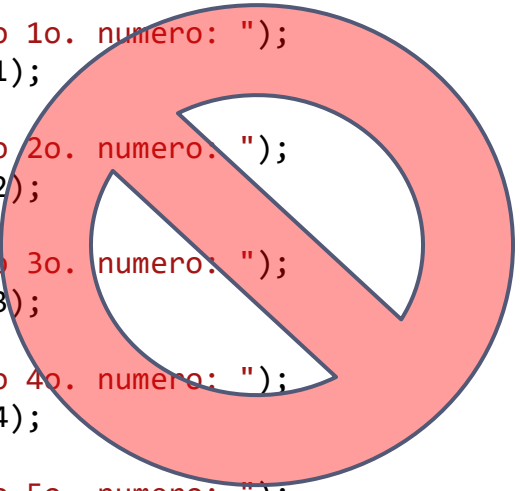
    printf("\nDigite o 4o. numero: ");
    scanf("%lf", &val4);

    printf("\nDigite o 5o. numero: ");
    scanf("%lf", &val5);

    soma = val1 + val2 + val3 + val4 + val5;

    printf("\nValores lidos: %.2f; %.2f; %.2f; %.2f; %.2f", val1, val2, val3, val4, val5);
    printf("\nO resultado da soma eh: %.2f", soma);

    return 0;
}
```



E se for para somar  
100 números?  
Usar 100 variáveis?  
Fica inviável

# Array (Vetor) – Problema 2

---

- ▶ Imagine este outro problema
  - ▶ leia as notas de uma turma de cinco estudantes e depois imprima as notas que são maiores do que a média da turma.
- ▶ Um algoritmo para esse problema poderia ser o mostrado a seguir.





# Array (Vetores) – Solução?

---

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      float n1,n2,n3,n4,n5;
05      printf("Digite a nota de 5 estudantes: ");
06      scanf("%f",&n1);
07      scanf("%f",&n2);
08      scanf("%f",&n3);
09      scanf("%f",&n4);
10      scanf("%f",&n5);
11      float media = (n1+n2+n3+n4+n5)/5.0;
12      if(n1 > media) printf("nota: %f\n",n1);
13      if(n2 > media) printf("nota: %f\n",n2);
14      if(n3 > media) printf("nota: %f\n",n3);
15      if(n4 > media) printf("nota: %f\n",n4);
16      if(n5 > media) printf("nota: %f\n",n5);
17      system("pause");
18      return 0;
19  }
```



# Array

---

- ▶ O algoritmo anterior apresenta uma solução possível.
- ▶ Porém, essa solução é inviável para uma lista de 100 alunos.



# Vetores

---

- ▶ Para 100 alunos, precisamos de:
  - ▶ Uma variável para armazenar a nota de cada aluno
    - ▶ **100 variáveis.**
  - ▶ Um comando de leitura para cada nota
    - ▶ **100 scanf()**
  - ▶ Um somatório de **100 notas.**
  - ▶ Um comando de teste para cada aluno
    - ▶ **100 statements if.**
  - ▶ Um comando de impressão na tela para cada aluno
    - ▶ **100 printf().**



# Por que usar array?

---

- ▶ As variáveis declaradas até agora são capazes de armazenar um único valor por vez.

01	<b>#include</b> <stdio.h>
02	<b>#include</b> <stdlib.h>
03	<b>int</b> main(){
04	<b>float</b> x = 10;
05	printf("x = %f\n",x);
06	x = 20;
07	printf("x = %f\n",x);
08	system("pause");
09	<b>return</b> 0;
10	}

Saída	x = 10.000000
	x = 20.000000



# Array (Vetor)

---

- ▶ Array ou “Vetor” é a forma mais familiar de dados estruturados.
- ▶ Basicamente, um array é um conjunto de componentes do mesmo tipo.



# Vetores

---

- ▶ Solução: utilizar vetores
- ▶ Um vetor (ou *array*) é um conjunto de componentes do mesmo tipo.
  - ▶ Ex: 12 números inteiros, cada um representando um mês do ano; 120 booleanos para indicar o estado de ocupação de quartos de um hotel; 2 números reais para o grau de miopia de uma pessoa; etc..
- ▶ Um vetor é um **tipo de dado** estruturado, isto é, existe uma relação estrutural entre seus valores. Os tipos de dados que vimos até o momento são tipos **elementares** (caracter, real, inteiro, lógico).
  - ▶ Um vetor é formado pela composição por tipos elementares ou de outros tipos estruturados

