Programação Estruturada

Prof. Paulo Henrique Pisani

http://professor.ufabc.edu.br/~paulo.pisani/

outubro/2018

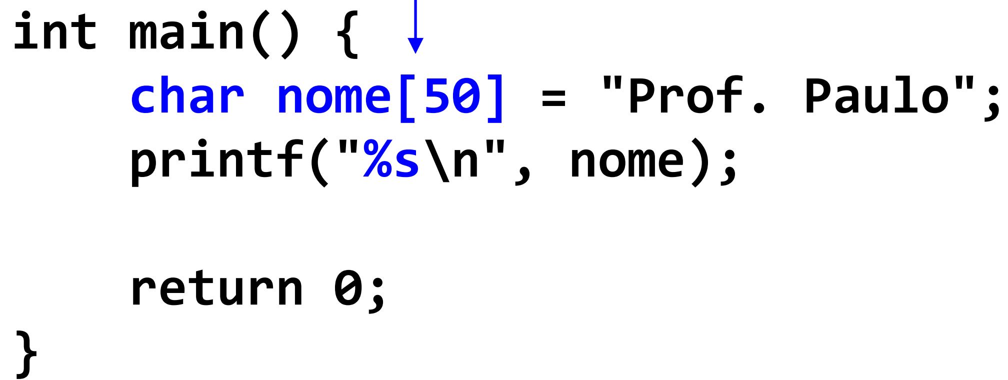
Tópicos

- Strings
- Matrizes

Strings

Strings

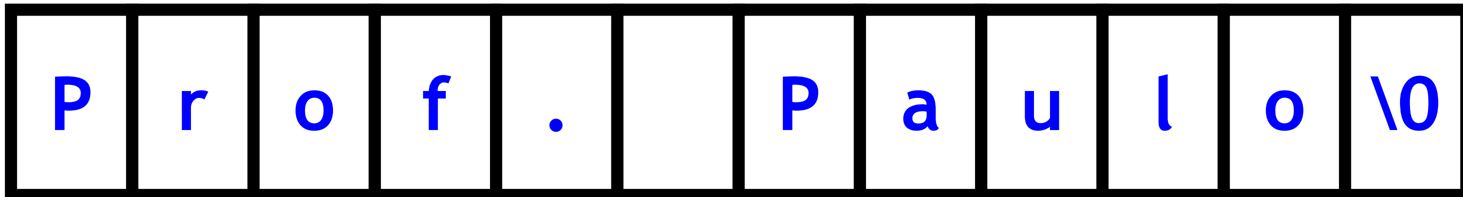
- Não há o tipo String em C;
- Para representar uma String em C, usamos um **vetor de char**.



```
#include <stdio.h>

int main() {
    char nome[50] = "Prof. Paulo";
    printf("%s\n", nome);

    return 0;
}
```

Strings

- Representação da String:



Vamos criar uma string caractere a caractere...

```
#include<stdio.h>

int main() {
    char universidade[50];
    universidade[0] = 'U';
    universidade[1] = 'F';
    universidade[2] = 'A';
    universidade[3] = 'B';
    universidade[4] = 'C';

    printf("%s\n", universidade);

    return 0;
}
```

O que será impresso?

Vamos criar uma string caractere a caractere...

```
#include<stdio.h>

int main() {
    char universidade[50];
    universidade[0] = 'U';
    universidade[1] = 'F';
    universidade[2] = 'A';
    universidade[3] = 'B';
    universidade[4] = 'C';

    printf("%s\n", universidade);

    return 0;
}
```

O que será impresso?

```
UFABCw▌\▒a■    ║k1w¡l1w└@
```

Vamos criar uma string caractere a caractere...

```c
#include<stdio.h>

int main() {
    char universidade[50];
    universidade[0] = 'U';
    universidade[1] = 'F';
    universidade[2] = 'A';
    universidade[3] = 'B';
    universidade[4] = 'C';
    universidade[5] = '\0';

    printf("%s\n", universidade);

    return 0;
}
```

O que será impresso?

