

# Introdução à Programação

## AULA 6 – Vetores e Strings

Prof<sup>a</sup>. Glaucia M. M. Campos

[glauciamelissa@uern.br](mailto:glauciamelissa@uern.br)

# Estrutura de Dados

---

O que é uma  
**ESTRUTURA DE DADOS?**

# Estrutura de Dados

---

**ESTRUTURA DE DADOS  
=  
CONJUNTO DE DADOS**

# Estrutura de Dados

---

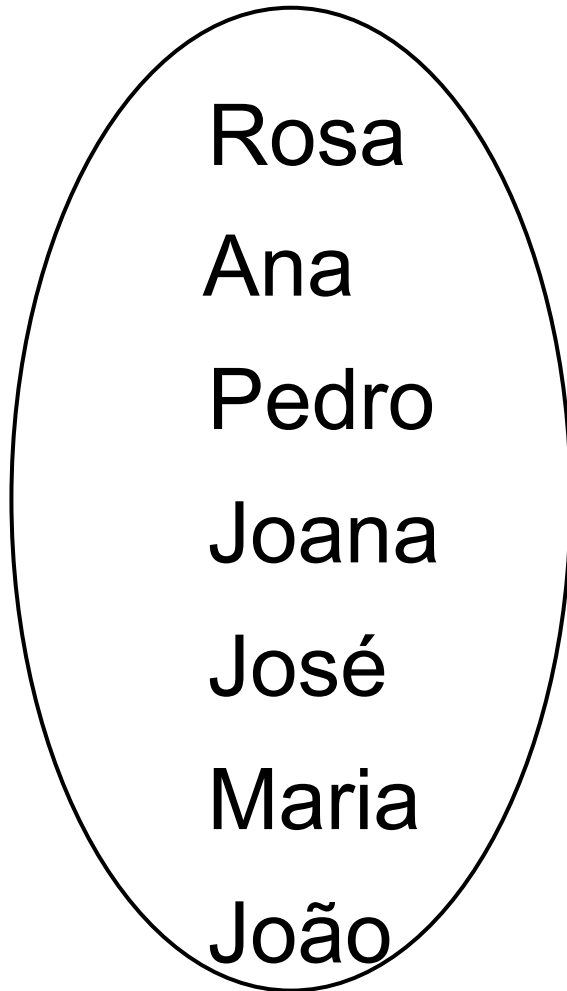
O que é uma  
estrutura de dados  
HOMOGÊNEA?