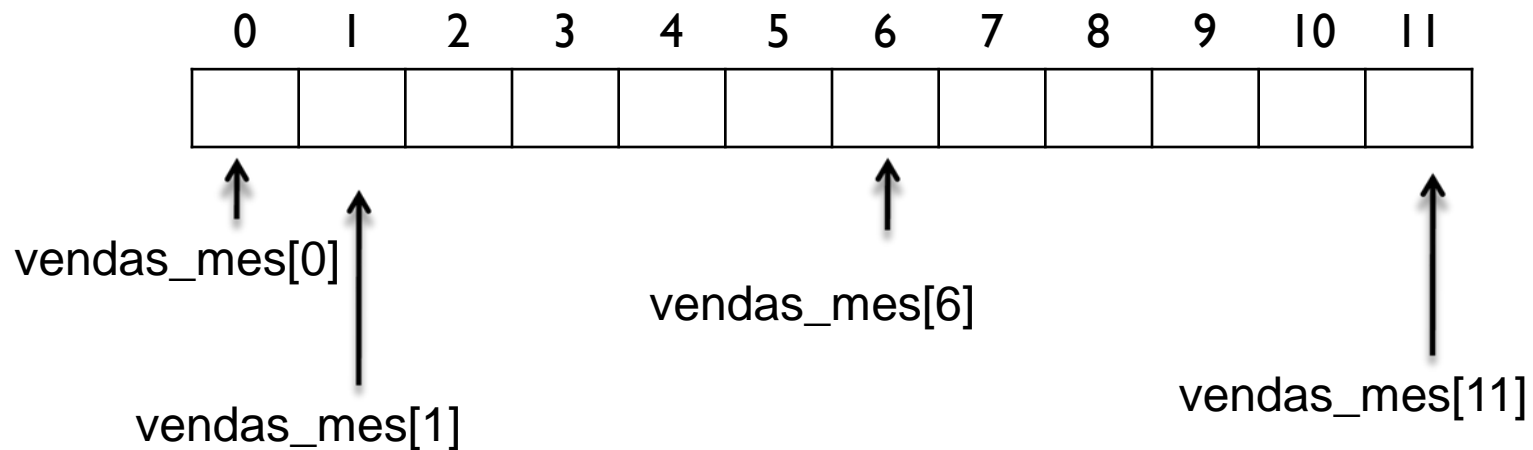


# Vetores

---

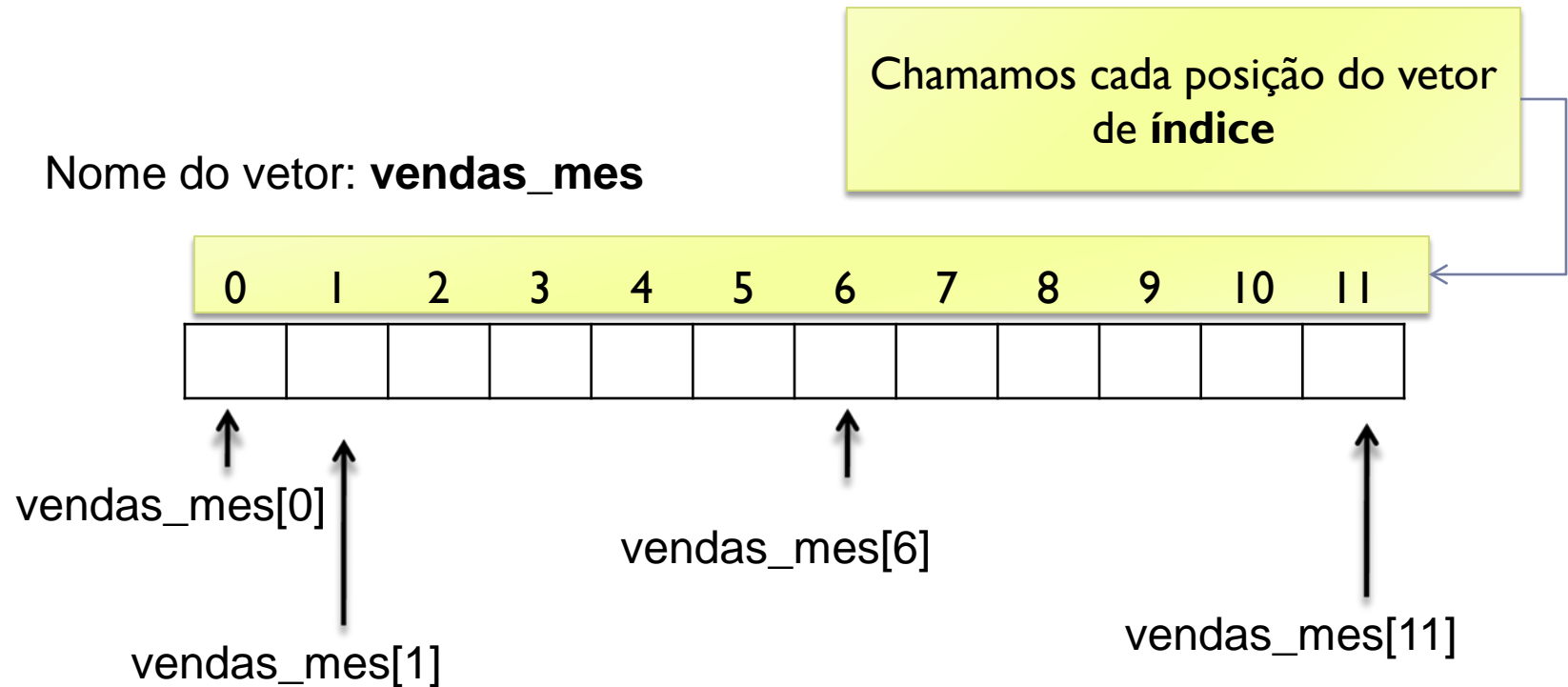
- ▶ Exemplo. Um vetor para as vendas do mês (chamado `vendas_mes`) pode ser ilustrado da seguinte forma

Nome do vetor: **`vendas_mes`**



# Vetores

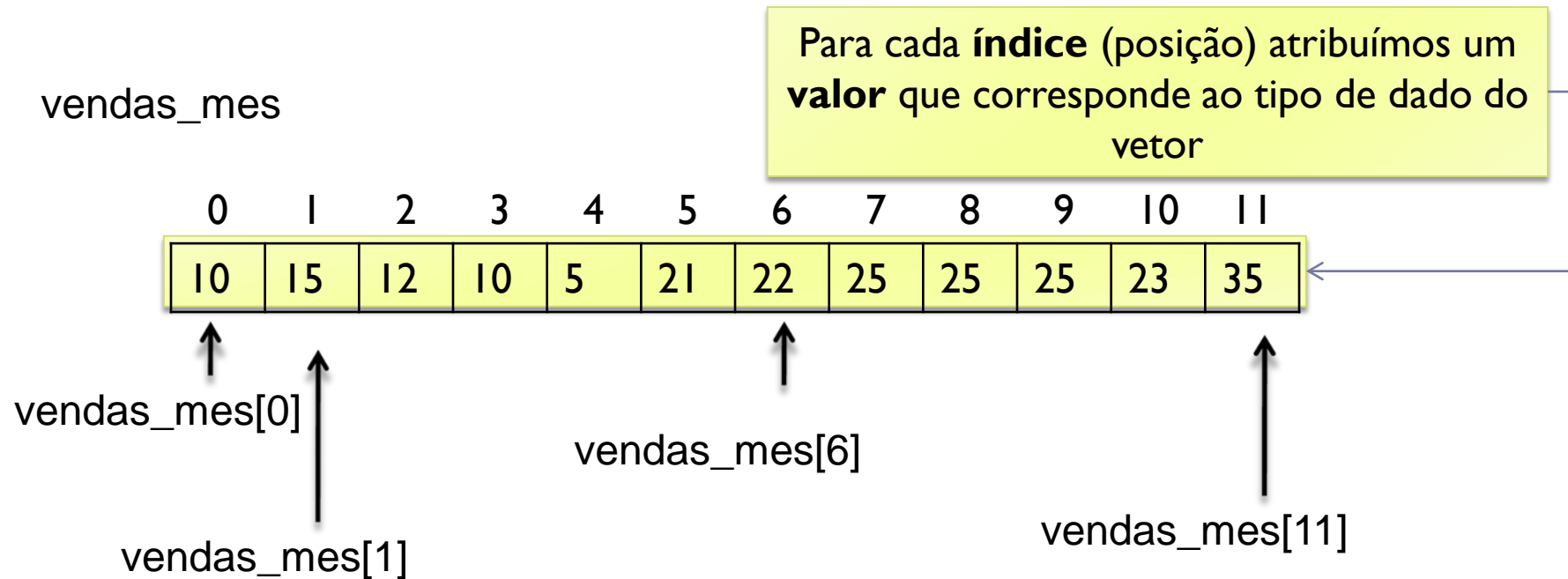
- ▶ Exemplo. Um vetor para as vendas do mês (chamado `vendas_mes`) pode ser ilustrado da seguinte forma





# Vetores

- ▶ Vendas do mês (chamado `vendas_mes`) pode ser ilustrado da seguinte forma, considerando como venda 10, 15, 12, 10, 5, 21, 22, 25, 25, 25, 23, 35 produtos, respectivamente para cada mês



(voltando)

---

- ▶ Faça um programa para ler 5 números e **mostrar**, após a leitura de todos os números, **os números lidos juntamente com resultado da soma** desses números

Solução: armazenar todos os valores em ~~variáveis~~ um array (vetor)



# Vetores

- ▶ Um vetor tem todas as características de uma variável, podendo aparecer em expressões e atribuições.
- ▶ Exemplo

	0	1	2	3	4	5	6	7	8	9	10	11
vendas_mes	10	15	12	10	5	21	22	25	25	25	23	35

Comando	Saída
<code>escreva(vendas_mes[0])</code>	10
<code>escreva(vendas_mes[9])</code>	25
<code>PrimeiroT &lt;- vendas_mes[0] + vendas_mes[1] + vendas_mes[2]</code> <code>Escreva(PrimeiroT)</code>	37
<code>vendas_mes[11] &lt;- vendas_mes[11] - 10</code> <code>Escreva(vendas[11])</code>	25
<code>Leia(vendas_mes[4]) // suponha que o usuário digitou 66</code> <code>Escreva(vendas_mes[4])</code>	66
<code>Leia(vendas_mes[12])</code>	ERRO