Lendo Strings

- Para ler Strings, passamos o vetor para o **scanf**:

```
#include<stdio.h>

int main() {
    char universidade[50];

    scanf("%s", universidade);
    printf("%s\n", universidade);

    return 0;
}
```

AH! Cadê o “&” no scanf???

Lendo Strings

- Para ler Strings, passamos o vetor para o **scanf**:

```
#include<stdio.h>

int main() {
    char universidade[50];

    scanf("%s", universidade);
    printf("%s\n", universidade);

    return 0;
}
```

Apesar de ser o scanf, observe que aqui não usamos o “&”

Lendo Strings

- Para ler Strings, passamos o vetor para o **scanf**:

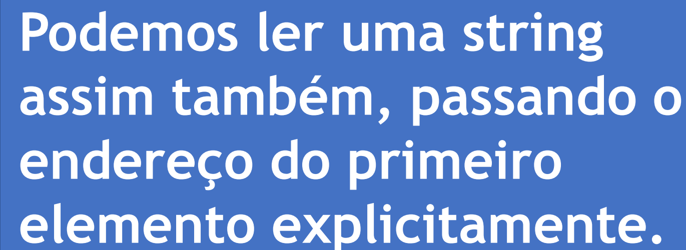
```
#include<stdio.h>

int main() {
    char universidade[50];

    scanf("%s", universidade);
    printf("%s\n", universidade);

    return 0;
}
```

Isso ocorre, porque o quando usamos o identificador do vetor sem os colchetes, ele representa o **endereço do primero elemento.**

Lendo Strings

- Para ler Strings, passamos o vetor para o **scanf**:

```
#include<stdio.h>

int main() {
    char universidade[50];

    scanf("%s", &universidade[0]);
    printf("%s\n", universidade);

    return 0;
}
```



Endereço do primeiro elemento

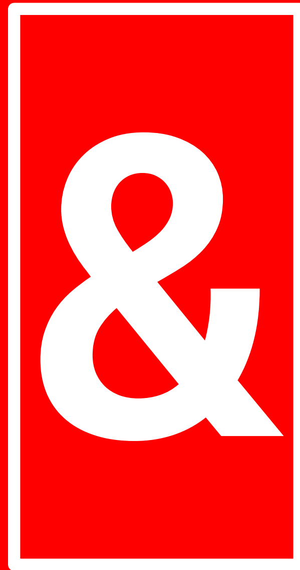
- Veja que o mesmo endereço de memória é impresso!

```
#include<stdio.h>

int main() {
    char universidade[50];

    printf("%p\n", universidade);

    printf("%p\n", &universidade[0]);

    return 0;
}
```

```c
int scanf( const char * format, ... );
```

<u>Endereços</u>

<u>das variáveis</u>



Este é o operador address-of!
Ele retorna o endereço do item a sua direita!

Por exemplo:
`&temp` retorna o endereço de `temp`
`&soma` retorna o endereço de `soma`

scanf

```
#include<stdio.h>

int main() {
    float numero;
    scanf("%f", &numero);

    scanf("%f", numero);

    char universidade[50];
    scanf("%s", universidade);

    scanf("%s", &universidade[0]);

    return 0;
}
```

O identificador do vetor reprensenta o endereço do primeiro elemento!

gets e puts

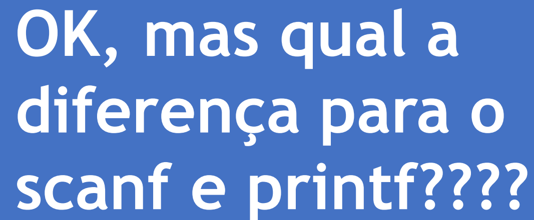
- **gets**: lê uma string;
- **puts**: imprime uma string e quebra a linha.

```
#include <stdio.h>

int main() {
    char texto[20];

    gets(texto);
    puts(texto);

    return 0;
}
```

gets e puts

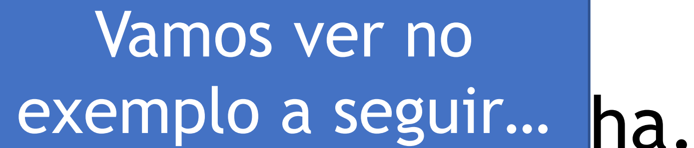
- **gets**: lê uma string;
- **puts**: imprime uma string e quebra a linha.

```
#include <stdio.h>

int main() {
    char texto[20];

    gets(texto);
    puts(texto);

    return 0;
}
```

