

Vetores e Matrizes em C

Vetores e Matrizes:

- Veremos como declarar cada um e
- como acessar seus elementos.
- Estudaremos também sua estrutura em memória.

String:

- São vetores de caracteres.
- Vamos ver como manipula-los.
- Veremos as funções de string.h.

Objetivos:

- Saber como declarar e acessar elementos de vetores e matrizes.
- Estudar os conceitos de alocação de memória.

- Um vetor ou uma matriz é uma coleção de variáveis do mesmo tipo que é referenciada por um nome em comum.
 - ▶ No caso de vetor (*array*), é uma coleção unidimensional.
 - São listas ordenadas de determinados tipos de dados.
 - - Iniciam com índice 0 (primeiro elemento do vetor) e vai até o último elemento declarado na variável.
 - ▶ No caso de matriz, é uma coleção bidimensional.

Vetores/Matrizes e ponteiro estão intimamente relacionados.

- ▶ Veremos em aulas futuras como trabalhar com ponteiros em geral.
- ▶ Aqui, veremos apenas como trabalhar com ponteiros sobre vetores e matrizes.

Declaração de um vetor:

<tipo> <nome do vetor>[<tamanho do vetor>;

tipo nome[tamanho];

Exemplo:

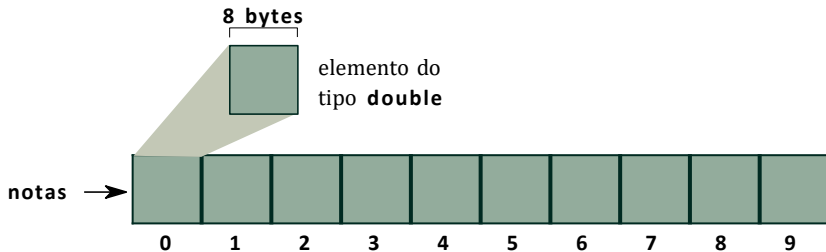
int vetor[10]; float nota[20];

- Assim como as variáveis primitivas, a declaração permite que o compilador **aloque memória** para guardar os dados.
- Os dados são todos do mesmo *tipo*.
 - ▶ Que deve ser especificado.
- A ele, atribui-se um *nome*.
- E, entre colchetes, deve-se especificar um tamanho.
 - ▶ Um valor inteiro que indica quantos elementos o vetor terá.

Declaração de Vetores (II)

- Um vetor é um conjunto de elementos de mesmo tipo **alocado contigualmente em memória**.
- Exemplo (guardar a nota de 10 alunos):

```
double notas[10];
```
- Neste exemplo, foi alocado um vetor de nome *notas*.
- Cada elemento tem o tipo `double` e são 10 deles.
- Se um `double` tem 8 bytes, seu tamanho total, em bytes é: $10 * 8 = 80$ bytes.



Assim como as demais variáveis, estes tipos de estrutura também podem ser inicializados na declaração:

Exemplo:

```
int VETOR [5] = {5, 10, 15, 20, 25};
```

Quando a estrutura é inicializada na declaração, o tamanho de uma das dimensões pode ser omitida, pois o mesmo será considerado a partir da quantidade de valores descritos entre chaves.

Assim, a declaração acima poderia ser reescrita da seguinte forma:

```
int VETOR [] = {5, 10, 15, 20, 25};
```

- Existe um operador em C que calcula, em tempo de compilação, o número de bytes de um determinado tipo.

- ▶ **sizeof**(*tipo*)

Para calcularmos o tamanho em memória, em bytes, de um vetor:

total em bytes = **sizeof**(*tipo*) * tamanho do vetor

- Cada elemento tem uma posição dentro do vetor.
 - ▶ Esta posição equivale a da memória.
 - ▶ A posição sempre varia de 0 até $n - 1$, onde n é o *tamanho*.
 - ▶ Na memória, em bytes:
 - F a posição 0 vai de 0 a $\text{sizeof}(\text{tipo}) - 1$,
 - F a posição 1 vai de $\text{sizeof}(\text{tipo})$ até $2 * \text{sizeof}(\text{tipo}) - 1$,
 - F a posição 2 vai de $2 * \text{sizeof}(\text{tipo})$ até $3 * \text{sizeof}(\text{tipo}) - 1$,
 - F a posição i vai de $i * \text{sizeof}(\text{tipo})$ até $(i + 1) * \text{sizeof}(\text{tipo}) - 1$,
 - F a posição $n - 1$ vai de $(n - 1) * \text{sizeof}(\text{tipo})$ até $n * \text{sizeof}(\text{tipo}) - 1$.