# Array – Declaração em C

---

- ▶ Arrays são agrupamentos de dados **adjacentes na memória** (espaço contíguo). Declaração:
  - ▶ `tipo_dado nome_array[tamanho];`
- ▶ O comando acima define um array de nome **nome\_array**, capaz de armazenar **tamanho** elementos adjacentes na memória do tipo **tipo\_dado**
  - ▶ Ex: `double notas[5];`



# Solução??

---

```
int main()
{
    double val[5], soma;

    printf("\nDigite o 1o. numero: ");
    scanf("%lf", &val[0]);

    printf("\nDigite o 2o. numero: ");
    scanf("%lf", &val[1]);

    printf("\nDigite o 3o. numero: ");
    scanf("%lf", &val[2]);

    printf("\nDigite o 4o. numero: ");
    scanf("%lf", &val[3]);

    printf("\nDigite o 5o. numero: ");
    scanf("%lf", &val[4]);

    soma = val[0] + val[1] + val[2] + val[3] + val[4];

    printf("\nValores lidos: %.2f; %.2f; %.2f; %.2f; %.2f", val[0], val[1], val[2], val[3], val[4]);
    printf("\nO resultado da soma eh: %.2f", soma);

    return 0;
}
```

Obs: note que ainda é necessário informar ao scanf o endereço de memória (&) em que o dado lido será armazenado

# Solução??

---

```
int main()
{
    double val[5], soma;

    printf("\nDigite o 1o. numero: ");
    scanf("%lf", &val[0]);

    printf("\nDigite o 2o. numero: ");
    scanf("%lf", &val[1]);

    printf("\nDigite o 3o. numero: ");
    scanf("%lf", &val[2]);

    printf("\nDigite o 4o. numero: ");
    scanf("%lf", &val[3]);

    printf("\nDigite o 5o. numero: ");
    scanf("%lf", &val[4]);

    soma = val[0] + val[1] + val[2] + val[3] + val[4];

    printf("\nValores lidos: %.2f; %.2f; %.2f; %.2f; %.2f", val[0], val[1], val[2], val[3], val[4]);
    printf("\nO resultado da soma eh: %.2f", soma);

    return 0;
}
```

E se for para somar  
100 números?  
Estamos usando uma só variável, (um  
vetor de double), mas ainda temos  
que indexar cada posição  
separadamente



# Solução

---

- ▶ Utilize loops e indexe o vetor com o contador do loop



## ► Relembrando

```
double val[5], soma;  
int contagem;
```

Crio o vetor `val[5]` para leitura de 5 double

```
// inicializando o valor de soma  
soma = 0;
```

```
// iniciando o contador  
contagem = 0;
```

```
while (contagem < 5){  
    printf("\nDigite o %do. numero: ", contagem + 1);  
    scanf("%lf", &val[contagem]);  
    soma = soma + val[contagem];  
    contagem = contagem + 1;  
}
```

Leio o valor em cada posição do vetor  
`val[contagem]`

```
contagem = 0;
```

----- continua no próximo slide



## ► Relembrando

```
contagem = 0;
printf("\nValores digitados: ");
while (contagem < 5){
    printf("%f; ", val[contagem]);
    contagem = contagem + 1;
}
printf("\nO resultado da soma eh: %.2f", soma);
return 0;
```

Imprime o valor para cada posição `val[contagem]`

# Solução

- ▶ É melhor utilizar loop **for**, pois o número de iterações é conhecido

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      double val[5], soma;
7      int i;
8
9      // inicializando o valor de soma
10     soma = 0;
11     for (i = 0; i < 5; i++) {
12         printf("\nDigite o %do. numero: ", i+1);
13         scanf("%lf", &val[i]);
14         soma = soma + val[i];
15     }
16
17     printf("\nValores digitados: ");
18     for (i = 0; i < 5; i++)
19         printf("%f; ", val[i]);
20
21     printf("\nO resultado da soma eh: %.2f", soma);
22     return 0;
23 }
```

# Solução

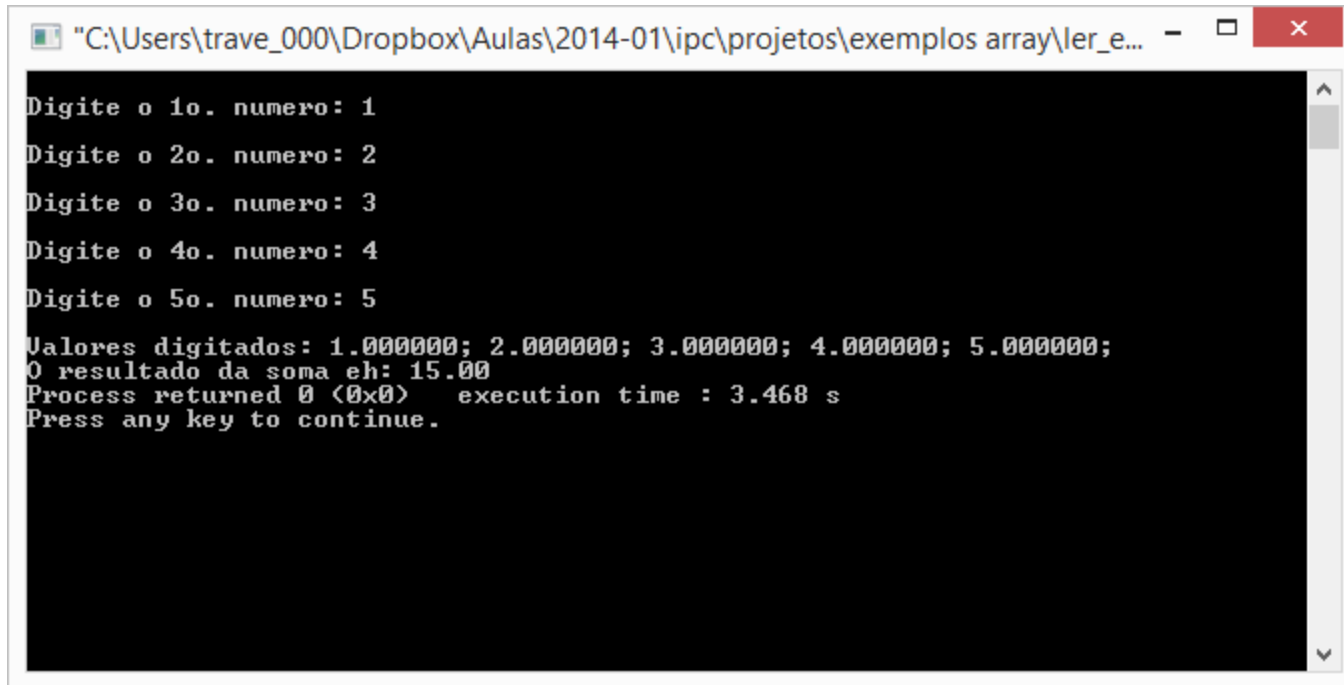
- ▶ É melhor utilizar loop **for**, pois o número de iterações é conhecido

- ▶ E se for para somar 100 números?