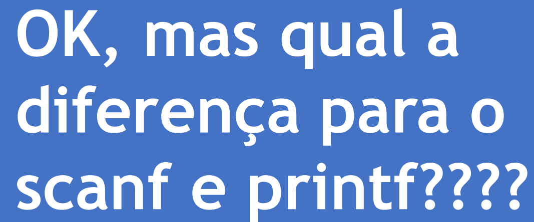


gets e puts

- **gets**: lê uma string;
- **puts**: imprime uma st ha.




```
#include <stdio.h>

int main() {
    char texto[20];

    gets(texto);
    puts(texto);

    return 0;
}
```



Há alguma diferença entre esses dois programas?

```c
#include <stdio.h>

int main() {
    char texto[20];

    scanf("%s", texto);
    printf("%s\n", texto);

    return 0;
}
```

```c
#include <stdio.h>

int main() {
    char texto[20];

    gets(texto);
    puts(texto);

    return 0;
}
```

Há alguma diferença entre esses dois programas?

```c
#include <stdio.h>

int main() {
    char texto[20];

    scanf("%s", texto);
    printf("%s\n", texto);

    return 0;
}
```

```c
#include <stdio.h>

int main() {
    char texto[20];

    gets(texto);
    puts(texto);

    return 0;
}
```

Sim! O `scanf("%s", texto)` para de ler a string quando encontra um caractere espaço, mas o gets não!!!

fgets

```
#include <stdio.h>

int main() {

    printf("Digite uma frase: ");

    char frase[6];
    fgets(frase, 6, stdin);
    puts(frase);

    return 0;
}
```

É recomendável utilizar o **fgets** ao invés do gets! o fgets limita a quantidade de caracteres lida.

```
Digite uma frase: UFABC
UFABC
```

Pegou apenas os 5 primeiros caracteres! →

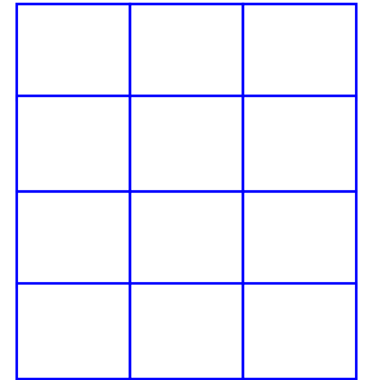
```
Digite uma frase: Universidade
Unive
```

Comprimento de uma String

- **Exercício**: leia uma String e calcule o comprimento da String.

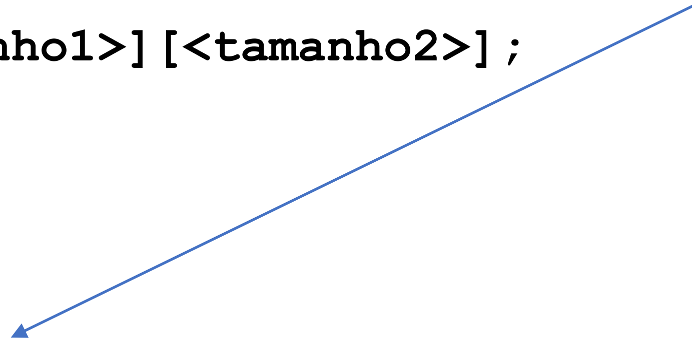
Matrizes

Matrizes



- **Declarar** matriz:

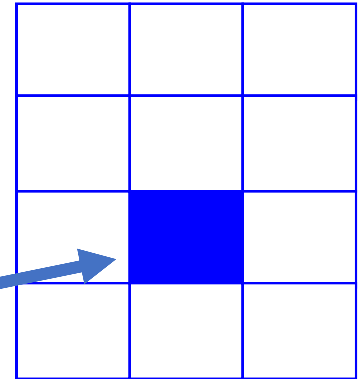
```
<tipo> <nome>[<tamanho1>][<tamanho2>];
```



