

# ESTRUTURA DE DADOS

Prof.<sup>a</sup> Priscilla Abreu

[priscilla.braz@rj.senac.br](mailto:priscilla.braz@rj.senac.br)



# Estrutura de dados



## Roteiro de Aula

- Objetivo da aula
- Objetivo da disciplina
- Informações sobre a disciplina
- Introdução a Estrutura de Dados
  - Tipos de dados e vetores
  - Cadeia de caracteres (Strings)

# Estrutura de dados



## Objetivo da aula

Apresentar os objetivos, informações e expectativas com a UC de Estrutura de Dados;

Introduzir a área de Estrutura de dados;

Revisar conceitos sobre tipos de dados.

# Estrutura de dados



## Objetivos da disciplina

### **Competência:**

Desenvolver estruturas de dados para armazenar e organizar informações de um sistema computacional de forma eficientemente, facilitando sua busca e modificação.

# Estrutura de dados



## Objetivos da disciplina

### Indicadores:

- Implementa estrutura de dados de acordo com as necessidades projetadas de um sistema computacional.
- Desenvolve sistemas computacionais utilizando estruturas de dados linear e não linear.
- Implementa programa de busca de informações utilizando algoritmos clássicos de busca.
- Implementa programa de ordenação de informações utilizando algoritmos clássicos de ordenação.

# Estrutura de dados



## Ementa da disciplina

- Tipos de dados estruturados homogêneos e heterogêneos;
- Cadeias de caracteres;
- Ponteiros;
- Alocação dinâmica;
- Listas lineares;
- Algoritmos de busca;
- Recursividade;
- Algoritmos de ordenação;
- Pilhas e filas;
- Árvores e Florestas;
- Introdução aos Grafos.

# Estrutura da dados



## Bibliografia da disciplina

Básica:

CORMEN, Thomas H. et. al. Algoritmos: teoria e prática. 3. ed. Rio de Janeiro: Elsevier, 2012.

LAFORE, Robert. Estrutura de dados & algoritmos em Java. Rio de Janeiro: Ciência Moderna, 2004.

PIVA JUNIOR, Dilermand. Estrutura de dados e técnicas de programação. Rio de Janeiro: Elsevier, 2014.

# Estrutura de dados



## Bibliografia da disciplina

### Complementar:

- CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. Introdução a estruturas de dados: com técnicas de programação em C. 2. ed. Rio de Janeiro: Elsevier, 2016.
- LORENZI, Fabiana; MATTOS, Patrícia Noll de; CARVALHO, Tanisi Pereira de. Estruturas de dados. São Paulo: Thomson Learning, 2007.
- MAIN, Michael. Estrutura de dados e outros objetos usando Java. 4. ed. Rio de Janeiro: Ciência Moderna, 2015.
- PUGA, Sandra; RISSETTI, Gerson. Lógica de programação e estruturas de dados com aplicações em Java. 2. ed. São Paulo: Pearson, 2009.



# Estrutura de dados



## Metodologia

- Foco no desenvolvimento de competências;
- **Sala de aula invertida;**
  - **Materiais disponibilizados previamente.**
- Introdução de conteúdos;
- Atividades e desafios práticos;

# Estrutura de dados



## Avaliação

- Avaliação contínua, no decorrer das atividades nas aulas;
- Avaliações formais:
  - 1ª avaliação: 29/03
  - 2ª avaliação: 19/04 e 26/04 (Seminário)
  - 3ª avaliação: 17/05
  - 4ª avaliação: 28/06

# Estrutura de dados



## Como aproveitar melhor o curso?

- Estudar previamente;
- Assistir às aulas;
- Fazer trabalhos e **exercícios** é indispensável;
- Esclarecer as dúvidas que surgirem;
- Complementar o aprendizado com estudo em livros, artigos, apostilas.

# Estrutura de dados



Nesta UC, estaremos focando na utilização da linguagem C para a apresentação de códigos e implementação de programas.

# DÚVIDAS?

# ESTRUTURA DE DADOS

## conceitos

# **ESTRUTURA DE DADOS**

# **X**

# **ALGORITMOS**



## **QUAL A RELAÇÃO ENTRE ESSAS ÁREAS?**



## ESTRUTURA DE DADOS X ALGORITMOS

- Estruturas de dados estão associadas a algoritmos;
- Bons algoritmos dependem da representação e da estrutura de dados adotada.
- Escolha da estrutura adequada depende diretamente do conhecimento de algoritmos para manipulá-la corretamente.