- Um elemento de um vetor é acessado por um 'índice'.
 - Que corresponde a posição de memória.
- Mas podemos acessá-lo de forma mais simples:
 - Para acessar a posição 0, usamos nome `vetor[0]`;
 - Para acessar a posição 1, usamos nome `vetor[1]`;
 - Para acessar a posição *i*, usamos nome `vetor[i]`;
 - Para acessar a posição *n-1*, usamos nome `vetor[n-1]`.
- Exemplo:
 - Uma atribuição de um valor para a variável `notas` pode ser feita por:

```
notas[0] = 7.4;  
notas[1] = 8.2;
```
 - A obtenção de um valor da variável `notas` pode ser feita por:

```
printf("Nota 0: %.2f", notas[0]);  
printf("Nota 1: %.2f", notas[1]);
```


- Ler as notas de 10 alunos, calcular a média e indicar quantas delas estão acima da média.

```
1  #include <stdio.h>
2  #include <conio.h>
3
4  int main() { //A palavra "int" indica que a função "main" retornará um valor do tipo inteiro
5              (geralmente o (zero) se o programa foi executado com sucesso).
6
7      int vet[10];
8
9      int i, n;
10
11     for(i=0; i < 10; i++) {
12
13         printf("Entre com o numero: ");
14
15         scanf("%d", &vet[i]);
16
17     }
18     return(0);
19 }
20
21
```

- Ler as notas de 10 alunos, calcular a média e indicar quantas delas estão acima da média.

```
1  #include <stdio.h>
2
3  int main(){
4      double notas[10], soma, media;
5      int i, contador; //contador contará quantas notas estão acima da média
6      soma = 0;        // i é para os loops
7      for(i=0; i<10; i++){
8          printf("Entre com a nota %d: ", i);
9          scanf("%lf", &notas[i]);
10         soma = soma + notas[i];          //Adiciona o valor da nota atual à variável soma.
11     }
12     media = soma/10; //Calcula a média das 10 notas
13     contador = 0;
14     for(i=0; i<10; i++){ // percorre os 10 elementos o array notas
15         if(notas[i] > media) //Verifica se a nota na posição i é maior que a média.
16             contador++; //incrementa o contador em 1 se nota for maior que a média
17     }
18     printf("Existem %d notas acima da media\n", contador);
19     return(0);
20 }
21
```

- Quando um vetor é declarado, não necessariamente ele será inicializado com 0.
 - ▶ Não conte com isso!
- Existem algumas maneiras de inicializar um vetor:
 - ▶ atribuição (ou leitura) e
 - ▶ inicialização direta.
- Inicialização direta:

Inicialização de vetor:

```
tipo nome[tamanho] = {lista_valores};
```

- A lista de valores é separada por vírgula.
- Exemplo:

```
char vogais[5] = {'a', 'e', 'i', 'o', 'u'};
```

- Os **limites** de acesso são de 0 a $n - 1$, onde n é o **tamanho**.
- C não verifica os limites de um vetor.
 - Se ultrapassarmos os limites do vetor, ele irá acessar **lixo**.
 - As consequências disso são:
 - F ou obter informação inválida,
 - F ou modificar posição de memória inválida.

Importante:

- ▶ **Ultrapassar os limites de um vetor em C é preocupante!**
- ▶ Pode causar **erros de execução** em todo o seu programa!
- ▶ Causa de boa parte dos erros em programas em C.
- ▶ **Cheque sempre os limites** de acesso e tenha certeza que não irá ultrapassar.