Exemplos

```
int matriz[4][3];

double matriz2[4][3];
```

Matrizes



- **Acessar** valores em uma matriz:

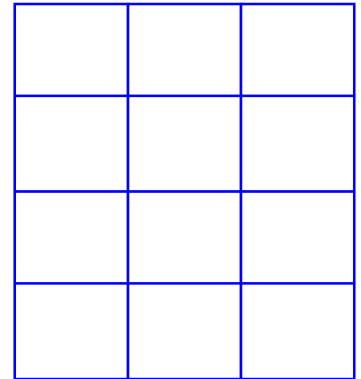
```
vetor[2][1]
```

Índices começam no 0 (zero)

- **Ler/Imprimir** elemento de matriz:

```
int matriz[4][3];

scanf("%d", &matriz[2][1]);
printf("%d\n", matriz[2][1]);
```

Matrizes



- **Percorrer** uma matriz:

```
int matriz[4][3];
int i, j;
for (i = 0; i < 4; i++)
    for (j = 0; j < 3; j++)
        matriz[i][j];
```

- **O que faz este código?**

```
int matriz[4][3];
int i, j;
for (i = 0; i < 4; i++)
    for (j = 0; j < 3; j++)
        matriz[i][j] = (i+1) * (j+1);
```

Matrizes

O que faz esse programa?

```c
#include<stdio.h>

int main() {
    float matriz[4][3];
    int i, j, c = 0;

    for (i = 0; i < 4; i++)
        for (j = 0; j < 3; j++)
            matriz[i][j] = c++;

    return 0;
}
```

E este outro?

```c
#include<stdio.h>

int main() {
    float matriz[4][3];
    int i, j, c = 0;

    for (i = 0; i < 4; i++)
        for (j = 0; j < 3; j++)
            matriz[i][j] = ++c;

    return 0;
}
```

- Quando o operador ++ está DEPOIS da variável (c++), primeiro ele retorna o valor *e depois incrementa*;
- Quando está ANTES (++c), primeiro incrementa *e depois retorna*.

O que faz esse programa?

```
#include<stdio.h>

int main() {
    float matriz[4][3];
    int i, j, c = 0;

    for (i = 0; i < 4; i++)
        for (j = 0; j < 3; j++)
            matriz[i][j] = c++;

    return 0;
}
```

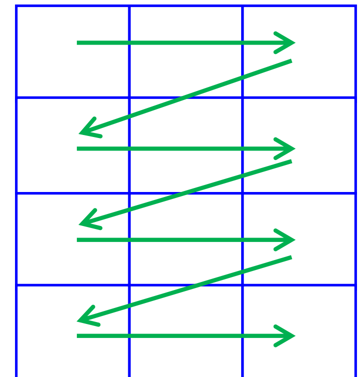
E este outro?

```
#include<stdio.h>

int main() {
    float matriz[4][3];
    int i, j, c = 0;

    for (i = 0; i < 4; i++)
        for (j = 0; j < 3; j++)
            matriz[i][j] = ++c;

    return 0;
}
```

Matrizes

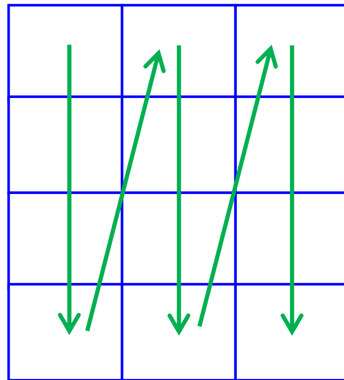
- Um pouco mais sobre **percurso em matrizes...**

```
int matriz[4][3];
int i, j;
for (i = 0; i < 4; i++)
    for (j = 0; j < 3; j++)
        matriz[i][j];
```



Matrizes

- Um pouco mais sobre **percurso em matrizes…**

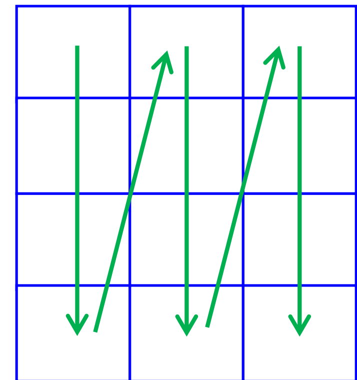


?