# Estrutura de dados



## INTRODUÇÃO

Tipos de dados compõem uma área essencial no contexto de algoritmos e também de estrutura de dados.

**QUAIS TIPOS DE DADOS  
VOCÊS JÁ CONHECEM?**

**COMO ESSES TIPOS DE  
DADOS PODEM SER  
DEFINIDOS E UTILIZADOS?**

## INTRODUÇÃO

### Tipos de dados

**Primitivos:** a partir dos quais podemos definir os demais

**Estruturados:** constituídos de dados primitivos e/ou estruturas

- Tipos primitivos
  - inteiro, real, lógico (boolean), caractere
- Tipos estruturados
  - Conjunto de informações agrupadas de uma forma coerente (com alguma relação entre elas)
    - Ex.: lista de presença da turma.

# Estrutura de dados

## INTRODUÇÃO

- Exemplos:
  - Tipos primitivos:
    - int idade;
    - float altura;
  - Estruturas (Tipos compostos):
    - float notas[50];
    - aluno alunos[50];



ALUNO
Nome
Matricula
Nota1
Nota2
Endereço

# Estrutura de dados



## REVISANDO...

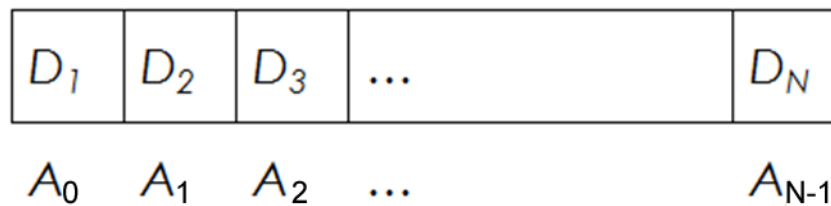
- O que são vetores?
- Para que servem? Quando utilizar?
- Que tipo de dado consigo armazenar em um vetor?
- É possível armazenar tipos de dados diferentes em um vetor? Por quê?

# Estrutura de dados



## VETOR

- É uma coleção de variáveis do mesmo tipo, referenciada por um nome comum;
- Um elemento específico é acessado através de um índice;
- São também denominados de tipos estruturados homogêneos unidimensionais.



# Estrutura de dados



## VETOR

- Declaração de um vetor em C:

tipo nome\_vetor[tamanho];

Tipo dos dados  
do conjunto

Identificador ou  
nome do conjunto

Dimensão, ou tamanho  
máximo do conjunto

# Estrutura de dados

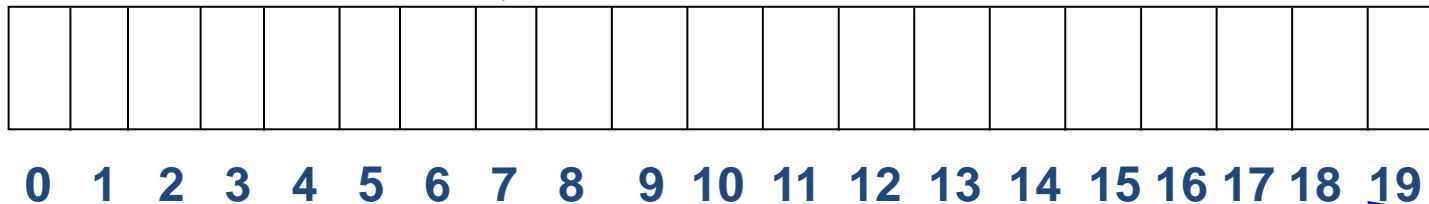


## VETORES

Nome: comum a todas as variáveis

Notas

(...)



Através da posição acessamos cada elemento do vetor.



# Estrutura de dados



## ACESSANDO UM ELEMENTO...

Coloca-se o nome da variável e entre [] coloca-se o índice, que indica a posição do elemento.

O índice é uma constante inteira, uma variável inteira ou um cálculo que resulte em valor inteiro.

Exemplos:

- `notas[1] = 10;`
- `vetor[5] = 30.4;`
- `i = 0;`
- `notas[i] = 7.5;`

# Estrutura de dados



## ACESSANDO VÁRIOS ELEMENTOS...

Cada vetor tem um único nome de variável, o que modifica é apenas a posição de cada elemento no vetor.