```
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main()
5  {
6      double val[100], soma;
7      int i;
8
9      // inicializando o valor de soma
10     soma = 0;
11     for (i = 0; i < 100; i++) {
12         printf("\nDigite o %do. numero: ", i+1);
13         scanf("%lf", &val[i]);
14         soma = soma + val[i];
15     }
16
17     printf("\nValores digitados: ");
18     for (i = 0; i < 100; i++)
19         printf("%f; ", val[i]);
20
21     printf("\nO resultado da soma eh: %.2f", soma);
22     return 0;
23 }
```

# Resultado

---



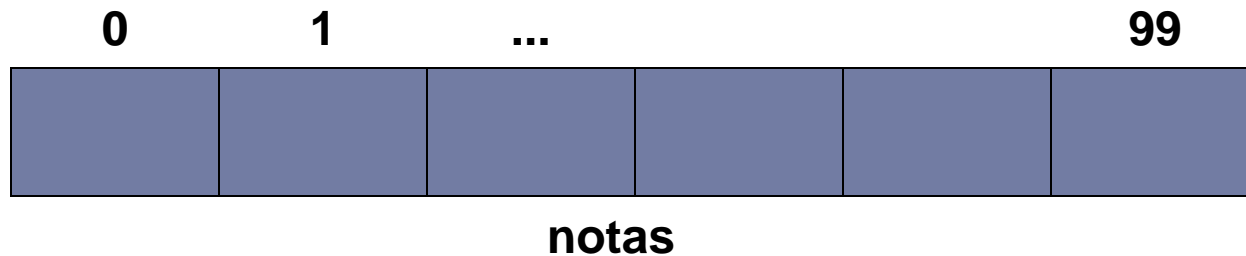
```
"C:\Users\trave_000\Dropbox\Aulas\2014-01\ipc\projetos\exemplos array\ler_e... - □ ×  
Digite o 1o. numero: 1  
Digite o 2o. numero: 2  
Digite o 3o. numero: 3  
Digite o 4o. numero: 4  
Digite o 5o. numero: 5  
Valores digitados: 1.000000; 2.000000; 3.000000; 4.000000; 5.000000;  
O resultado da soma eh: 15.00  
Process returned 0 (0x0)   execution time : 3.468 s  
Press any key to continue.
```



# Array - Definição

---

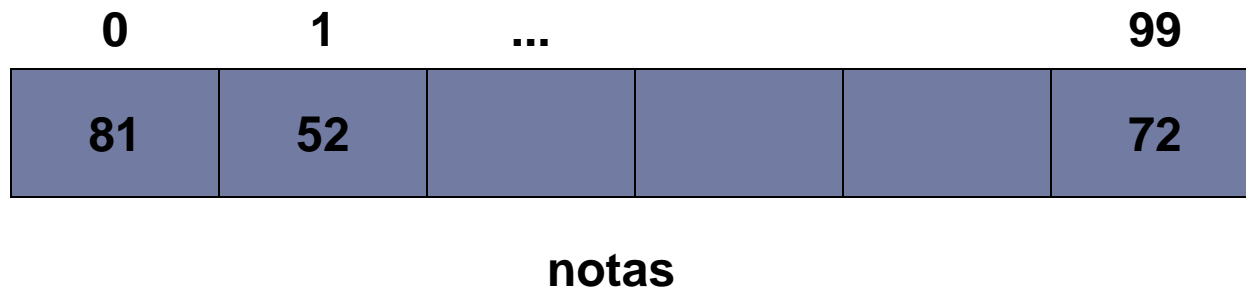
- ▶ As variáveis têm relação entre si
  - ▶ **todas** armazenam **notas** de alunos
- ▶ Podemos declará-las usando um **ÚNICO nome** para todos os 100 alunos
  - ▶ notas = conjunto de 100 números acessados por um índice = array.



# Array - Definição

---

- ▶ Na linguagem C a numeração do array começa sempre do zero.
- ▶ Isto significa que, no exemplo anterior, os dados serão indexados de 0 a 99.
  - ▶ `notas[0], notas[1], ..., notas[99]`



# Vetores – Índice Inválido

---

## ► Observação

- Se o usuário digitar mais de 100 elementos em um array de 100 elementos, o programa tentará ler normalmente.
- Porém, o programa os armazenará em uma parte não alocada de memória, pois o espaço alocado foi para somente 100 elementos.
- Isto pode resultar nos mais variados erros no instante da execução do programa.



# Array = variável

---

- ▶ Cada elemento do array tem todas as características de uma variável e pode aparecer em expressões e atribuições.

- ▶ `notas[2] = notas[3] + notas [20]`

- ▶ Ex: somar todos os elementos de notas:

```
int soma = 0;
```

```
for(i=0;i < 100;i++)
```

```
    soma = soma + notas[i];
```





# Array - Características

---

- ▶ **Características básicas de um Array**
  - ▶ Estrutura homogênea, isto é, formada de elementos do mesmo tipo.
  - ▶ todos os elementos da estrutura são igualmente acessíveis, isto é, o tempo e o tipo de procedimento para acessar qualquer um dos elementos do array são **iguais**.
  - ▶ cada elemento componente desta estrutura tem um índice próprio segundo sua posição no conjunto



# Exercício

---

## ▶ Exercício

- ▶ Para um array  $A$  com 5 números inteiros, formular um algoritmo que determine o maior elemento deste array.



# Exercício

---

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    // declarando o vetor e inicializando seus elementos
    int A[5] = {3,18,2,51,45};
    int N = 5,i;

    int Maior = A[0];

    for(i = 1; i < N; i++){
        if (Maior < A[i])
            Maior = A[i];
    }
    printf("O maior valor eh: %d", Maior);
    return 0;
}
```

---



# Observação sobre a memória

---

- ▶ As variáveis que declaramos no nosso código ocupam determinada quantidade de bytes na memória.
  - ▶ **char** *c*; 1 byte
  - ▶ **int** *a*; 4 bytes
  - ▶ **double** *d*; 8 bytes
- ▶ E os vetores, quantos bytes ocupam?
  - ▶ **char** *c*[10];
  - ▶ **int** *a*[15];
  - ▶ **double** *d*[2];



# Observação sobre a memória

---

- ▶ As variáveis que declaramos no nosso código ocupam determinada quantidade de bytes na memória.
  - ▶ **char** `c`; 1 byte
  - ▶ **int** `a`; 4 bytes
  - ▶ **double** `d`; 8 bytes
- ▶ E os vetores, quantos bytes ocupam?
  - ▶ **char** `c[10]`;  $10 \times 1 = 10$  bytes
  - ▶ **int** `a[15]`;  $15 \times 4 = 60$  bytes
  - ▶ **double** `d[2]`;  $2 \times 8 = 16$  bytes



# Observação sobre a memória

---

- ▶ **char** `c[10]`;  $10 \times 1 = 10$  bytes
- ▶ **int** `a[15]`;  $15 \times 4 = 60$  bytes
- ▶ **double** `d[2]`;  $2 \times 8 = 16$  bytes
  
- ▶ As regiões de memória alocadas para os vetores são contínuas, ou seja, os endereços consecutivos de cada índice dos vetores serão ‘vizinhos’
  
- ▶ Podemos imaginar a memória como uma sequência de linear blocos de 1 byte



# Observações sobre a memória

---

Endereço	Blocos	Tamanho
1		(1 byte)
2		(1 byte)
3		(1 byte)
4		(1 byte)
5		(1 byte)
6		(1 byte)
7		(1 byte)
8		(1 byte)
9		(1 byte)
10		(1 byte)
11		(1 byte)
12		(1 byte)
13		(1 byte)
14		(1 byte)
....		