**ESTRUTURA DE DADOS  
HOMOGÊNEA  
=  
CONJUNTO DE DADOS  
DO MESMO TIPO**

# Estrutura de Dados

---

Alunos



**Conjunto de Alunos**

**=**

**Estrutura de dados  
homogênea**

**=**

**Estrutura de dados do  
tipo CHARACTER**

# Estrutura de Dados

---

**Conjunto de Idades**

**=**

**Estrutura de dados  
homogênea**

**=**

**Estrutura de dados do  
tipo INTEIRO**

**Idades**

23

38

47

19

26

52

29

# Estrutura de Dados Unidimensional

---

O que é estrutura de dados  
**UNIDIMENSIONAL?**

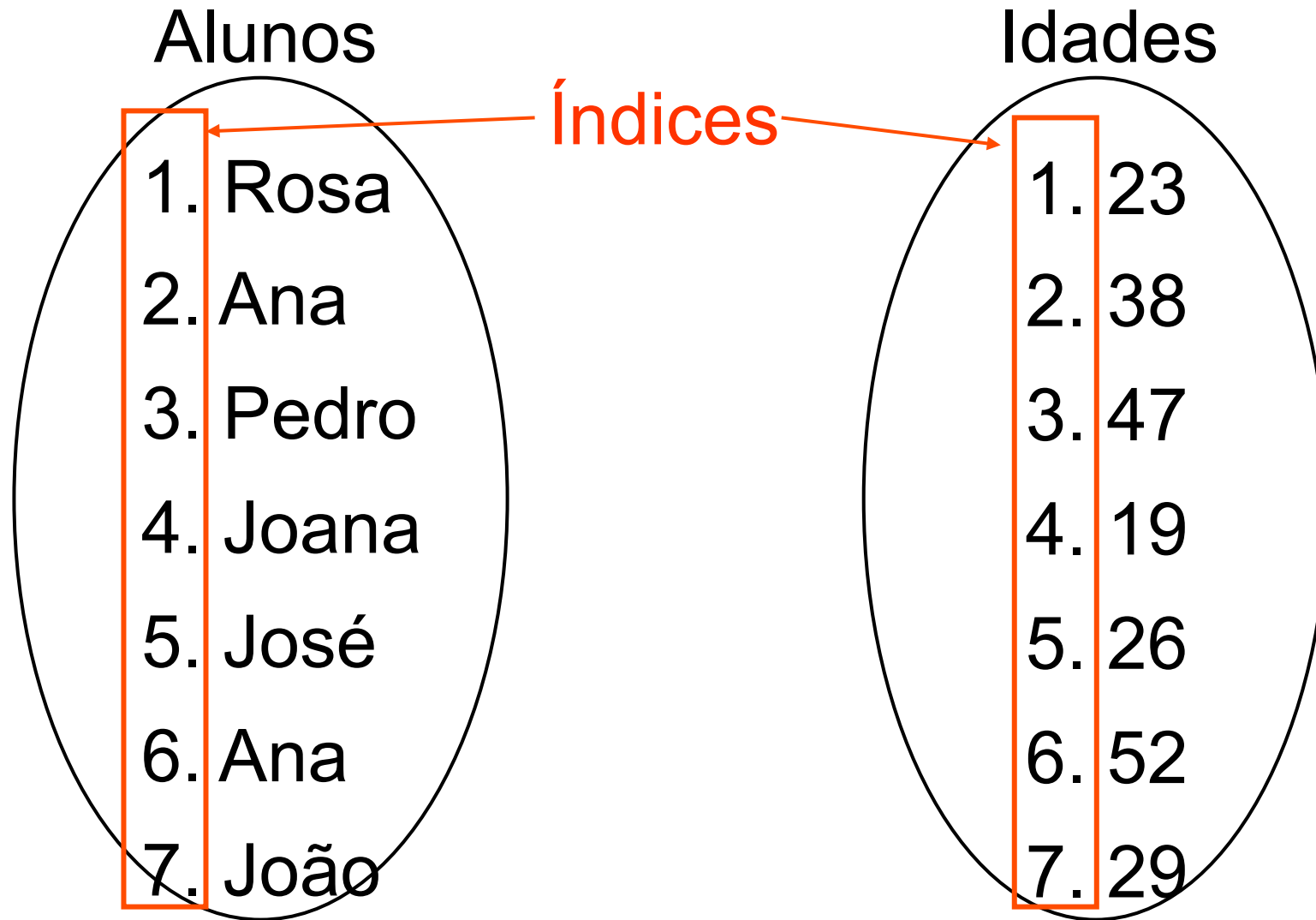


# Estrutura de Dados Unidimensional

---

**ESTRUTURA DE DADOS  
UNIDIMENSIONAL  
=  
CONJUNTO DE DADOS  
onde cada elemento é  
identificado por um  
único índice**

# Estrutura de Dados Unidimensional



# Estrutura de Dados Unidimensional

---

**ESTRUTURA DE DADOS  
UNIDIMENSIONAL**

**=**

**VETOR em Computação**

# Estrutura de Dados Unidimensional

---

Qual o nome e a idade  
do Aluno 5?

# Estrutura de Dados Unidimensional

---



José, 26!