## USO DE ESTRUTURA DE REPETIÇÃO!!!

# Estrutura de dados



## ACESSANDO VÁRIOS ELEMENTOS...

9.5	10	8	9.4	3.5	2.9	7	8	6.8	10
notas[0]	notas[1]	notas[2]	notas[3]	notas[4]	notas[5]	notas[6]	notas[7]	notas[8]	notas[9]

**IDENTIFICAÇÃO  
DA VARIÁVEL**

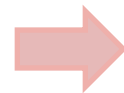


notas

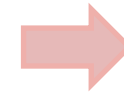


todos os elementos

**VARIAÇÃO**



índice



0 ... 9

# Estrutura de dados



## ACESSANDO VÁRIOS ELEMENTOS...

9.5	10	8	9.4	3.5	2.9	7	8	6.8	10
notas[0]	notas[1]	notas[2]	notas[3]	notas[4]	notas[5]	notas[6]	notas[7]	notas[8]	notas[9]

```
for (i=0; i<=9; i++){  
    printf("Informe a nota %d:", i );  
    scanf("%f",&notas[i]);  
}
```

# Estrutura de dados



## ATIVIDADE

Um concurso para a vaga de professor de uma universidade teve 10 professores concorrendo à vaga. Escreva um programa que armazene as 10 notas em um vetor e encontre a maior nota armazenada.

# Estrutura de dados



## ATIVIDADE

0	1	2	3	4
15	26	12	48	10

**MAIOR**

A variável maior guarda o maior valor encontrado no vetor a cada momento, até que ele seja completamente analisado.

# Estrutura de dados



## ATIVIDADE

$i = 0$

0	1	2	3	4
15	26	12	48	10

Inicializamos a variável maior com um dos valores do próprio vetor, supondo que esse seja o maior valor, inicialmente.

Por facilidade, inicializamos com o primeiro valor do vetor.

**MAIOR**

15

**maior = vet[0];**

# Estrutura de dados

## ATIVIDADE

$i = 1$

0	1	2	3	4
15	26	12	48	10



$\text{vetor}[i] > \text{maior} ?$

**MAIOR**

~~15~~

$\text{maior} = \text{vetor}[i];$



# Estrutura de dados



## ATIVIDADE

0	1	2	3	4
15	26	12	48	10



**MAIOR**

**26**

# Estrutura de dados



## ATIVIDADE

$i = 2$

0	1	2	3	4
15	26	12	48	10



$\text{vetor}[i] > \text{maior} ?$

**MAIOR**

26

**Variável maior não  
precisa ser atualizada!**

# Estrutura de dados



## ATIVIDADE

$i = 3$

0	1	2	3	4
15	26	12	48	10



$\text{vetor}[i] > \text{maior} ?$

**MAIOR**

~~26~~

$\text{maior} = \text{vetor}[i];$

# Estrutura de dados



## ATIVIDADE

0	1	2	3	4
15	26	12	48	10



**MAIOR**

**48**

# Estrutura de dados



## ATIVIDADE

$i = 4$

0	1	2	3	4
15	26	12	48	10

$\text{vetor}[i] > \text{maior} ?$

**MAIOR**

**48**



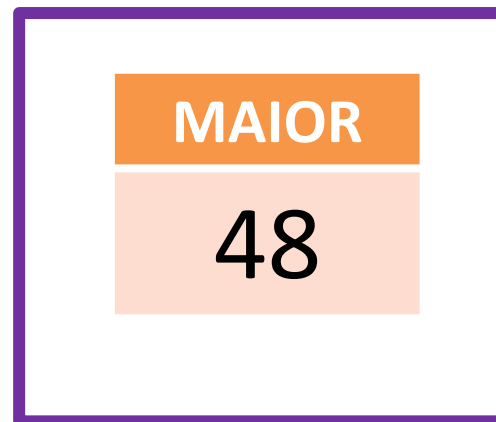
**Variável maior não  
precisa ser atualizada!**

# Estrutura de dados



## ATIVIDADE

0	1	2	3	4
15	26	12	48	10



# Estrutura de dados



## ATIVIDADE

**VAMOS IMPLEMENTAR???**

# Estrutura de dados



## ATIVIDADE

```
#include <stdio.h>
#define max 10
int main(){
    int i;
    float notas[max], maior;
    for(i=0; i<max; i++){
        printf("Informe a nota do professor %d:\n",i+1);
        scanf("%f",&notas[i]);
        if(i==0){
            maior = notas[i];
        }
        else{
            if(notas[i]>maior){
                maior = notas[i];
            }
        }
    }

    printf("A maior nota foi: %.1f",maior);
}
```



# DÚVIDAS?

# STRINGS

# Estrutura de dados



## CARACTER

Caracteres são representados internamente por códigos numéricos.