# Observações sobre a memória

---

	Endereço	Blocos	Variável	tipo
<b>char</b> <i>c</i> ;	1		c	char
	2			
	3			
	4			
	5			
	6			
	7			
	8			
	9			
	10			
	11			
	12			
	13			
	14			
	....			

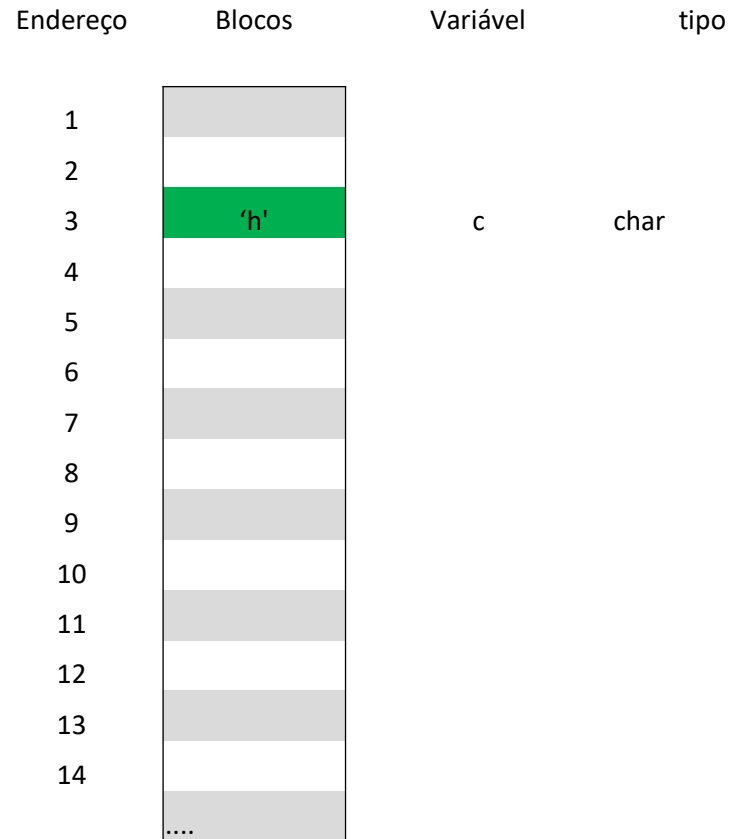




# Observações sobre a memória

---

```
char c;  
c = 'h';
```

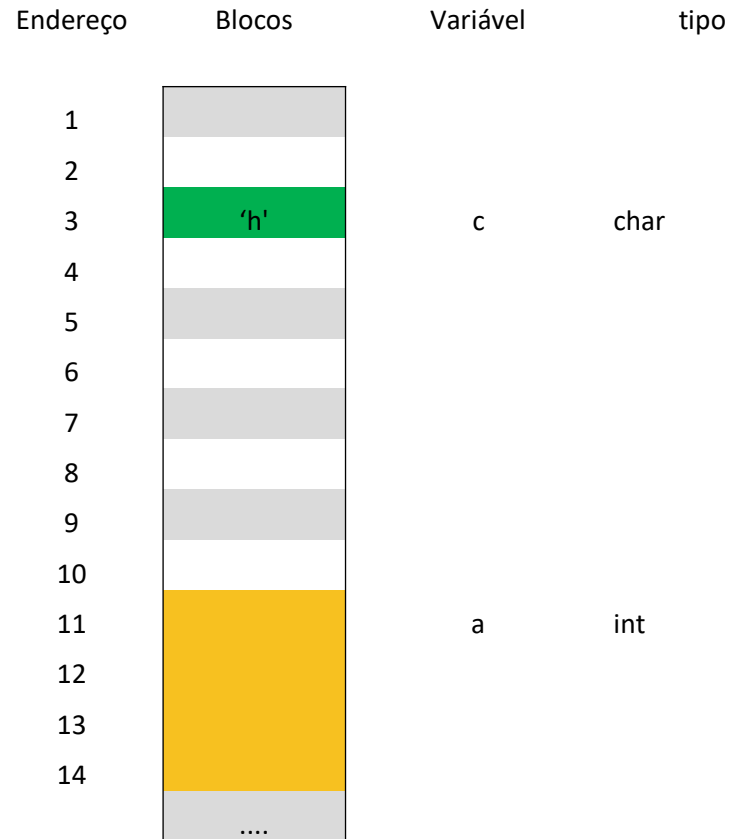


# Observações sobre a memória

---

```
char c;  
c = 'h';
```

```
int a;
```

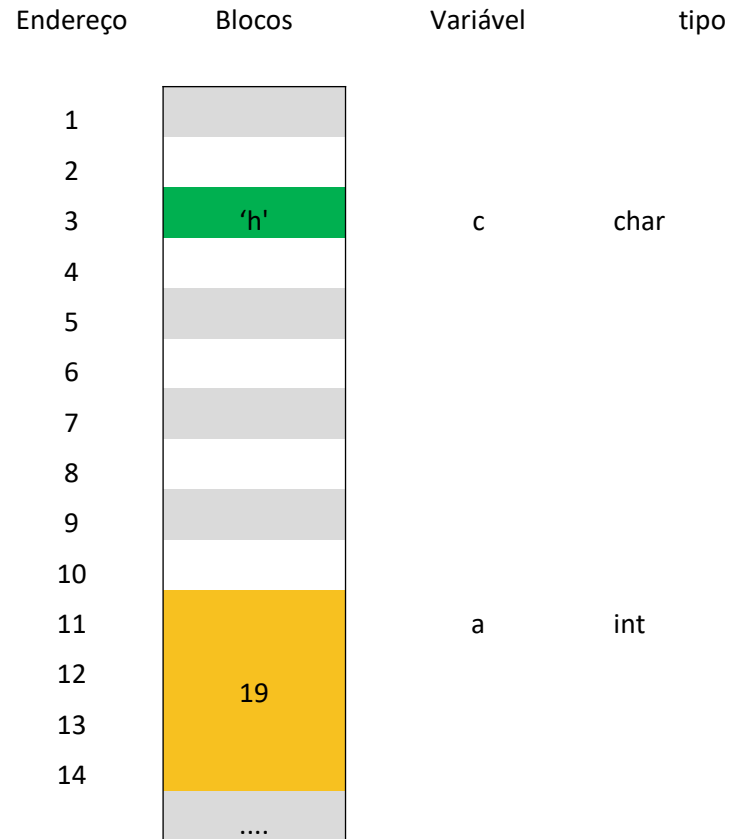


# Observações sobre a memória

---

```
char c;  
c = 'h';
```

```
int a;  
a = 19;
```

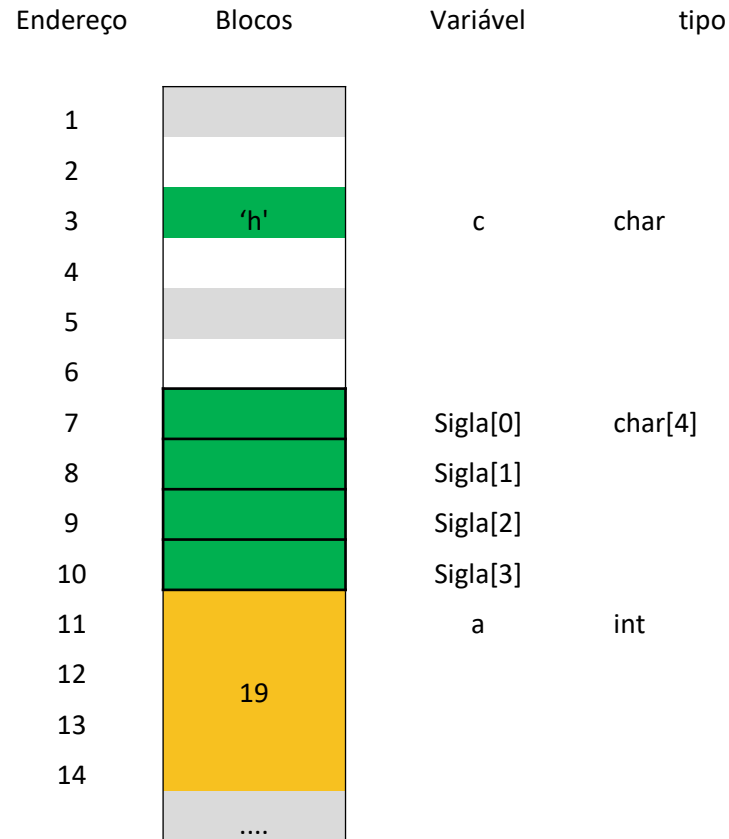


# Observações sobre a memória

```
char c;  
c = 'h';
```

```
int a;  
a = 19;
```

```
char Sigla[4];
```



# Observações sobre a memória

```
char c;  
c = 'h';
```

```
int a;  
a = 19;
```

```
char Sigla[4];  
Sigla[0] = 'U';  
Sigla[1] = 'F';  
Sigla[2] = 'U';  
Sigla[3] = '\0';
```

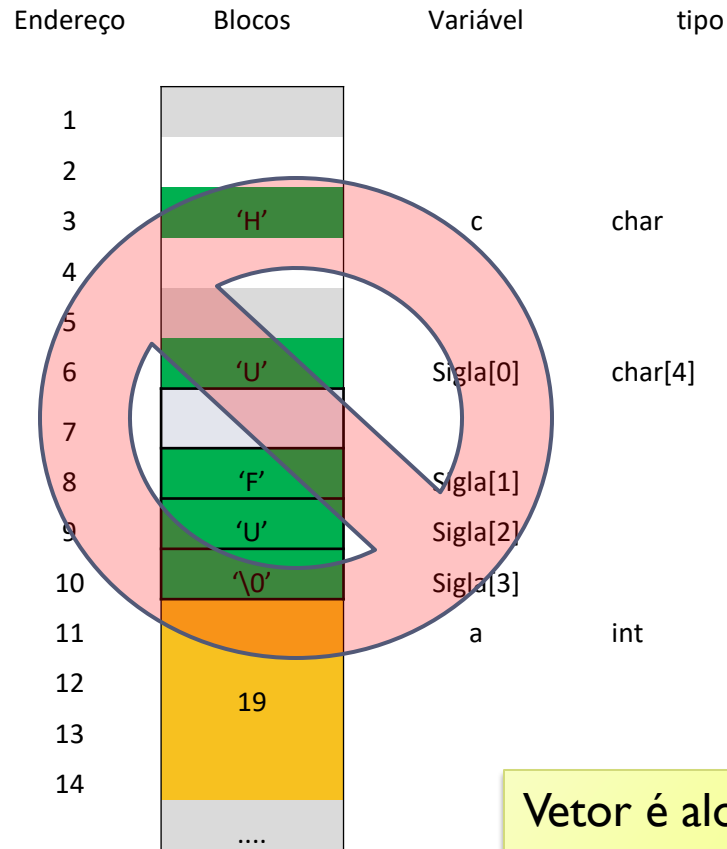
Endereço	Blocos	Variável	tipo
1			
2			
3	'H'	c	char
4			
5			
6			
7	'U'	Sigla[0]	char[4]
8	'F'	Sigla[1]	