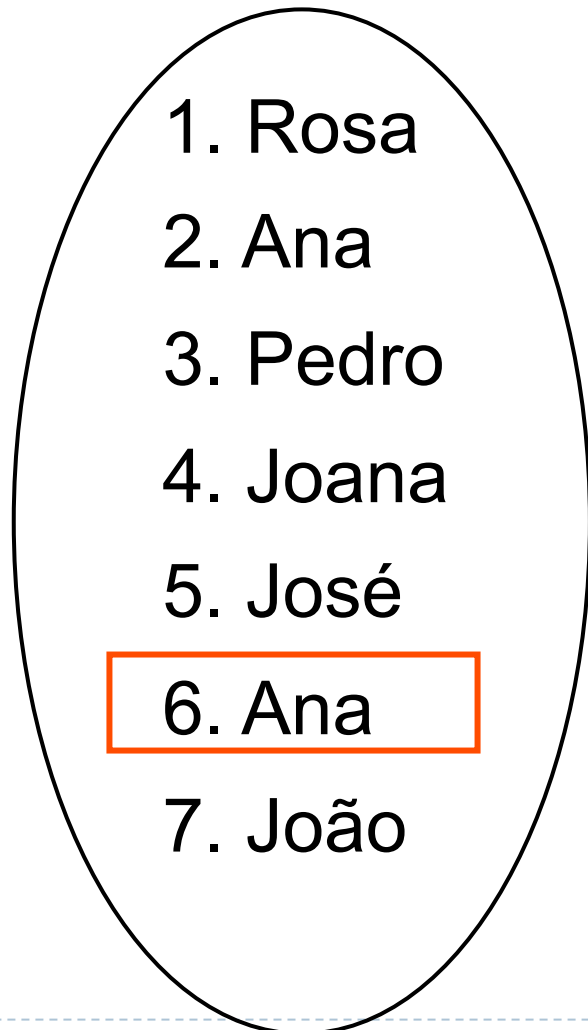
# Estrutura de Dados Unidimensional

---

Como isso funciona  
no Computador?

