

## **Lista de Exercícios II**

**PE-L2 – v1.5**

**Prof. Paulo Joia Filho**

- 1 Instruções
- 2 Lista de Exercícios
- 3 Sobre a Lista...

- 1 **Instruções**
- 2 Lista de Exercícios
- 3 Sobre a Lista...

# Ferramentas necessárias

- Para resolver os exercícios você irá precisar de um compilador C instalado, preferencialmente:
  - *GNU Compiler Collection* (GCC) para plataformas Linux; ou
  - *Minimalist GNU for Windows* (MinGW) para plataformas Windows.
- Lembre-se: a lista de exercícios é uma **atividade individual!**
  - ☞ *Neste tipo de atividade o capricho e a organização são importantes.*

# Apresentação dos resultados e entrega

## Passos a serem seguidos:

- 1 Crie uma pasta com nome na forma:

**PE\_MeuPrimeiroNome\_MeuRA** *(sem espaços!)*

- 2 Para cada exercício da lista crie um arquivo **.c**, implemente as funções nele e salve na pasta criada anteriormente, com a seguinte nomenclatura:

**ex1.c, ex2.c, ...**

- Não use caracteres maiúsculos, espaços ou hífen nos nomes.
- Ao finalizar, compacte a pasta de modo a produzir um arquivo com o mesmo nome e extensão **.zip** *(não compacte como rar!)*

*Exemplo:*

**PE\_Paulo\_20181099.zip**

# Apresentação dos resultados e entrega

- 3 Crie um documento no **LibreOffice Writer** (ou Microsoft Word se preferir).
- 4 Neste documento, faça uma capa simples, intitulada:

## Lista de Exercícios II

A capa deve conter:

- Disciplina, turma e turno;
- RA e seu nome completo.

- 5 Apresente as soluções dos exercícios neste documento (código + exemplo de funcionamento) em ordem crescente, conforme proposto na lista.

👉 *Apresentar o enunciado do exercício no documento é opcional.*

# Apresentação dos resultados e entrega

- 6 Salve o documento com o nome: **PE\_MeuPrimeiroNome\_MeuRA**. Ao finalizar, exporte o documento para o formato pdf.

*Exemplo:*

**PE\_Paulo\_20181099.pdf**

- 7 Submeta os dois documentos produzidos, o **.pdf** e o **.zip**, pelo Tidia:

**Atividade Complementar - Lista 2**

<http://tidia4.ufabc.edu.br/x/Ta9pEC>

# Solução esperada

Apresente toda informação empregada na solução de forma organizada!

## ❖ Programas C

1. Toda questão deve apresentar um arquivo contendo o código-fonte em C.
2. Salve cada arquivo de acordo com o padrão de nomes explicado anteriormente.
3. Faça comentários no código para aumentar a clareza do documento quando necessário.

## ❖ Documento PDF

1. Faça um *print screen* da tela de código do **gedit** ou outro editor que estiver usando e apresente no documento pdf.
2. Logo abaixo mostre o resultado da execução do programa para algum valor de sua escolha.
3. Se necessário, explique a sequência de passos de forma clara e objetiva.



# Correção dos exercícios

## Regras importantes

**Fiquem atentos!**

- Listas submetidas na atividade errada do Tidia ou por e-mail não serão consideradas.
- Não envie arquivos no formato **rar**; caso não abra no Linux, a lista não será considerada.
- Se um dos arquivos não for enviado (**pdf ou zip**) ou estiver **corrompido** a lista não será considerada.
- Questões que não apresentarem o programa C correspondente não serão consideradas.
- Questões com arquivos de código fora do padrão de nomes serão penalizadas.
- Cada questão deve ter **um único arquivo .c** associado, não crie cabeçalhos nem arquivos de código adicionais, pois irá falhar durante a compilação automática.
- Os arquivos da pasta **common** podem ser utilizados como parte da solução, desde que seja respeitado o caminho `../common`. Exemplo:

```
#include "../common/pe_utility2.h"
```

- Não modifique os arquivos da pasta **common**, nem implemente novas funções.
- Não envie a pasta **common**, a versão utilizada para correção será a última disponibilizada no site.