### **Tipo char:**

- Tamanho de char = 1 byte = 8 bits = 256 valores distintos.
- Tabela de códigos: define correspondência entre caracteres e códigos numéricos
  - Exemplo: ASCII

# Estrutura de dados



## CADEIA DE CARACTERES

Uma cadeia de caracteres, mais conhecida como **string**, é uma sequência de letras e símbolos, onde os símbolos podem ser espaços em branco, dígitos e vários outros, tais como pontos de exclamação e interrogação, símbolos matemáticos, etc.

## CADEIA DE CARACTERES

Em C não existe um tipo de dado string explícito. Para isso, utiliza-se um vetor de caracteres. Uma string é um vetor de caracteres com um **terminador**.

**O terminador é o caractere '\0' cujo valor numérico é 0.**

Por essa razão é necessário prever o final de uma string, que deve conter uma posição a mais do que o número de caracteres que se deseja armazenar.

Por exemplo, para declarar um vetor que guarda uma string de 10 caracteres declara-se:

```
char str[11];
```

# Estrutura de dados



## CADEIA DE CARACTERES

Formas de definir uma string:

- Como array:

```
char a[6] = {'S', 'E', 'N', 'A', 'C', '\0'}
```

ou

```
char a[] = "SENAC"
```

# Estrutura de dados



## CADEIA DE CARACTERES

Definindo e inicializando uma cadeia de caracteres:

```
char texto[100] = "Olá Mondo!";
```

Como é um vetor, podemos corrigir o caractere errado da posição 5

```
texto[5] = 'u';
```

## CADEIA DE CARACTERES

Como string não é um tipo definido em C, algumas operações não são permitidas utilizando uma string.

- Inicialização da string apenas na declaração;

- Cópia entre strings

```
str1 = str2    //operação inválida
```

- Comparação entre strings

```
if (str1 == str2) //operação inválida
```

- Para isso, utilizaremos funções de manipulações de strings.



## CADEIA DE CARACTERES

- Uso da função de `scanf()`;
  - `scanf("%s", nome);`
  - Sem uso do operador de endereço `&`.
  - Lê somente até o primeiro espaço informado e adiciona `'\0'` ao final.
- Problemas na leitura de diversas strings, principalmente com repetições: grava o ENTER no buffer de entrada.
  - Uso de `fflush(stdin)` antes do `scanf()`;
- Utilização da função `gets()` e `fgets()`.

# Estrutura de dados



## CADEIA DE CARACTERES

### Exemplo:

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

```
int main (){
```

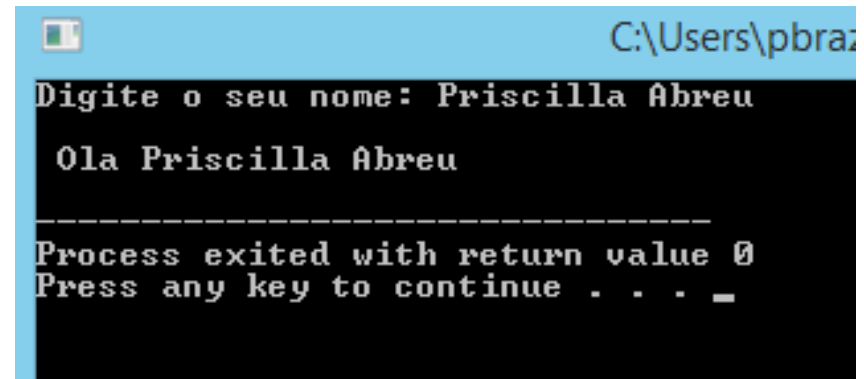
```
    char string[100];
```

```
    printf ("Digite o seu nome: ");
```

```
    fgets(string, 100, stdin); //gets (string);
```

```
    printf ("\n Ola %s",string);
```

```
}
```