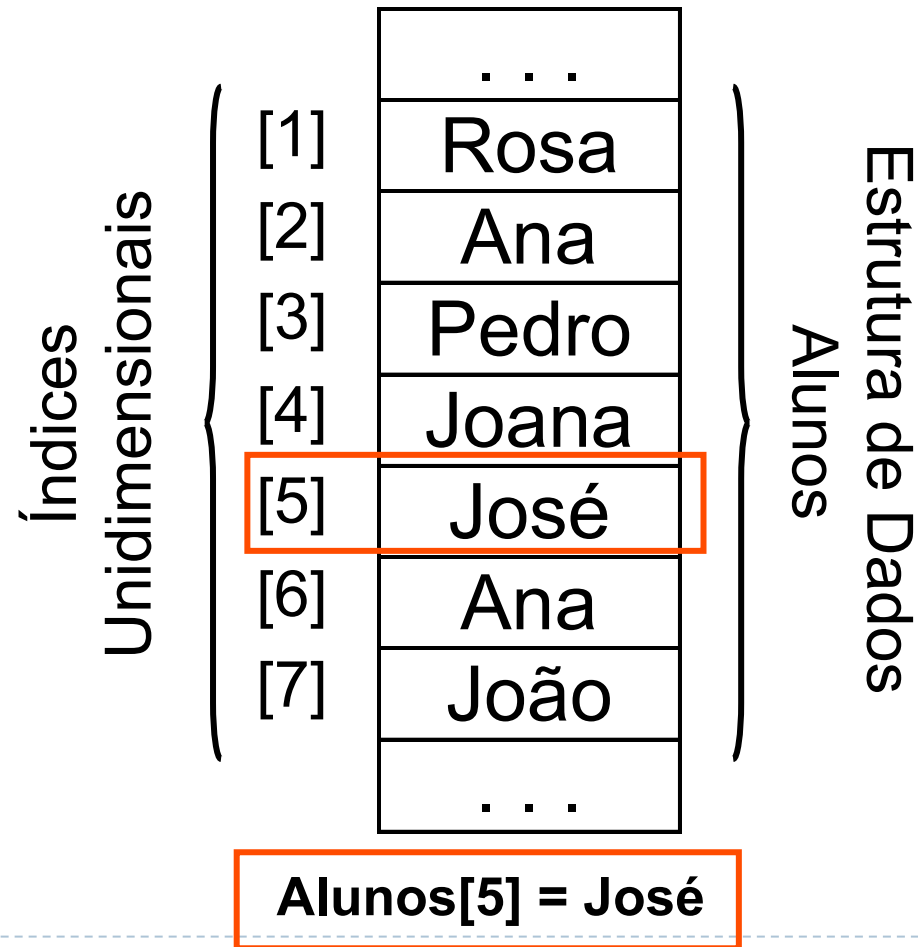
# Estrutura de Dados Unidimensional

## Aluno



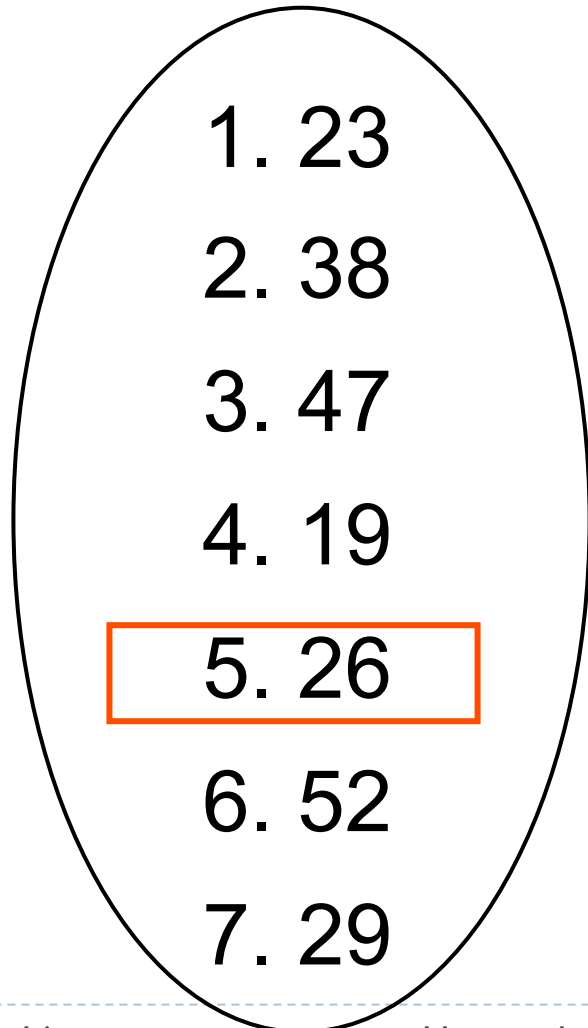
## Memória

### RAM

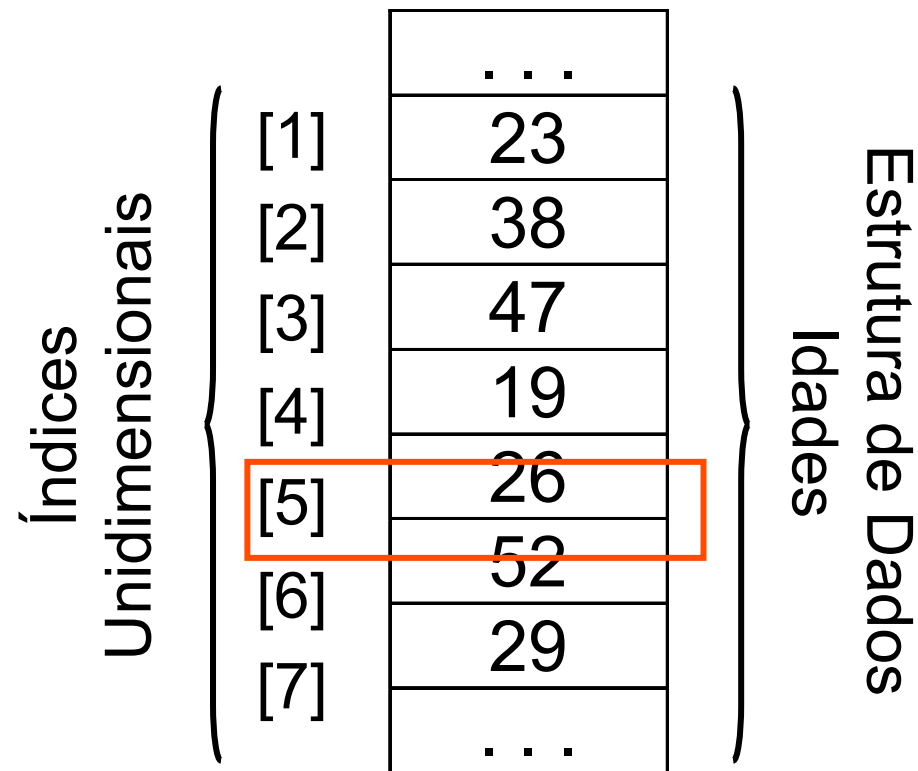


# Estrutura de Dados Unidimensional

Idades



Memória  
RAM



**Idades[5] = 26**



# Estrutura de Dados Unidimensional

---

Como se faz isso  
no Algoritmo?

# Estrutura de Dados Unidimensional

---

- ▶ Variáveis indexadas

- ▶ Sintaxe:

**<tipo> nome\_variavel [índice]**

- ▶ O índice de uma variável indexada precisa ser um inteiro com valor limitado ao número de posições de memória declaradas pela variável
    - ▶ O valor deste inteiro pode ser uma constante, o valor de uma expressão, o valor de uma variável ou o valor retornado por uma função

# Estrutura de Dados Unidimensional

---

- ▶ Variáveis indexadas
  - ▶ Assim, são válidas as expressões:

```
int x[10], i = 5, a;  
  
a = x[3];  
a = x[i];  
a = x[x[a]];  
a = x[random(10)];
```

# Estrutura de Dados Unidimensional

---

- ▶ Variáveis indexadas

- ▶ Atenção!