- Este exemplo ultrapassa (escrevendo) os limites de um vetor:

```
1  #include <stdio.h>
2
3  int main(){
4      int count[10], i;
5
6      for(i=0; i<15; i++){
7          count[i] = i;
8      }
9      return(0);
10 }
```

- O compilador irá gerar o executável, como se estivesse certo.
 - ▶ Ele não checa os limites!
 - ▶ O programador é quem deve checar!
- Ao executar, é provável que ocorrerá um erro de execução.
 - ▶ A chamada (e temida!) **falha de segmentação**.
 - ▶ *Segmentation fault*.

- 1 Escrever um programa em C para ler uma quantidade de n alunos. Depois, ler a nota de cada um deles e calcular a média aritmética. Contar quantos alunos estão com a nota acima de 6.0 e imprimir esta informação.
- 2 Escreva um programa em C que, dada uma sequência de n números, imprima-os na ordem inversa a de leitura.

Resposta do Exercícios de Fixação

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

```
int main() {
```

```
    int n, count = 0;  
    float sum = 0.0, media;
```

```
    printf("Informe a quantidade de alunos: ");  
    scanf("%d", &n);
```

```
    float notas[n];
```

```
    for (int i = 0; i < n; i++) {  
        printf("Informe a nota do aluno %d: ", i + 1);  
        scanf("%f", &notas[i]);  
        sum += notas[i];  
        if (notas[i] > 6.0) {  
            count++;  
        }  
    }
```

```
    media = sum / n;  
    printf("A média aritmética é: %.2f\n", media);  
    printf("Quantidade de alunos com nota acima de 6.0: %d\n", count);
```

```
    return 0;
```

```
}
```

- 1 Escrever um programa em C para ler uma quantidade de n alunos. Depois, ler a nota de cada um deles e calcular a média aritmética. Contar quantos alunos estão com a nota acima de 6.0 e imprimir esta informação.

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

```
int main() {  
    int n;
```

```
    printf("Informe a quantidade de números: ");  
    scanf("%d", &n);
```

```
    int numeros[n];
```

```
    for (int i = 0; i < n; i++) {  
        printf("Informe o número %d: ", i + 1);  
        scanf("%d", &numeros[i]);  
    }
```

```
    printf("Números na ordem inversa: ");  
    for (int i = n - 1; i >= 0; i--) {  
        printf("%d ", numeros[i]);  
    }  
    printf("\n");
```

```
    return 0;
```

```
}
```

- 2 Escreva um programa em C que, dada uma sequência de n números, imprima-os na ordem inversa a de leitura.

- C suporta vetores multidimensionais.
 - ▶ bidimensionais,
 - ▶ tridimensionais,
 - ▶ ...
- Vamos ver, agora, a matriz: vetor bidimensional.
 - ▶ Não veremos neste curso outros tipos de vetores multidimensionais.

nome →

0					...	
1					...	
2					...	
.
m-1					...	
	0	1	2	3	...	n-1

- Uma matriz é um conjunto de valores de **tamanho** $m \times n$.
 - ▶ O número de linhas é m .
 - ▶ O número de colunas é n .
- Toda matriz é referenciada por um **identificador** (nome da matriz).
- Agora, cada valor de uma matriz é referenciado por um **par de índices** (i, j) .

- Semelhante à declaração de vetor.

Declaração de uma matriz:

<tipo> <nome da matriz>[<qtdd de linhas>] [<qtdd de colunas>];
tipo nome[m][n];

Exemplo:

```
int MATRIZ [2][2];  
float mat[3][5];
```

- Onde m e n são as dimensões da matriz, cada uma são valores inteiros entre colchetes.
 - ▶ m representa o número de linhas da matriz e
 - ▶ n representa o número de colunas da matriz.

- Uma matriz também é um conjunto de elementos de mesmo tipo **alocado contigualmente em memória**.
- Exemplo (guardar 12 inteiros):

```
int num[3][4];
```
- Neste exemplo, foi alocada uma matriz de nome *num*.
- Cada elemento tem o tipo int e são 12 deles (3×4).
- Se um int tem 4 bytes, seu tamanho total, em bytes é: $12 * 4 = 48$ bytes.

num [t] [i]	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Para calcularmos o tamanho em memória, em bytes, de uma matriz:

total em bytes = **sizeof(tipo)** * m * n


- Cada elemento tem uma posição dentro da matriz.
 - ▶ Esta posição equivale a da memória.
 - ▶ Agora, precisamos variar duas posições: de 0 a $m - 1$ para achar a linha e de 0 a $n - 1$ para achar a coluna.