# Correção dos exercícios

## Deteção de plágio



- Será utilizada a tecnologia

### **Measure Of Software Similarity (Moss)**

<http://theory.stanford.edu/~aiken/moss/>

para verificação de similaridade de código.

*O sistema criado em 1994, é bastante efetivo nesta tarefa. Por exemplo, simples troca de nomes de variáveis são identificadas pelo Moss.*

*Portanto, evitem faltas desta natureza. Pois, se um certo grau de similaridade for detectado entre duas ou mais listas, todos os envolvidos receberão Conceito F.*

## 1 Instruções

## 2 Lista de Exercícios

- Alocação de memória
- Estruturas, enumerações e tipos
- Entrada e saída com arquivos
- Recursividade

## 3 Sobre a Lista...

# Alocação Dinâmica de Arranjos

## Exercício 1 (Inversão de Strings)

Escreva a função **strflip** que receba a string **s** como argumento e inverta **s** sem declarar ou alocar novos arranjos (inversão **in-place**). A string **s** deve ser alocada dinamicamente na função **main** com tamanho máximo de 256 caracteres. A função deve ter a seguinte assinatura:

```
void strflip(char *s)
```

### Exemplos de Funcionamento

Entre uma palavra ou frase:

AMOR

Frase invertida:

ROMA

Entre uma palavra ou frase:

PROGRAMAR E UMA ARTE, FICAR LOUCO FAZ PARTE!

Frase invertida:

!ETRAP ZAF OCUOL RACIF ,ETRA AMU E RAMARGORP

# Alocação Dinâmica de Arranjos

## Exercício 2

Faça um programa que sorteie  $n$  **números ímpares** entre 1 e 999 e armazene-os em um vetor de inteiros **alocado dinamicamente**. Em seguida, escreva uma função que receba este vetor e seu tamanho como parâmetros, e devolva a soma de seu elementos. Assinatura da função:

```
int sum_vector(int *v, int n)
```

### Exemplo de Funcionamento

Tamanho do vetor:  $n = 10$

Vetor sorteado:

[ 863 795 33 75 931 907 579 461 357 611 ]

Soma dos elementos = 5612

# Alocação Dinâmica de Arranjos

## Definição (Selection Sort)

É um algoritmo de ordenação baseado em passar o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição, e assim, sucessivamente, até o elemento que está na posição  $n - 1$  (penúltimo elemento).

## Exercício 3 (Ordenação por Seleção)

Semelhante ao Exercício 2, mas desta vez sorteie  $n$  **números pares** no intervalo de 2 até 1000. Troque a função **sum\_vector** por:

```
void selection_sort(int *v, int n)
```

Esta função deve ordenar **v** usando o algoritmo *selection sort* explicado acima.

## Observação:

- Não use vetor auxiliar para fazer a troca (ordenação **in-place**).

# Alocação Dinâmica de Arranjos

## Exercício 3

### Exemplo de Funcionamento

Tamanho do vetor:  $n = 12$

Vetor sorteado:

[ 288 722 14 526 540 704 560 106 198 290 952 912 ]

Vetor ordenado:

[ 14 106 198 288 290 526 540 560 704 722 912 952 ]

# Alocação Dinâmica de Arranjos Bidimensionais

## Exercício 4

Faça uma função que receba como parâmetros uma matriz  $A$  de dimensão  $m \times n$  e os inteiros  $r$ ,  $c$  representando uma linha e uma coluna da matriz, respectivamente. A função deve retornar a soma total dos elementos que pertencem à linha  $r$  e à coluna  $c$  da matriz, sem somar duas vezes elementos que pertencem à interseção. Assinatura:

```
int sum_rowcol(int **A, int m, int n, int r, int c)
```

### Observações:

- Aloque a matriz  $A$ , dinamicamente, como um ponteiro para ponteiros de inteiros.
- Não esqueça de liberar a memória no final.