- Ao utilizar variáveis indexadas, temos que controlar o valor do índice no intervalo 0 a  $n-1$ , onde  $n$  é o número de posições de memória.
    - Caso contrário, podemos ter invasão de memória.

# Estrutura de Dados Unidimensional

---

```
#include <stdio.h>

int main(){
    int matricula[7];
    float notas[7];
    int i;
    for(i=0; i<7; i++){
        printf("Digite matricula: ");
        scanf("%d", &matricula[i]);
        printf("Digite nota: ");
        scanf("%f", &notas[i]);
    }
    printf("Aluno %d tem nota %f", matricula[5], notas[5]);
    system("pause");
    return 0;
}
```

# Estrutura de Dados Unidimensional

---

- ▶ Declaração de vetores
- ▶ Semântica:
  - ▶ São estruturas de dados homogêneas unidimensionais que permitem agrupar diversas informações dentro de uma variável
  - ▶ Estas correspondem a um grupo de posições contínuas na memória que possuem o mesmo nome e o mesmo tipo de dado e são acessadas por um **ÚNICO** índice
  - ▶ Seu tamanho é definido por constantes inteiras e positivas e a definição do seu nome segue as mesmas regras aplicadas para identificadores

# Estrutura de Dados Unidimensional

---

## Nota I

Os vetores matricula e notas têm 7 posições cada. Isto é, cada um equivale a 7 variáveis de mesmo nome, só distinguíveis pelos seus índices. Ou seja, os dois vetores juntos equivalem a criar 14 variáveis, só que é menos trabalhoso!

# Estrutura de Dados Unidimensional

---

E se aumentar de 7 para  
700 Alunos?



# Estrutura de Dados Unidimensional

---

## ► Uso de constantes

```
#include <stdio.h>
#define N 700

int main(){
    int matricula[N];
    float notas[N];
    int i;
    for(i=0; i<=N; i++){
        printf("Digite matricula: ");
        scanf("%d", &matricula[i]);
        printf("Digite nota: ");
        scanf("%f", &notas[i]);
    }
    printf("Aluno %d tem nota %f", matricula[5], notas[5]);
    system("pause");
    return 0;
}
```

# Estrutura de Dados Unidimensional

---

- ▶ A definição de uma constante associa um nome a um valor que não se altera.
- ▶ A definição de constantes melhora a legibilidade do programa e facilita sua manutenção.

# Estrutura de Dados Unidimensional

---

## Nota II

Resolver o problema anterior sem vetores implicaria no árduo trabalho de ter que declarar e manipular 700 variáveis!

Por isso, a solução de certos problemas só é viável usando estruturas de dados!

## Nota III

Não é possível operar com todos os elementos do vetor de uma só vez. Por isso, o correto é acessar cada um de seus elementos isoladamente.

# Estrutura de Dados Unidimensional

---

## Nota IV

O acesso a cada elemento de um vetor é feito pela manipulação do seu índice entre [colchetes]!

# Estrutura de Dados Unidimensional

---

## Exemplo

```
idades[5] = 10;
```

```
...
```

```
printf("%d", idades[5]);
```

```
...
```

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

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

# Estrutura de Dados Unidimensional

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

```
int main() {  
    float notas[6]={2.3, 4.5, 4.7, 3.5, 1.2, 3.8};  
    float inversonotas[6];  
    int i,j;  
    i=5;  
    for(j=0;j<6;j++){  
        inversonotas[j]=notas[i];  
        i--;  
    }  
  
    for(j=0;j<6;j++){  
  
        printf("%.2f \n", inversonotas[j]);  
    }  
    system("pause");  
    return 0;  
}
```