- Um elemento de uma matriz é acessado por dois 'índices'.
 - Que correspondem à posição de memória.
- Acessamos da seguinte maneira:
 - Para acessar a posição (0,0), usamos nome `matriz[0][0]`;
 - Para acessar a posição (0,1), usamos nome `matriz[0][1]`;
 - Para acessar a posição (1,0), usamos nome `matriz[1][0]`;
 - Para acessar a posição (i,j), usamos nome `matriz[i][j]`;
 - Para acessar a posição (m-1,n-1), usamos nome `matriz[m-1][n-1]`.

Matriz: Exemplo 2

- Guardar em uma matriz os números de 1 a 12.

```
1  #include <stdio.h>
2
3
4  int main() {
5      int i, j, num[3][4];
6
7      // Preenchendo a matriz com valores de 1 a 12
8      for (i = 0; i < 3; i++) // Loop externo para as linhas da matriz (de 0 a 2)
9          for (j = 0; j < 4; j++) // Loop interno para as colunas da matriz (de 0 a 3)
10             num[i][j] = (i * 4) + j + 1;
11
12     // Exibe a matriz
13     for (i = 0; i < 3; i++) {
14         for (j = 0; j < 4; j++)
15             printf("%3d", num[i][j]);
16         printf("\n");
17     }
18
19     return 0;
20 }
```

- 
- Calcula o valor a ser armazenado na posição $[i][j]$
 - O valor é calculado como $(i * 4) + j + 1$:
 - Por exemplo, para $i = 0$, j varia de 0 a 3, resultando em 1, 2, 3, 4
 - Para $i = 1$, j varia de 0 a 3, resultando em 5, 6, 7, 8 e assim por diante

Matriz: Exemplo 2

- Guardar em uma matriz os números de 1 a 12.

```
1  #include <stdio.h>
2
3
4  int main() {
5      int i, j, num[3][4];
6      // Preenchendo a matriz com valores de 1 a 12
7      for (i = 0; i < 3; i++)
8          for (j = 0; j < 4; j++)
9              num[i][j] = (i * 4) + j + 1;
10     // Exibe a matriz
11     for (i = 0; i < 3; i++) {
12         for (j = 0; j < 4; j++)
13             printf("%3d", num[i][j]);
14         printf("\n");
15     }
16     return 0;
17 }
```

Saída no console

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

- Uma matriz pode ser inicializada diretamente também.
- Sintaxe semelhante a do vetor:

Inicialização de matriz:

```
tipo nome[m][n] = {lista_valores};
```

- Mas agora, cada elemento da lista de valores é um vetor entre chaves ({ e }).
 - ▶ Serão *m* vetores com *n* elementos.
- Exemplo:

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

- Para escrever uma matriz, existe um artifício muito comum:

```
/* Matriz M de inteiros 10 por 10. */  
for(i=0; i<10; i++){  
    for(j=0; j<10; j++) // Imprime linha, dado a lado.  
        printf("%4d ", M[i][j]);  
    printf("\n"); // Quebra linha para imprimir proxima linha  
}
```

- Podemos ler, usando `scanf`, cada elemento de uma matriz.
- Se a matriz tem dimensões grandes, é tediosa esta tarefa.
- Uma maneira usada para testar é gerar aleatoriamente seus valores.
 - ▶ Usar a função `rand()`, da biblioteca `stdlib.h`, por exemplo.
 - ▶ Lembrando que para gerar um número aleatório até um *limite*, devemos fazer:
F `numero_aleatorio = rand() % limite;`

▶ Exemplo:

```
int num[10][10], i, j;  
for(i=0; i<10; i++)  
    for(j=0; j<10; j++)  
        num[i][j] = rand() % 100 - 50; // Numeros de -50 a 49.
```

- Para gerar matrizes com valores diferentes, usa-se o `srand(semente)` antes do `rand()`.
 - ▶ *semente* é um número que faz diferir o conjunto de números aleatórios a ser gerado.
 - ▶ Deve-se usar sementes diferentes, então.