9	'U'	Sigla[2]	
10	'\0'	Sigla[3]	
11	19	a	int
12			
13			
14			
	....		

# Observações sobre a memória

```
char c;  
c = 'h';
```

```
int a;  
a = 19;
```

```
char Sigla[4];  
Sigla[0] = 'U';  
Sigla[1] = 'F';  
Sigla[2] = 'U';  
Sigla[3] = '\0';
```



Vetor é alocado em blocos contínuos de memória

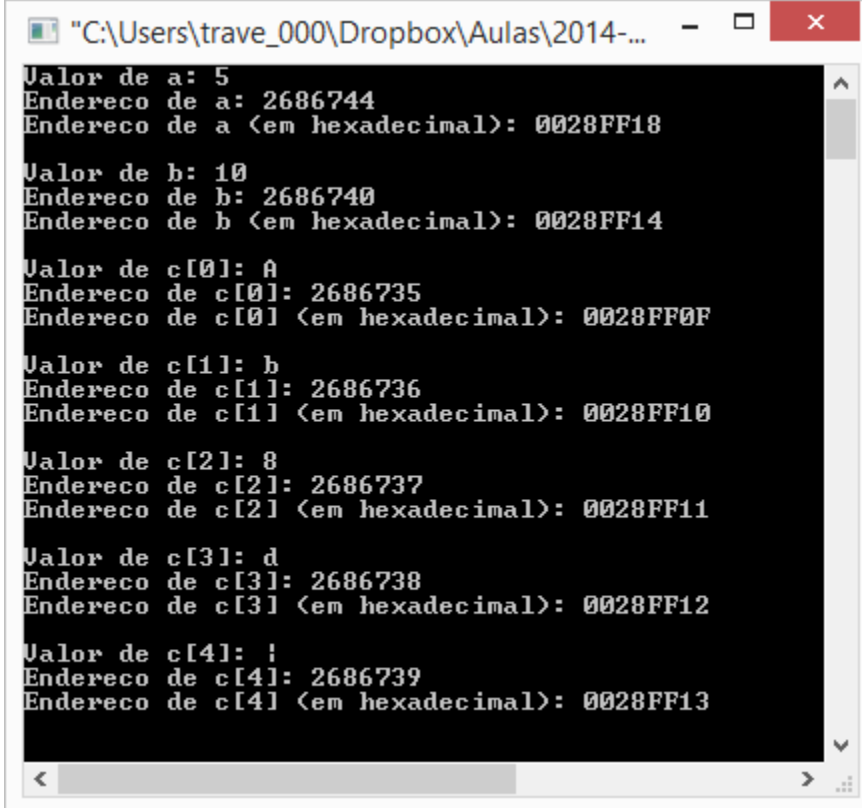
# Endereço de variáveis

- Para descobrir o endereço de uma variável em C, use o operador &

```
6     int i;
7     int a = 5;
8     int b = 10;
9     char c[5] = {'A', 'b', '8', 'd', '|'};
12    printf("Valor de a: %d \n", a);
13    printf("Endereco de a: %d \n", &a);
14    printf("Endereco de a (em hexadecimal): %p \n\n", &a);
15
16    printf("Valor de b: %d \n", b);
17    printf("Endereco de b: %d \n", &b);
18    printf("Endereco de b (em hexadecimal): %p \n\n", &b);
19
20    for (i=0; i < 5; i++){
21        printf("Valor de c[%d]: %c \n", i, c[i]);
22        printf("Endereco de c[%d]: %d \n", i, &c[i]);
23        printf("Endereco de c[%d] (em hexadecimal): %p \n\n", i, &c[i]);
24    }
```