Inicialização de um vetor

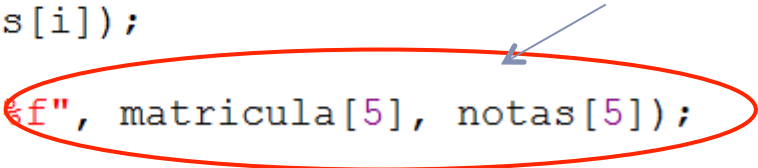
# Estrutura de Dados Unidimensional

---

```
#include <stdio.h>
#define N 7

int main() {
    int matricula[N];
    float notas[N];
    int i;
    for(i=0; i<=N; i++){
        printf("Digite matricula: ");
        scanf("%d", &matricula[i]);
        printf("Digite nota: ");
        scanf("%f", &notas[i]);
    }
    printf("Aluno %d tem nota %f", matricula[5], notas[5]);
    system("pause");
    return 0;
}
```

Acesso a posições específicas de um vetor





# Estrutura de Dados Unidimensional

---

```
#include <stdio.h>

int main() {
    int VetA[4], VetB[4], i, j;
    for(i=0; i<=3; i++) {
        printf("Digite VetA[%d]: ", i);
        scanf("%d", &VetA[i]);
        VetB[i]=1;
        for(j=2; j<=VetA[i]; j++) {
            VetB[i]*=j;
        }
    }
    for(i=0; i<=3; i++) {
        printf("VetB[%d]: %d \n", i, VetB[i]);
    }
    system("pause");
    return 0;
}
```

# Estrutura de Dados Unidimensional

```
#include <stdio.h>

int main() {
    int Fibonnacci[100], i, qtde;
    do{
        printf("Digite quantidade de termos: ");
        scanf("%d", &qtde);
    }while((qtde<=0) || (qtde>100));
    Fibonnacci[0]=Fibonnacci[1]=1;
    if(qtde==1) {

        printf("%d", Fibonnacci[0]);
    }else if(qtde==2) {

        printf("%d ,", Fibonnacci[0]);

        printf("%d ", Fibonnacci[1]);
    }else{
        printf("%d, %d ",Fibonnacci[0],Fibonnacci[1]);
        for (i=2;i<qtde;i++){
            Fibonnacci[i]=Fibonnacci[i-1]+Fibonnacci[i-2];
            printf(", %d ",Fibonnacci[i]);
        }
    }
    printf("\n");
    system("pause");
    return 0;
}
```

# Estrutura de Dados Unidimensional

---

```
#include <stdio.h>

int main() {

    int i, j, d=0, VetA[6], VetB[6], VetC[12];

    for ( i =0; i < 6; i ++ ) {

        printf ( " Vetor %i de A:", i );
        scanf ( "%d", &VetA[i]);
        printf ( " Vetor %i de B:", i );
        scanf ( "%d", &VetB[i]);

    }
    for ( i = 0; i < 6; i ++ ) {
        for ( j = 0; j < 6; j ++ ) {
            if ( VetA[i] == VetB[ j ] ) {
                printf ( " %d e intersecção dos dois \n", VetB[ j ] );
                VetC[d]=VetA[i];
                d++;
            }
        }
    }
    for ( i =0; i < d-1; i ++ ) {
        printf ( " Vetor %i de C: %d \n", i, VetC[i] );
    }

}
```

# Estrutura de Dados Unidimensional

---

```
for ( i =0; i < d-1; i ++ ) {
    for(j=i+1;j<d;j++){
        if(VetC[i]>VetC[j]){
            aux=VetC[i];
            VetC[i]=VetC[j];
            VetC[j]=aux;
        }
    }

    printf ( " Vetor %i de C: %d \n", i, VetC[i] );

}*/

system("PAUSE");
return 0;
}
```

---

# Estrutura de Dados Unidimensional

---

```
#include <stdio.h>

int main() {

    int VetNumeros[10], i, maior, menor;
    for(i=0; i<=9; i++){
        scanf("%d", &VetNumeros[i]);
    }
    maior=VetNumeros[0];
    menor=VetNumeros[0];
    for(i=1; i<=9; i++){
        if(VetNumeros[i]>maior)
            maior=VetNumeros[i];
        else if(VetNumeros[i]<menor)
            menor=VetNumeros[i];
    }
    printf("Maior: %d e Menor: %d", maior, menor);
    system("pause");
    return 0;
}
```

# Estrutura de Dados Unidimensional

---

```
#include <stdio.h>

int main(){
    int num, i;
    printf("Numero:");
    scanf("%d", &num);
    int vetor[num];
    for(i=0; i<num; i++)
        scanf("%d", &vetor[i]);
    for(i=0; i<num; i++)
        printf("%d", &vetor[i]);
    //getch();
    system("pause");
    return 0;
}
```