A screenshot of a Windows command prompt window. The title bar shows the path "C:\Users\pbraz". The window contains the following text: "Digite o seu nome: Priscilla Abreu", "Ola Priscilla Abreu", a separator line of dashes, "Process exited with return value 0", and "Press any key to continue . . . \_".

```
C:\Users\pbraz
Digite o seu nome: Priscilla Abreu
Ola Priscilla Abreu
-----
Process exited with return value 0
Press any key to continue . . . _
```

# Estrutura de dados



## CADEIA DE CARACTERES

### Função puts ()

Exemplo:

```
#include <stdio.h>

int main() {
    char str[80] = "mensagem";
    puts(str);
}
```

# Estrutura de dados



## CADEIA DE CARACTERES

### Função puts ()

```
#include <stdio.h>

int main() {
    char nome[15];
    printf("Digite seu nome: ");
    fgets(string, 15, stdin);
    printf("Olá ");
    puts(nome);
}
```

# Estrutura de dados



## CADEIA DE CARACTERES

Considere que uma string de no máximo 50 caracteres seja lida por um programa em C. Após a leitura, a string deve ser analisada e o programa deve informar quantas vezes o caracter 'a' ('A') foi lido.

**Como fazer essa verificação???**

Importante percorrer o vetor até encontrar  
o caracter '\0'!!!

# Estrutura de dados



## CADEIA DE CARACTERES

```
#include <stdio.h>
int main (){
    char string[50];
    int qtd=0, i=0;
    printf ("Digite um texto ou palavra: ");
    fgets(string, 50, stdin);
    while (string[i]!='\0'){
        if (string[i]=='a' || string[i]=='A')
            qtd++;
        i++;
    }
    printf("Quantidade de vezes que o caracter 'a' foi
digitado: %d\n", qtd);
}
```

# Estrutura de dados



## CADEIA DE CARACTERES

Calculando o tamanho da string

```
...  
int i, n=0;  
char s[100];  
fgets(s, 100, stdin);  
for (i=0; s[i] != '\0'; i++)  
    n++;
```

Posso utilizar apenas a variável i também para a contagem.

# Estrutura de dados



## CADEIA DE CARACTERES

A biblioteca padrão fornece várias funções úteis para manipular strings.

Para usá-las, você deve incluir o cabeçalho `string.h` no início dos seus arquivos.



# Estrutura de dados



## CADEIA DE CARACTERES

### Função strcpy ()

Cópia do conteúdo de uma string.

- Cópia entre strings

`str1 = str2`      `//operação inválida`

Sintaxe:

`strcpy (destino, origem);`

# Estrutura de dados



## CADEIA DE CARACTERES

Função strcat ()

Concatena duas strings;

Não verifica tamanho;

Sintaxe:

strcat (destino, origem);

# Estrutura de dados



## CADEIA DE CARACTERES

Exemplo:

```
#include <stdio.h>
#include <string.h>
int main( ) {
    char str1[20], str2[10];
    strcpy(str1, "bom ");
    strcpy(str2, "dia");
    strcat(str1, str2);
    puts(str1);
}
```

## CADEIA DE CARACTERES

Ao criar duas strings com o mesmo conteúdo e compará-las como faria com números, verá que elas "não são iguais".

Isso ocorre porque, na verdade, o que está sendo comparado são os endereços de memória onde estão guardadas as strings.

Para comparar o conteúdo de duas strings, deve-se usar a função strcmp:

```
int strcmp (char *s1, char *s2);
```

# Estrutura de dados



## CADEIA DE CARACTERES

Função strcmp ()

Compara duas strings;  
Se iguais, retorna 0.

Sintaxe:

`strcmp (str1, str2);`

Um valor menor que zero significa que str1 é menor que str2.

Um valor maior que zero significa que str1 é maior que str2.

# Estrutura de dados



## CADEIA DE CARACTERES

### Função strlen

A função strlen retorna o tamanho, em caracteres, de uma string dada.

A função procura o terminador de string e calcula a distância dele ao início da string.

### Exemplo:

```
char nome[] = "Jose da Silva";  
int s = strlen (nome);  
//s conterà o valor 13
```

# Estrutura de dados



## CADEIA DE CARACTERES

Função `strupr(string)`

A função `strupr(string)` converte o conteúdo da string informada em letras maiúsculas.

Função `strlwr(string)`

A função `strlwr(string)` converte o conteúdo da string informada em letras minúsculas.