# Endereço de variáveis

- ▶ Para descobrir o endereço de uma variável em C, use o operador &



```
"C:\Users\trave_000\Dropbox\Aulas\2014-... - [X]
Valor de a: 5
Endereco de a: 2686744
Endereco de a <em hexadecimal>: 0028FF18

Valor de b: 10
Endereco de b: 2686740
Endereco de b <em hexadecimal>: 0028FF14

Valor de c[0]: A
Endereco de c[0]: 2686735
Endereco de c[0] <em hexadecimal>: 0028FF0F

Valor de c[1]: b
Endereco de c[1]: 2686736
Endereco de c[1] <em hexadecimal>: 0028FF10

Valor de c[2]: 8
Endereco de c[2]: 2686737
Endereco de c[2] <em hexadecimal>: 0028FF11

Valor de c[3]: d
Endereco de c[3]: 2686738
Endereco de c[3] <em hexadecimal>: 0028FF12

Valor de c[4]: !
Endereco de c[4]: 2686739
Endereco de c[4] <em hexadecimal>: 0028FF13
```



# Arrays bidimensionais - matrizes

---

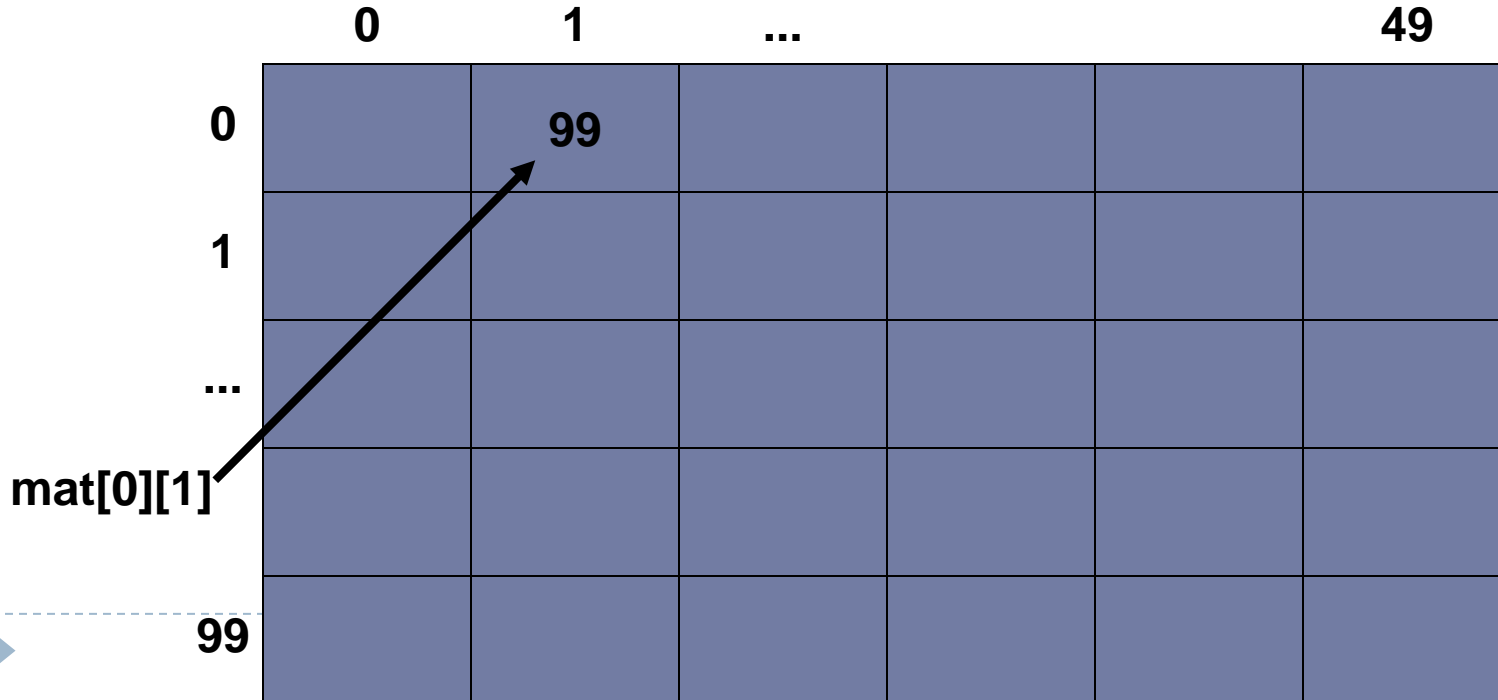
- ▶ Também chamados de “matrizes”, contém:
  - ▶ arranjados na forma de uma tabela de 2 dimensões;
  - ▶ necessita de dois índices para acessar uma posição: um para a linha e outro para a coluna
  - ▶ Índices começam sempre na posição ZERO.
- ▶ Declaração
  - ▶ `tipo_variável nome_variável[linhas][colunas];`



# Arrays bidimensionais - matrizes

---

- ▶ Ex.: um array que tenha 100 linhas por 50 colunas
  - ▶ `int mat[100][50];`
  - ▶ `mat[0][1] = 99;`



# Arrays bidimensionais - matrizes

---

- ▶ Como uma matriz possui dois índices, precisamos de dois comandos de repetição para percorrer todos os seus elementos.



# Arrays bidimensionais - matrizes

---

```
01  #include <stdio.h>
02  #include <stdlib.h>
03  int main(){
04      int mat[100][50];
05      int i,j;
06      for (i = 0; i < 100; i++){
07          for (j = 0; j < 50; j++){
08              printf("Digite o valor de mat[%d][%d]: ",i,j);
09              scanf("%d",&mat[i][j]);
10          }
11      }
12      system("pause");
13      return 0;
14  }
```



# Arrays Multidimensionais

---

- ▶ Arrays podem ter diversas dimensões, cada uma identificada por um par de colchetes na declaração
  - ▶ `int vet[5];` // 1 dimensão
  - ▶ `float mat[5][5];` // 2 dimensões
  - ▶ `double cub[5][5][5];` // 3 dimensões
  - ▶ `int X[5][5][5][5];` // 4 dimensões