Soma de Matrizes: Exemplo 3

```
1 #include <stdio.h>
2 #include <stdlib.h>
3
4 int main(){
5     int i, j;
6     int A[5][4], B[5][4], C[5][4];
7
8     srand(1);
9     for(i=0; i<5; i++){
10         for(j=0; j<4; j++){
11             A[i][j] = rand() % 100 - 50;
12
13         }
14     }
15     srand(2);
16     for(i=0; i<5; i++){
17         for(j=0; j<4; j++){
18             B[i][j] = rand() % 100 - 50;
19
20         }
21     }
22     for(i=0; i<5; i++){
23         for(j=0; j<4; j++){
24             C[i][j] = A[i][j] + B[i][j];
25         }
26     }
```

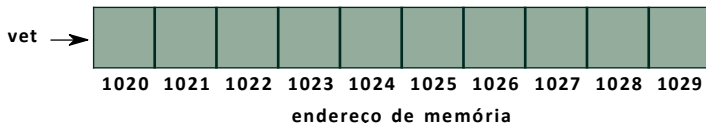
```
21     printf("Matriz A:\n");
22     for(i=0; i<5; i++){
23         for(j=0; j<4; j++){
24             printf("%4d", A[i][j]);
25             printf("\n");
26         }
27     }
28     printf("Matriz B:\n");
29     for(i=0; i<5; i++){
30         for(j=0; j<4; j++){
31             printf("%4d", B[i][j]);
32             printf("\n");
33         }
34     }
35     printf("Matriz C (soma):\n");
36     for(i=0; i<5; i++){
37         for(j=0; j<4; j++){
38             printf("%4d", C[i][j]);
39             printf("\n");
40         }
41     }
42     return(0);
43 }
```

- 1 Escreva um programa em C que leia do teclado uma matriz $m \times n$ de reais e imprima a sua transposta. Considere que as matrizes não terão dimensões maiores que 100.
- Lembrando: quando não sabemos o tamanho da matriz, alocamos o máximo de tamanho que esperamos que ela tenha.
 - F Melhor sobrar que faltar!
 - F Mas sem exagerar no tamanho: matriz ocupa espaço de memória.
 - F Os valores podem ser testados para saber se estão no intervalo esperado.
 - F Por exemplo, uma matriz de float (4 bytes) 1000×1000 ocupará 4000000 bytes ou 4 MB.
 - F Uma matriz de float (4 bytes) 10000×10000 ocupará 400000000 bytes ou 400 MB.
 - F Uma matriz de float (4 bytes) 100000×100000 ocupará 40000000000 bytes ou 40 GB.
 - F Pode ser que a memória do computador não suporte alocar toda a memória.

- Conforme já dito, vetores/matriz e ponteiros tem uma relação muito próxima.
- **Ponteiro, em C, é uma variável que guarda um endereço de memória.**
 - ▶ Dizemos que a variável é um ponteiro e ela aponta para uma área de memória.
 - ▶ Veremos com mais detalhes em aulas futuras.
 - ▶ Por enquanto, só precisamos saber que um ponteiro guarda um endereço de memória.
- Quando declaramos um vetor ou matriz em C, é alocada uma posição de memória de tamanho $\text{sizeof}(\text{tipo}) * \text{tamanho}$.
- O nome do vetor/matriz, sem índice, guarda a posição de memória do primeiro elemento.
 - ▶ O compilador C, por saber que se trata de um vetor, faz sucessivas somas a esse endereço de memória de acordo com o índice.

- Considere um vetor *vet* declarado por:

char vet[10];



- O valor de *vet* é 1020.
- O valor de *&vet[0]* (le-se endereço de *vet[0]*) também é 1020.
- O valor de *&vet[1]* é 1021 que é *vet + 1*.
- O valor de *&vet[2]* é 1022 que é *vet + 2*.
- Assim, sucessivamente.