# Alocação Dinâmica de Arranjos Bidimensionais

## Exercício 4

### Exemplo de Funcionamento

Informe as dimensoes da matriz: m x n = 4 3

Escolha uma linha (de 1 a m): r = 4

Escolha uma coluna (de 1 a n): c = 3

Informe os elementos da matriz:

A[1][1] = 1

A[1][2] = 2

A[1][3] = 3

A[2][1] = 4

A[2][2] = 5

A[2][3] = 6

A[3][1] = 7

A[3][2] = 8

A[3][3] = 9

A[4][1] = 10

A[4][2] = 11

A[4][3] = 12

Soma da linha 4 com a coluna 3 = 51

# Alocação Dinâmica de Arranjos Bidimensionais

## Exercício 5 (Permutação de Linhas de uma Matriz)

Faça uma função que receba como parâmetros uma matriz  $A$  de dimensão  $m \times n$  e dois inteiros  $r1$ ,  $r2$ . A função deve trocar o conteúdo das linhas  $r1$  e  $r2$  entre si. Esta operação é conhecida como permutação de linhas.

Assinatura:

```
void swap_row(int *A, int n, int r1, int r2)
```

Observações:

- Neste caso, a matriz  $A$  deve ser alocada como um ponteiro de inteiros; use aritmética de ponteiros para percorrer a matriz.
- Não esqueça de liberar a memória no final.

# Alocação Dinâmica de Arranjos Bidimensionais

## Exercício 5

### Exemplo de Funcionamento

Informe as dimensoes da matriz: m x n = 3 4

Informe a 1a linha (de 1 a m): r1 = 2

Informe a 2a linha (de 1 a m): r2 = 3

Informe os elementos da matriz:

A[1][1] = 1

A[1][2] = 2

A[1][3] = 3

A[1][4] = 4

A[2][1] = 5

A[2][2] = 6

A[2][3] = 7

A[2][4] = 8

A[3][1] = 9

A[3][2] = 10

A[3][3] = 11

A[3][4] = 12

Matriz de entrada:

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

Matriz permutada:

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

# Alocação Dinâmica de Arranjos Bidimensionais

## Exercício 6 (Quadrado Mágico)

Uma matriz quadrada de inteiros é um quadrado mágico se a soma dos elementos de cada linha, de cada coluna, da diagonal principal e da diagonal secundária são todas iguais. A matriz abaixo é um exemplo de quadrado mágico:

$$A = \begin{bmatrix} 3 & 4 & 8 \\ 10 & 5 & 0 \\ 2 & 6 & 7 \end{bmatrix}$$

Faça uma função que receba uma matriz quadrada como entrada e determine se ela é um quadrado mágico. Use a seguinte assinatura para a função:

```
int magic_square(int **A, int n)
```

Observações:

- Aloque a matriz  $A$ , dinamicamente, como um ponteiro para ponteiros.
- Se  $A$  for um quadrado mágico retorne 1, caso contrário, retorne 0.

# Alocação Dinâmica de Arranjos Bidimensionais

## Exercício 6

### Exemplo de Funcionamento

Informe a dimensao da matriz: n = 4

Informe os elementos da matriz:

A[1][1] = 16

A[1][2] = 3

A[1][3] = 2

A[1][4] = 13

A[2][1] = 5

A[2][2] = 10

A[2][3] = 11

A[2][4] = 8

A[3][1] = 9

A[3][2] = 6

A[3][3] = 7

A[3][4] = 12

A[4][1] = 4

A[4][2] = 15

A[4][3] = 14

A[4][4] = 1

Esta matriz É um quadrado magico.