Matrizes

- Um pouco mais sobre **percurso em matrizes...**

```
int i, j;
int matriz[4][3];
for (int j = 0; j < 3; j++)
    for (int i = 0; i < 4; i++)
        matriz[i][j];
```



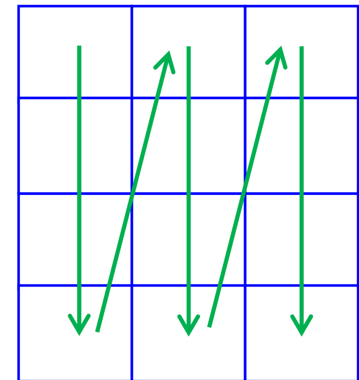
Matrizes

- Um pouco mais sobre **percurso em matrizes...**

Percorre colunas

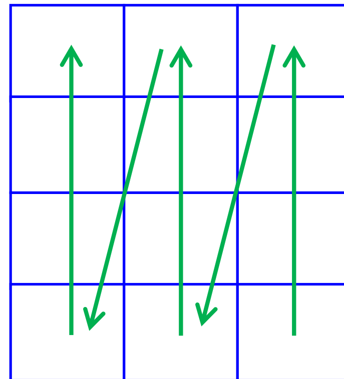
```
int i, j;
int matriz[4][3];
for (int j = 0; j < 3; j++)
    for (int i = 0; i < 4; i++)
        matriz[i][j];
```

Percorre cada elemento na coluna



Matrizes

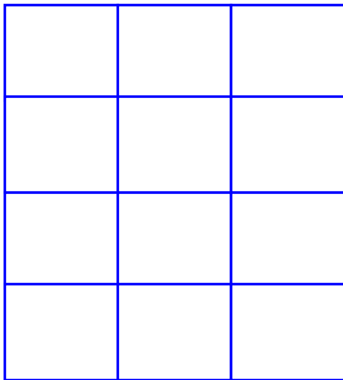
- Um pouco mais sobre **percurso em matrizes...**

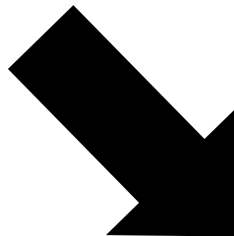


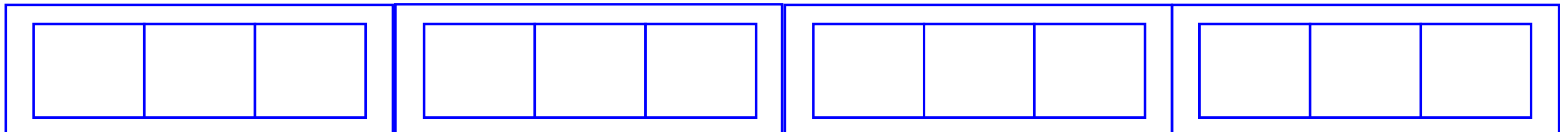
?

Matriz é um vetor de vetores

- Internamente, a matriz é um vetor unidimensional, em que cada elemento é um vetor unidimensional.







Exemplo

- Aplicar função que retorna soma dos valores de um vetor linha a linha de uma matriz;

```
int soma_vetor(int vetor[], int comp) {
    int soma = 0;
    int i;
    for (i = 0; i < comp; i++)
        soma += vetor[i];

    return soma;
}
```

```c
#include <stdio.h>

int soma_vetor(int vetor[], int comp) {
    int soma = 0;
    int i;
    for (i = 0; i < comp; i++)
        soma += vetor[i];

    return soma;
}

int main() {
    int m[3][4] = { {3,4,5,5}, {1,3,6,5}, {8,1,2,5} };

    int i;
    for (i = 0; i < 3; i++)
        printf("%d\n", soma_vetor(m[i], 4));

    return 0;
}
```

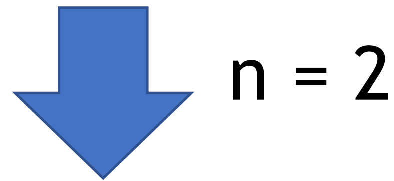
Exercício 1 - Cifra de César

- Faça uma função para criptografar uma frase com a cifra de César sobre uma string;
- Faça também uma função para reverter o processo.