Exemplo de Cópia de Vetores

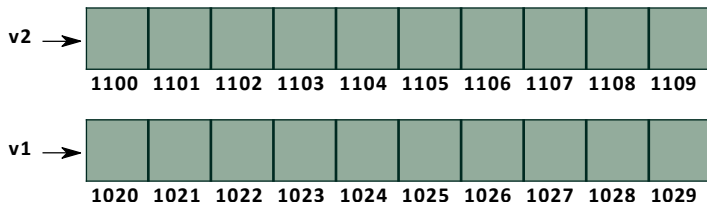
- Suponha a necessidade de realizar a cópia de vetores.
- Para isso, considere o seguinte trecho de código:

```
int v1[3] = {1, 3, 4};  
int v2[3];  
printf("end. v1: %p e end. v2: %p\n", v1, v2);  
v2 = v1;  
printf("end. v1: %p e end. v2: %p\n", v1, v2);
```

- Se o endereço de v1 é 2200 e de v2 é 2300:
 - ▶ O primeiro printf imprimirá: 2200 e 2300.
 - ▶ Mas o segundo printf imprimirá: 2200 e 2200.
- Acontecerá apenas uma atribuição de ponteiros.

Exemplo de Cópia de Vetores (II)

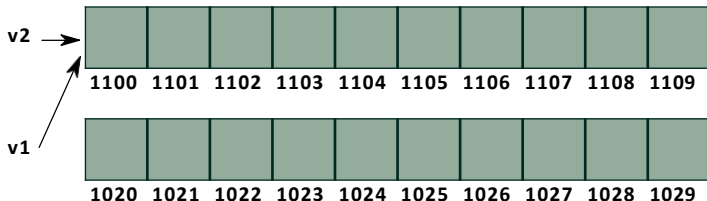
- No exemplo anterior, acontecerá apenas uma atribuição de ponteiros.



- Duas consequências:
 - 1 Não houve uma cópia dos valores: se modificar em v2, modificará também v1.
 - 2 Pior: a posição de memória apontada por v1 não terá mais referência!
 - F O programa está consumindo memória inútil!

Exemplo de Cópia de Vetores (II)

- No exemplo anterior, acontecerá apenas uma atribuição de ponteiros.



- Duas consequências:
 - 1 Não houve uma cópia dos valores: se modificar em v2, modificará também v1.
 - 2 Pior: a posição de memória apontada por v1 não terá mais referência!
 - F O programa está consumindo memória inútil!

Exemplo de Cópia de Vetores (III)

- Para realizar a cópia de vetores corretamente:

```
int v1[3] = {1, 3, 4};  
int v2[3], i;  
  
printf("end. v1: %p e end. v2: %p\n", v1, v2);  
for(i=0; i<3; i++) v2[i] = v[i];  
printf("end. v1: %p e end. v2: %p\n", v1, v2);
```

- Isto é, os valores devem ser copiados elemento a elemento.
- Neste caso, cada vetor estará com sua respectiva posição de memória.
 - ▶ Se o endereço de v1 é 2200 e de v2 é 2300:
 - F O primeiro printf imprimirá: 2200 e 2300.
 - F O segundo printf também imprimirá: 2200 e 2300.

- String é um conjunto de caracteres.
 - ▶ Pode ser usada para representar palavras, frases, texto em geral;
 - ▶ Ou qualquer outro conjunto de caracteres que faça sentido.
 - Já vimos a constante string:
 - ▶ É delimitada por aspas duplas (").
 - ▶ Embora não exista o tipo string em C.
 - Em C, uma string é definida como um **vetor de caracteres mais um terminador**.
 - O terminador é um caractere nulo: o `'\0'`.
 - ▶ Geralmente representado pelo valor 0 (posição 0 da tabela ASCII).
 - ▶ Ele serve para avisar ao compilador que terminou a string.
- F **Ele não é impresso.**

- Basta declarar um vetor de caracteres.

Declaração:

```
char nome[tamanho];
```

- *nome*: é o nome da string.
- *tamanho*: é o número de caracteres que o vetor comporta, **incluindo o terminador**.
 - ▶ Exemplo:

P	r	o	g	r	a	m	a	\0
0	1	2	3	4	5	6	7	8

Declaração de String

- Basta declarar um vetor de caracteres.

Declaração:

```
char nome[tamanho];
```

- *nome*: é o nome da string.
- *tamanho*: é o número de caracteres que o vetor comporta, **incluindo o terminador**.
 - ▶ Exemplo:

P	r	o	g	r	a	m	a	\0
0	1	2	3	4	5	6	7	8

Importante:

- O tamanho mínimo é dado pela quantidade de caracteres desejados + 1.
- No exemplo, a string "programa" deve ter o tamanho mínimo de 9.