## Exercício 7

A estrutura abaixo é utilizada para armazenar informações sobre produtos de um supermercado:

```
#define MAX_NOME 80

typedef struct {
    char *nome;
    double preco;
    int quantidade;
} produto_t;
```

Implemente a função `calc_produto` para calcular o preço médio e a quantidade total de produtos em estoque. A função deve apresentar a seguinte assinatura:

```
void calc_produto(produto_t *produto, int n,
                  double *preco_medio, int *qtd_total)
```

### Observação:

- Crie o vetor de produtos via alocação dinâmica de memória.
- Não esqueça de alocar memória para cada nome de produto a ser armazenado.

## Exercício 7

### Exemplo de Funcionamento

```
Nr de produtos: n = 3
```

```
-----  
Produto 1:
```

```
Nome: Leite
```

```
Preco: 2.85
```

```
Qtde: 12
```

```
-----  
Produto 2:
```

```
Nome: Miojo
```

```
Preco: 3.20
```

```
Qtde: 20
```

```
-----  
Produto 3:
```

```
Nome: Sabonete
```

```
Preco: 4.5
```

```
Qtde: 32
```

```
Preco medio = 3.52
```

```
Qtd. total = 64
```

## Exercício 8

Duplique o [Exercício 7](#), substituindo a função `calc_produto` pela função

```
void ordena_produto(prodoto_t *produto, int n)
```

Esta função deve receber o vetor de produtos como parâmetro e colocá-lo em ordem crescente de preços usando o método *selection sort* (ver detalhes no [Exercício 3](#)).



## Exercício 8

### Exemplo de Funcionamento

Nr de produtos: n = 3

-----  
Produto 1:

Nome: Sabonete

Preco: 4.5

Qtde: 32

-----  
Produto 2:

Nome: Leite

Preco: 2.85

Qtde: 12

-----  
Produto 3:

Nome: Miojo

Preco: 3.2

Qtde: 20

Produtos ordenados:

Leite                   \$ 2.85

Miojo                   \$ 3.20

Sabonete               \$ 4.50

## Exercício 9

Utilize as estruturas abaixo para criar uma relação **1:n** entre uma turma e seus respectivos alunos:

```
typedef struct {
    size_t ra;
    char *nome;
    float media;
} aluno_t;

typedef struct {
    char* codigo;
    int nr_alunos;
    aluno_t* alunos;
} turma_t;
```

Para isto, codifique as funções abaixo com base no *template* apresentado no próximo slide:

```
turma_t* cria_turma(int nr_alunos); // aloca mem. p/ todas as estruturas
void cadastra_alunos(turma_t *turma); // cadastra os alunos
void lista_turma(turma_t *turma); // lista os alunos
void exclui_turma(turma_t *turma); // libera mem. de todas as estruturas
```

Exercício 9 – *Template*

```

1  #include "../common/pe_utility2.h"

3  #define MAX_CODIGO 50
   #define MAX_NOME 100

5

   typedef struct {
7     size_t ra;
       char *nome;
9     float media;
   } aluno_t;

11

   typedef struct {
13     char* codigo;
       int nr_alunos;
15     aluno_t* alunos;
   } turma_t;

17

   turma_t* cria_turma(int nr_alunos);
19 void cadastra_alunos(turma_t *turma);
   void lista_turma(turma_t *turma);
21 void exclui_turma(turma_t *turma);

```

```

int main() {
24     int n;
       printf("Nr de alunos: "); scanf("%d", &n);
26     turma_t *turma = cria_turma(n);

28     printf("Cod. da turma: "); scanf("%s",
       turma->codigo);
       cadastra_alunos(turma);

30

       char c;
32     printf("\nListar alunos [y/n]? "); scanf("
       %c", &c);
       if (c == 'y') lista_turma(turma);
34     exclui_turma(turma);

36     return 0;
}

```

## Exercício 9<sup>1</sup>

### Exemplo de Funcionamento

```
Nr de alunos: 3
Cod. da turma: PE-NOTURNO
-----
Cadastro de Alunos
-----
Aluno #1
  RA: 11111
  Nome: Alan_Turing
  Media: 10
Aluno #2
  RA: 2222
  Nome: Karl_Marx
  Media: 3.4
Aluno #3
  RA: 3333
  Nome: Dennis_Ritchie
  Media: 9.8

Listar alunos [y/n]? y
-----
Listagem da Turma ABC
-----
Ord RA           Nome                Media
1 11111          Alan_Turing          10.0
2 2222           Karl_Marx            3.4
3 3333           Dennis_Ritchie       9.8
```

<sup>1</sup>Se usar `scanf`, digite os nomes sem espaço.