Cifra de César

- Método muito simples para criptografar uma mensagem consiste em substituir cada letra pela que esta *n* posições na frente.

O vento hoje estava muito forte



n = 2

Q xgpvq jqlg guvcxc owkvq hqtvg

Exercício 1 - Cifra de César

- Como seria o protótipo dessas funções?

Exercício 1 - Cifra de César

- Como seria o protótipo dessas funções?

```
void encrypt(char texto_secreto[], int n)

void decrypt(char texto_protegido[], int n)
```

```
void encrypt(char texto_secreto[], int n) {
    int i, comp = obter_comprimento(texto_secreto);
    for (i = 0; i < comp-1; i++)
        if (texto_secreto[i] != ' ')
            texto_secreto[i] += n;
}

void decrypt(char texto_protegido[], int n) {
    int i, comp = obter_comprimento(texto_protegido);
    for (i = 0; i < comp-1; i++)
        if (texto_protegido[i] != ' ')
            texto_protegido[i] -= n;
}
```

Exercício 2 – remover a segunda palavra de uma frase

- Faça uma função que receba uma frase e remova a segunda palavra de uma frase.

```
void remove_segunda_palavra(char frase[], int comp)
```

Exercício 2 – remover a segunda palavra de uma frase

- Faça uma função que receba uma frase e remova a segunda palavra de uma frase.

```
void remove_segunda_palavra(char frase[], int comp)
```

Podemos fazer esta função sem o parâmetro de comprimento?

Exercício 2 – remover a segunda palavra de uma frase

- Faça uma função que receba uma frase e remova a segunda palavra de uma frase.

```
void remove_segunda_palavra(char frase[], int comp)
```

Podemos fazer esta função sem o parâmetro de comprimento?

Sim, nesse caso, teríamos que encontrar o caractere ‘\0’ primeiro (ele indica o fim da string).

Exercício 3 – remover a palavra de índice i de uma frase

- Faça uma função que receba uma frase e remova a palavra de índice *i* de uma frase (considere que a primeira palavra tem índice 1).

```
void remove_palavra(char frase[], int i, int comp)
```

Referências

- CELES, W.; CERQUEIRA, R.; RANGEL, J. L. Introdução a Estruturas de Dados. Elsevier/Campus, 2004.

Bibliografia básica

- PINHEIRO, F. A. C. Elementos de programação em C. Porto Alegre, RS: Bookman, 2012.
- FORBELLONE, A. L. V.; EBERSPACHER, H. F. Lógica de programação: a construção de algoritmos e estruturas de dados. 3ª edição. São Paulo, SP: Prentice Hall, 2005.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: teoria e prática. 2ª edição. Rio de Janeiro, RJ: Campus, 2002.

Bibliografia complementar

- AGUILAR, L. J. Programação em C++: algoritmos, estruturas de dados e objetos. São Paulo, SP: McGraw-Hill, 2008.
- DROZDEK, A. Estrutura de dados e algoritmos em C++. São Paulo, SP: Cengage Learning, 2009.
- KNUTH D. E. The art of computer programming. Upper Saddle River, USA: Addison- Wesley, 2005.
- SEDGEWICK, R. Algorithms in C++: parts 1-4: fundamentals, data structures, sorting, searching. Reading, USA: Addison-Wesley, 1998.
- SZWARCFITER, J. L.; MARKENZON, L. Estruturas de dados e seus algoritmos. 3a edição. Rio de Janeiro, RJ: LTC, 1994.
- TEWNENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. Estruturas de dados usando C. São Paulo, SP: Pearson Makron Books, 1995.