- Uma string pode ser inicializada por várias maneiras:

- ▶ Como em vetores:

```
char str[9] = {'P', 'r', 'o', 'g', 'r', 'a', 'm', 'a', '\0'};
```

- ▶ Inicialização abreviada:

```
char str[9] = "Programa";
```

F Neste caso, o terminador é colocado automaticamente pelo compilador de C.

F Esta atribuição só é válida na declaração da variável.

- ▶ Leitura do teclado:

```
char str[9];  
scanf(" %s", str);
```

F Neste caso, o terminador é colocado pela função scanf.

F O tamanho deve ser suficiente para guardar a string lida.

- ▶ Existem outras maneiras através de funções de bibliotecas.

- Por uma string ser um vetor de caracteres, operações geralmente triviais não são simples de serem realizadas.
- Algumas dessas operações são :
 - ▶ tamanho da string,
 - ▶ cópia de strings,
 - ▶ comparação entre strings,
 - ▶ concatenação (junção) de strings,
 - ▶ entre outras. . .

- Por uma string ser um vetor de caracteres, operações geralmente triviais não são simples de serem realizadas.
- Algumas dessas operações são :
 - ▶ tamanho da string,
 - ▶ cópia de strings,
 - ▶ comparação entre strings,
 - ▶ concatenação (junção) de strings,
 - ▶ entre outras. . .
- Para entender melhor o porquê, considere a operação de cópia :
 - ▶ Dadas duas strings $s1$ e $s2$ devidamente declaradas.
 - ▶ Se fizermos $s1 = s2$, estamos atribuindo o ponteiro (`endereço`) e não os valores das strings.
F assim como em vetores!

- Por uma string ser um vetor de caracteres, operações geralmente triviais não são simples de serem realizadas.
- Algumas dessas operações são :
 - ▶ tamanho da string,
 - ▶ cópia de strings,
 - ▶ comparação entre strings,
 - ▶ concatenação (junção) de strings,
 - ▶ entre outras. . .
- Para entender melhor o porquê, considere a operação de cópia:
 - ▶ Dadas duas strings $s1$ e $s2$ devidamente declaradas.
 - ▶ Se fizermos $s1 = s2$, estamos atribuindo o ponteiro (endereço) e não os valores das strings.
F assim como em vetores!
- Sendo assim, para acessar os valores do vetor de caracteres, devemos realizar as operações elemento a elemento.
 - ▶ Através de seus índices, assim como em vetores.

- Escrever um programa em C que conte quantas letras há em uma string lida do teclado.

```
1 #include <stdio.h>
2 /* Este programa le uma string do teclado e indica o seu tamanho. */
3 int main(){
4     char str[100];
5     int tamanho = 0;
6
7     printf("Digite uma string: ");
8     scanf("%s", str);
9     while(str[tamanho] != '\0'){
10         tamanho++;
11     }
12     printf("O tamanho de %s e' %d\n", str, tamanho);
13     return(0);
14 }
```

String: Exemplo 5

- Escrever um programa em C que leia duas strings do teclado e verifique se elas são iguais.

```
1 #include <stdio.h>
2 /* Este programa le duas strings e as compara. */
3 int main(){
4     char str1[100], str2[100];
5     int i = 0, iguais = 1;
6
7     printf("Digite a primeira string: ");
8     scanf("%s", str1);
9     printf("Digite a segunda string: ");
10    scanf("%s", str2);
11    while((str1[i] != '\0' || str2[i] != '\0') && iguais){
12        if(str1[i] != str2[i]) iguais = 0;
13        i++;
14    }
15    if(iguais) printf("\'%s\' é igual a \'%s\'\n", str1, str2);
16    else printf("\'%s\' é diferente de \'%s\'\n", str1, str2);
17    return(0);
18 }
```