## Exercício 10

Duplique o código do [Exercício 9](#) e adicione duas novas funcionalidades:

```
float media_turma(turma_t *turma);  
aluno_t* max_media(turma_t *turma);
```

A primeira deve retornar a **média aritmética da turma**, enquanto que a segunda, deve retornar a **maior média da turma** e o nome do respectivo aluno. Imprima os valores retornados a partir da função **main**.

### Exemplo de Funcionamento

```
Nr de alunos: 3  
Cod. da turma: PE-NOTURNO
```

```
-----  
Cadastro de Alunos  
-----
```

```
Aluno #1
```

```
  RA: 11111
```

```
  Nome: Alan_Turing
```

```
  Media: 10
```

```
  ...
```

```
Listar alunos [y/n]? n
```

```
-----  
Media da turma: 7.73
```

```
  Maior media: 10.0  (Alan_Turing)
```

# Manipulação de Arquivos

## Exercício 11

Faça um programa que grave duas matrizes quadradas  $A$  e  $B$  em arquivo, **A.txt** e **B.txt**, respectivamente (siga o exemplo explicado em aula). As matrizes devem conter números inteiros aleatórios no intervalo  $[-100, 100]$ .

### Exemplo de Funcionamento

Dimensoes de A: 3 2

Dimensoes de B: 2 4

Arquivo A.txt:

3 2

95 -70

-44 12

-35 23

Arquivo B.txt:

2 4

32 -2 71 3

-81 54 55 43

### Dica:

Para gerar as matrizes  $A$  e  $B$  foram usadas as seguintes sementes, respectivamente:

```
srand(7.3);
```

```
srand(13.7);
```

# Manipulação de Arquivos

## Exercício 12

Leia as matrizes  $A$  e  $B$  a partir dos arquivos do exercício anterior e calcule  $A \times B$ . Exiba o resultado da multiplicação no console.

### Exemplo de Funcionamento

Arquivo A.txt:

3 2

95 -70

-44 12

-35 23

Arquivo B.txt:

2 4

32 -2 71 3

-81 54 55 43

Matriz A x B:

3 4

8710 -3970 2895 -2725

-2380 736 -2464 384

-2983 1312 -1220 884

### Dica:

Leia cada matriz e armazene em um `int**`, isto evita a aritmética com índices durante a multiplicação de matrizes.

# Manipulação de Arquivos

## Exercício 13

O arquivo `inteiros.txt` contém 50 números inteiros aleatórios. A partir da leitura deste arquivo, produza dois novos arquivos, um contendo os números pares (`pares.txt`) e outro contendo os números ímpares (`impares.txt`). Dicas:

- Crie um ponteiro para cada arquivo, desse modo é possível separar pares e ímpares à medida que o arquivo de inteiros é lido.
- Como recuperar a quantidade de inteiros no arquivo, caso ela não fosse informada?

### Resultado Esperado

#### Arquivo `pares.txt`:

```
66 804 448 364 330 170 178 18 48 856 562 992 858 630 674 362 886 516
722 704 790 918 638 350 122 338 758
```

#### Arquivo `impares.txt`:

```
881 709 599 179 223 235 957 553 19 183 211 607 347 779 615 327 551 691
281 271 717 543 451
```



# Manipulação de Arquivos

## Exercício 14

Considere o arquivo `temperaturas.txt` que contém as temperaturas de uma cidade do interior no período de um ano (uma temperatura por linha). Faça um programa que calcule e apresente:

- a) A menor temperatura ocorrida;
- b) A maior temperatura ocorrida;
- c) A temperatura média;
- d) O número de dias nos quais a temperatura foi inferior à temperatura média;
- e) O desvio padrão deste conjunto de valores.