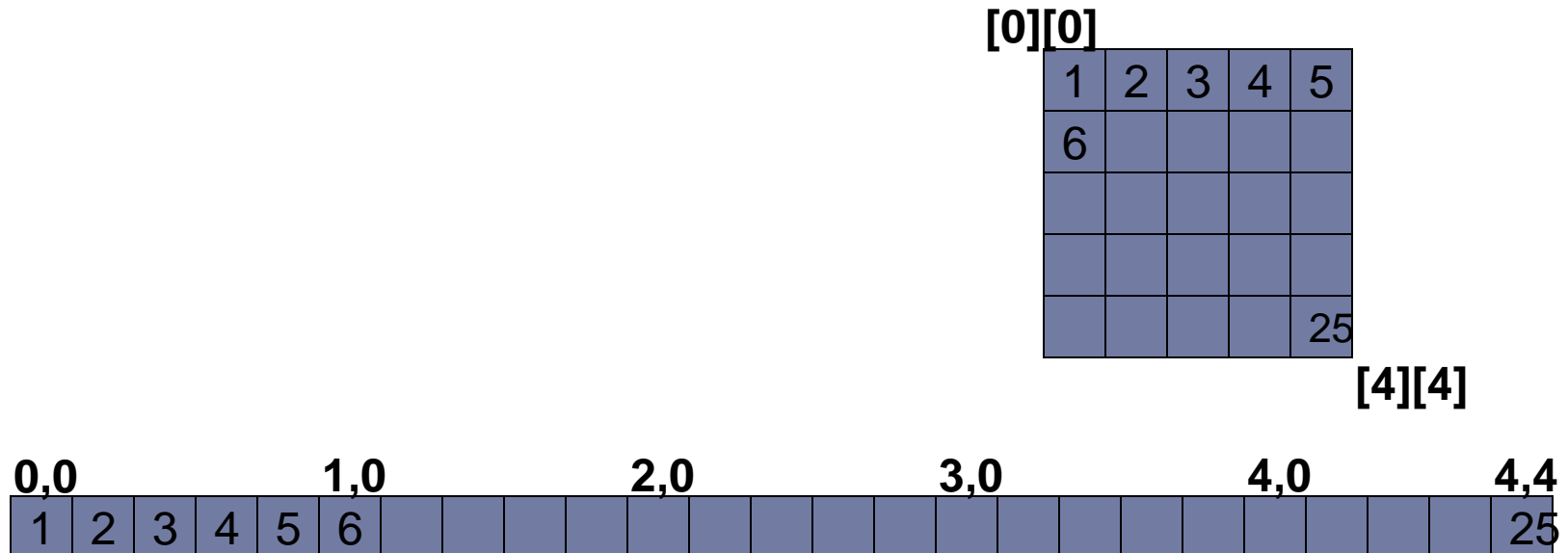


# Arrays Multidimensionais

---

- ▶ Apesar de terem o comportamento de estruturas com mais de uma dimensão, na memória os dados são armazenados linearmente:

- ▶ `int mat[5][5];`



# Arrays Multidimensionais

---

- ▶ Um array N-dimensional funciona basicamente como outros tipos de array. Basta lembrar que o índice que varia mais rapidamente é o índice mais à direita.
  - ▶ `int vet[5];` // 1 dimensão
  - ▶ `float mat[5][5];` // 2 dimensões
  - ▶ `double cub[5][5][5];` // 3 dimensões
  - ▶ `int X[5][5][5][5];` // 4 dimensões



# Exercício

---

- ▶ Dado um array A de 3x5 elementos inteiros, calcular a soma dos seus elementos.
- ▶ Exemplo

1	5	0	0	3
2	3	7	0	0
0	0	2	1	2

- ▶ Soma =  $1 + 5 + 0 + 0 + 3 + 2 + 3 + 7 + 0 + 0 + 0 + 0 + 2 + 1 + 2 = 26$





# Exercício

---

```
int soma = 0;
int A[3][5];
int i,j;

for(i=0;i<3;i++){
    for(j=0;j<5;j++){
        soma = soma + A[i][j];
    }
}
printf("%d", soma);
```



# Exercício

---

- ▶ Dado duas matrizes reais de dimensão 2x3, fazer um programa para calcular a soma delas.
  - ▶ Exemplo de como é a soma de duas matrizes

$$\begin{bmatrix} 1 & 3 & 2 \\ 1 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 5 \\ 7 & 5 & 0 \end{bmatrix} = \begin{bmatrix} 1+0 & 3+0 & 2+5 \\ 1+7 & 0+5 & 0+0 \end{bmatrix} = \begin{bmatrix} 1 & 3 & 7 \\ 8 & 5 & 0 \end{bmatrix}$$



# Exercício

---

```
float A[2][3], B[2][3], Soma[2][3];  
int i,j;
```

```
// << suponha comandos de leitura de A e B aqui >>
```

```
for(i=0;i<2;i++){  
    for(j=0;j<3;j++){  
        Soma[i][j] = A[i][j] + B[i][j];  
    }  
}
```



# Inicialização

---

- ▶ Arrays podem ser inicializados com certos valores durante sua declaração. A forma geral de um array com inicialização é:
  - ▶ `tipo_da_variável nome_da_variável [tam1][tam2] ... [tamN] = {lista_de_valores};`



# Inicialização

---

- ▶ A lista de valores é composta por valores (do mesmo tipo da variável) separados por vírgula. Os valores devem ser dados na ordem em que serão colocados na matriz.

```
float vect[6] = { 1.3, 4.5, 2.7, 4.1, 0.0, 100.1 };
```

```
int mat[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
```

```
int mat[3][4] = { {1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} };
```

```
char str[10] = { 'j', 'o', 'a', 'o', '\0' };
```

```
char str[10] = "Joao";
```

```
char nomes[3][10] = { "Joao", "Maria", "Jose" };
```



---

► `int mat[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };`

```
#include <stdio.h>
#include <stdlib.h>
```

```
int main()
```

```
{
```

```
    int mat[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
```

```
    int i,j;
```

```
    for (i = 0; i < 3; i++){
```

```
        for (j = 0; j < 4; j++){
```

```
            printf("%d\t",mat[i][j]);
```

```
        }
```

```
        printf("\n");
```

```
    }
```

```
    return 0;
```

```
}
```

"D:\Dropbox\Aulas\2014-01\ipc\projetos\exemplos array\ordem\_matriz\bi

1	2	3	4
5	6	7	8
9	10	11	12

Process returned 0 (0x0) execution time : 0.359 s  
Press any key to continue.

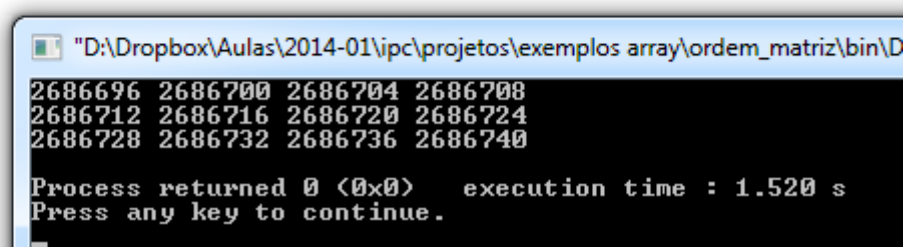
## ► Observe os endereços das variáveis

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int mat[3][4] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
    int i,j;

    for (i = 0; i < 3; i++){
        for (j = 0; j < 4; j++){
            printf("%d\t", &mat[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```



```
"D:\Dropbox\Aulas\2014-01\ipc\projetos\exemplos array\ordem_matriz\bin\D
2686696 2686700 2686704 2686708
2686712 2686716 2686720 2686724
2686728 2686732 2686736 2686740

Process returned 0 (0x0) execution time : 1.520 s
Press any key to continue.
```

# Inicialização sem tamanho

---

- ▶ Inicialização sem especificação de tamanho
  - ▶ **char mess[ ] = "Linguagem C: flexibilidade e poder.";**  
//A string mess terá tamanho 36.
  - ▶ **int matrix[ ][2] = { 1,2,2,4,3,6,4,8,5,10 };** //O número de linhas de matrix será 5.
    - ▶ 1 2
    - ▶ 2 4
    - ▶ 3 6
    - ▶ 4 8
    - ▶ 5 10





# Inicialização sem tamanho

---

- ▶ Nesse tipo de inicialização, o compilador C vai considerar o tamanho do dado declarado como sendo o tamanho do array.
- ▶ Isto ocorre durante a compilação e não poderá mais ser mudado durante o programa.
- ▶ Isto é útil quando não queremos contar quantos caracteres serão necessários para inicializarmos uma string.



# Referências

---

- ▶ Slides: alguns slides foram baseados em slides do Prof. André Backes