String: Exemplo 6

- Escrever um programa em C que testa se uma determinada string lida do teclado é um número inteiro com ou sem sinal.

```
1  #include <stdio.h>
2  /* Este programa verifica se uma string capturada do teclado
3     'e um número inteiro com sinal. */
4  int main(){
5     char str[100];
6     int i = 0, numero = 1;
7
8     printf("Digite uma string: ");
9     scanf("%s", str);
10    if(str[0] == '-' || str[0] == '+'){
11        i = 1;
12        if(str[1] == '\0') numero=0;
13    }
14    while(str[i] != '\0' && numero == 1){
15        if(str[i] < '0' || str[i] > '9') numero = 0;
16        i++;
17    }
18    if(numero == 1) printf("%s e' um numero\n", str);
19    else printf("%s nao e' um numero\n", str);
20    return(0);
```

- Trabalhar com string é algo trabalhoso.
 - ▶ Poderia ser mais simples.
- Por isso, existe uma biblioteca chamada **string.h** que permite, entre outras coisas, **manipular strings**.

Nome	Significado
strcpy(s1, s2)	Copia s2 em s1.
strcat(s1, s2)	Concatena s2 ao final de s1 (em s1).
strlen(s)	Retorna o tamanho de s.
strcmp(s1, s2)	Retorna 0 se s1 e s2 são iguais, menor que 0 se tamanho de s1 é menor que s2 e maior que 0 se tamanho de s1 é maior que tamanho de s2.
strchr(s1, c)	Retorna um ponteiro para a primeira ocorrência de c em s1.
strstr(s1, s2)	Retorna um ponteiro para a primeira ocorrência de s2 em s1.

```
1 #include <stdio.h>
2 #include <string.h>
3
4 int main(){
5     char s1[100], s2[100];
6
7     scanf("%s", s1);
8     scanf("%s", s2);
9
10    printf("comprimentos: %ld e %ld\n", strlen(s1), strlen(s2));
11    if(strcmp(s1, s2) == 0) printf("As strings são iguais\n");
12    strcat(s1, s2);
13    printf("s1: %s\n", s1);
14    strcpy(s1, "Testando...\n");
15    printf("%s", s1);
16    if(strchr("ola", 'a')) printf("a está em ola\n");
17    if(strstr("ola mundo", "ola")) printf("ola encontrado\n");
18    return(0);
19 }
```

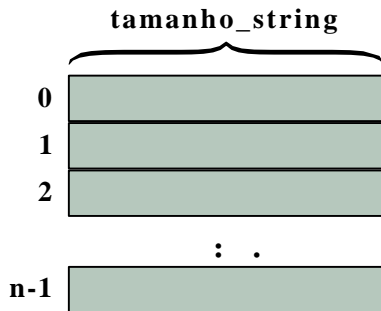

- Para as entradas "ola" e "ola", a saída na tela será:

```
comprimentos: 3 e 3  
As strings sao iguais  
s1: olaola  
Testando...  
a esta em ola  
ola encontrado
```

- É comum a necessidade do uso de um vetor de strings.
 - ▶ Ou até de matrizes de strings.
- Para declarar um vetor de strings:

```
char vetStr[n][tamanho_string];
```

- Cada string terá o tamanho indicado por `tamanho_string`;
- O vetor varia de 0 a $n - 1$ de strings.



Biblioteca string.h: Exemplo 8

- Ler 3 nomes, guardar em um vetor e escrever os nomes na ordem inversa.

```
1  #include <stdio.h>
2
3  /* Este programa lê 3 nomes, guarda em um vetor e os escreve
4   * na ordem inversa. */
5  int main(){
6
7      char nome[3][100];
8      int i;
9
10     for(i=0; i<3; i++){
11         printf("Digite o nome %d: ", i);
12         scanf("%s", nome[i]);
13     }
14     for(i=2; i>=0; i--){
15         printf("%s\n", nome[i]);
16     }
17     return(0);
18 }
```

- ❶ Escreva um programa em C que leia uma string do teclado e indique se ela é um palíndromo.
 - ▶ Um palíndromo é uma palavra que pode ser lida tanto da direita para a esquerda quanto da esquerda para a direita igualmente.
 - ▶ Exemplos:
 - F osso,
 - F radar,
 - F reviver,
 - F omiss'issimo.
- ❷ Escreva um programa em C que leia duas strings e indique se elas são iguais ou se alguma está contida na outra.