**Dica:** os itens **d)** e **e)** dependem do cálculo da temperatura média, portanto, é aconselhável ler os dados e armazená-los em um *array*, evitando desse modo, percorrer o arquivo múltiplas vezes.

## Exercício 14

### Resultado Esperado

```
Menor temperatura: 10.1  
Maior temperatura: 24.8  
Temperatura media: 17.7  
Temperaturas inferior a media: 184  
Desvio padrao do conjunto: 2.7
```

### Validando os Resultados com Python

```
>>> L = [20, 20.5, 20, ..., 15.5, 16]  
>>> min(L)  
10.1  
>>> max(L)  
24.8  
>>> import statistics as stat  
>>> mean = stat.mean(L)  
>>> "%.1f" % mean  
'17.7'  
>>> sum(i < mean for i in L)  
184  
>>> "%.1f" % stat.stdev(L)  
'2.7'
```

# Manipulação de Arquivos

## Exercício 15

Considere o arquivo **votos.txt** que contém, em cada linha, um valor de 0 a 5. Estes valores representam o resultado de uma eleição, tal que 0 representa um voto nulo ou abstenção e os valores de 1 a 5 representam os códigos de identificação dos candidatos. Faça um programa que apure o resultado dessa eleição. O programa deverá apresentar como resultado:

- a) O código de identificação e a quantidade de votos do candidato mais votado;
- b) O código de identificação e a quantidade de votos do candidato menos votado;
- c) A quantidade de votos nulos/abstenções.

**Dica:** crie um *array* com 6 posições (índices de 0 a 5) usando **calloc**, para acumular os votos de cada candidato; isto facilita muito a tarefa, pois o índice do vetor coincide com o valor lido do arquivo.

# Manipulação de Arquivos

## Exercício 15

### Resultado Esperado

```
Nr total de votos: 5000
Nr de votos em 0: 834
Nr de votos em 1: 831
Nr de votos em 2: 858
Nr de votos em 3: 818
Nr de votos em 4: 824
Nr de votos em 5: 835

Candidato mais votado: 2 => 858 votos
Candidato menos votado: 3 => 818 votos
Votos nulos/abstencoes: 0 => 834 votos
```

# Funções Recursivas

## Exercício 16 (Menor Valor de um Vetor)

Faça uma função recursiva que receba um vetor de double e seu tamanho como entrada. Ao final, retorne o **menor** elemento deste vetor.

Assinatura:

```
double get_min(double *v, int n)
```

Observações:

- Aloque **v**, dinamicamente, na função **main**.
- Lembre-se de liberar a memória no final.

### Exemplo de Funcionamento

```
Nr de elementos: n = 4
Informe os valores:
v[0] = 3.5
v[1] = -2
v[2] = 7.8
v[3] = 5
min(v) = -2
```

# Funções Recursivas

## Exercício 17

Crie uma função recursiva para calcular o valor da soma abaixo para  $n$  termos ( $n = 50$ ).

$$S = \frac{1}{1} + \frac{3}{2} + \frac{5}{3} + \frac{7}{4} + \cdots + \frac{99}{50}$$

Assinatura:

```
float sum_serie(int n)
```

### Exemplo de Funcionamento

Nr de termos:  $n = 50$

Soma = 95.500793

# Funções Recursivas

## Definição (Série de Gregory-Leibniz)

Em matemática, a fórmula de Leibniz para calcular  $\pi$  (também denominada série de Gregory-Leibniz, em reconhecimento ao trabalho de James Gregory), é dada por:

$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \dots = \frac{\pi}{4}$$

## Exercício 18 (Estimando o valor de $\pi$ )

Escreva uma função recursiva para aproximar  $\pi$  pela série de Gregory-Leibniz. O número de termos da série deve ser informado pelo usuário.

### Notas:

- Faça todos os cálculos com precisão dupla;
- Formate a saída com 15 casas decimais.

### Exemplo de Funcionamento

```
Nr de termos: n = 1000  
Pi = 3.140592653839794
```

# Funções Recursivas

## Exercício 19

Faça uma função recursiva para converter um número inteiro  $x$  ( $x > 0$ ), do formato decimal para o formato binário. Assinatura:

```
void dec2bin(int x)
```

Nota:

- Neste caso, não é necessário retornar um valor, faça a própria função imprimir o resto da divisão por 2, após a chamada recursiva, é mais simples!

### Exemplos de Funcionamento

```
x(10) = 12  
x(2) = 1100
```

```
x(10) = 1270  
x(2) = 10011110110
```



# Funções Recursivas

## Exercício 20

Crie a função recursiva **isinstr** que recebe uma string **s** (no máximo 100 caracteres) e um caracter **c** como parâmetros e retorna quantas vezes o caracter **c** aparece na string; retorne zero caso  $c \notin s$ . Note que, esta função diferencia caracteres maiúsculos de minúsculos (*case sensitive*).

```
int instr(const char *s, char c, int i)
```

**Nota:** utilize o exercício que calcula o comprimento da string como modelo.

### Exemplos de Funcionamento

Frase: Anotaram a Maratona

Caracter: a

6

Frase: Anotaram a Maratona

Caracter: M

1

## Exercício 20 – *Template*

```
1  #include <stdio.h>
   #include <stdlib.h>
3
   #define MAXLEN 100
5
   int instr(const char *s, char c, int i) {
7   // code here...
   }
9
   int main() {
11  char *s = (char*) malloc (MAXLEN * sizeof(char));
    printf("Frase: "); fgets(s, MAXLEN, stdin);
13
    char c;
15  printf("Caracter: "); scanf(" %c", &c);
    printf("%d\n", instr(s, c, 0));
17
    free(s);
19  return 0;
   }
```

- 1 Instruções
- 2 Lista de Exercícios
- 3 Sobre a Lista...**
  - Algumas considerações
  - Referências bibliográficas

# Importante!

## *Dicas para realizar uma boa prova:*

- ✓ Resolver e entender os exercícios da Lista.
- ✓ Rever os conceitos apresentados durante as aulas.
- ✓ Consultar a bibliografia sugerida sobre o assunto quando surgir dúvidas.
- ✓ Procurar ajuda se as dúvidas persistirem!

# Referências Bibliográficas I



Aguilar, L. J. (2008).

*Programação em C++: Algoritmos, Estruturas de Dados e Objetos.*  
McGraw-Hill, São Paulo.



Cormen, T. H., Leiserson, C. E., Rivest, R. L., e Stein, C. (2002).

*Algoritmos: Teoria e Prática.*  
Elsevier, Rio de Janeiro.



Drozdek, A. (2009).

*Estrutura de Dados e Algoritmos em C++.*  
Cengage Learning, São Paulo.



Forbellone, A. L. V. e Eberspacher, H. F. (2005).

*Lógica de Programação: A Construção de Algoritmos e Estrutura de Dados.*  
Pearson Prentice Hall, São Paulo, 3 edition.



Knuth, D. E. (2005).

*The Art of Computer Programming.*  
Addison-Wesley, Upper Saddle River, NJ, USA.



Pinheiro, F. d. A. C. (2012).

*Elementos de Programação em C.*  
Bookman, Porto Alegre.

# Referências Bibliográficas II



Sedgewick, R. (1998).

*Algorithms in C: Parts 1-4, Fundamentals, Data Structures, Sorting, Searching.*  
Addison-Wesley, Boston, 3rd edition.



Szwarcfiter, J. L. e Markenzon, L. (1994).

*Estruturas de Dados e Seus Algoritmos.*  
LTC, Rio de Janeiro.



Tenenbaum, A. A., Langsam, Y., e Augenstein, M. J. (1995).

*Estruturas de Dados Usando C.*  
Makron Books, São Paulo.