# Estrutura de Dados Unidimensional

---

```
#include <stdio.h>
#define TAM 6

int main() {
    int VetA[TAM], VetB[TAM], i;
    for(i=0; i<TAM; i++) {
        printf("Digite numero %d: ", i);
        scanf("%d", &VetA[i]);

    }
    for(i=0; i<TAM; i++) {
        VetB[i]=VetA[i]*2;
    }

    for(i=0; i<TAM; i++) {
        printf("VetB[%d]: %d \n", i, VetB[i]);
    }
    system("pause");
    return 0;
}
```

# Estrutura de Dados Unidimensional

---

```
#include <stdio.h>

int main(){
    char letras[6];
    int i, vogais=0, consoante=0;
    for(i=0;i<6;i++){
        printf("Digite Letra: ");
        letras[i]=getche();
        printf("\n");
        letras[i]=toupper(letras[i]);
        if(letras[i]=='A' || letras[i]=='E' || letras[i]=='I' || letras[i]=='O' || letras[i]=='U')
            vogais+=1;
        else
            consoante+=1;
    }
    printf("Vogais %d e Consoante %d \n:", vogais, consoante);

    system("pause");
    return 0;
}
```



# Estrutura de Dados Unidimensional

---

```
#include <stdio.h>

int main() {
    int VetA[8], i, j, aux;
    for (i=0; i<8; i++)
        scanf("%d", &VetA[i]);

    for (i=0; i<7; i++) {
        for (j=i+1; j<8; j++) {
            if (VetA[i]>VetA[j]) {
                aux=VetA[i];
                VetA[i]=VetA[j];
                VetA[j]=aux;
            }
        }
    }
    for (i=0; i<8; i++)
        printf("%d ", VetA[i]);

    system("pause");
    return 0;
}
```

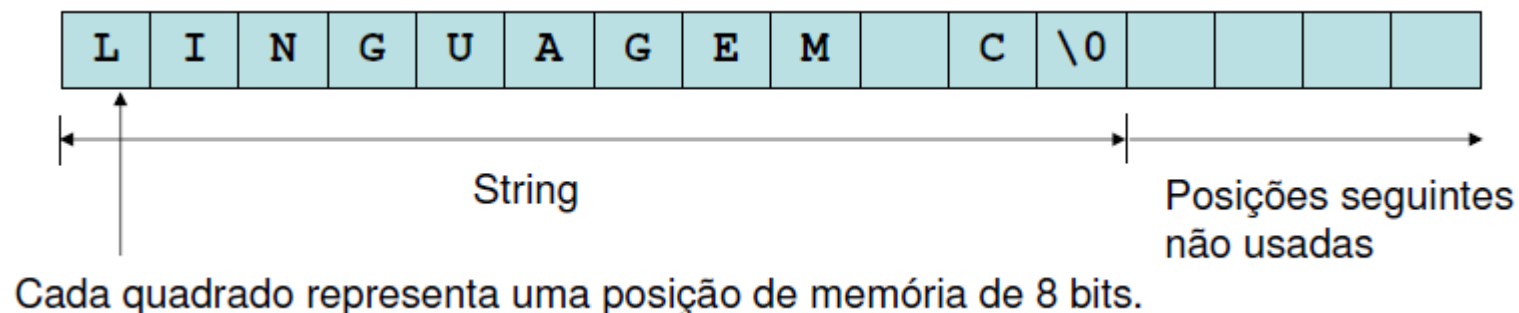
# Representação de strings

---

- ▶ Strings são conjuntos de caracteres e cada caractere é codificado como um inteiro de 8 bits (código ASCII).
- ▶ Assim, cada caractere que compõe uma string pode ser imaginado como inteiro no intervalo  $[-128, 127]$ .
- ▶ Ou se imaginarmos, que o bit de sinal não é utilizado, o caractere pode ser representado por um inteiro no intervalo  $[0, 255]$ .
- ▶ Lembre-se que no código ASCII padrão, cada caractere é codificado usando-se 7 bits (0 a 127).

# Representação de strings

- ▶ Uma string é armazenada em bytes consecutivos de memória.
- ▶ Para identificar o final de uma string, a linguagem C utiliza um caractere especial: `'\0'` (cód.ASCII zero).
- ▶ Exemplo: seja a string “Linguagem C”. Imagine a representação desta string na memória como:



# Representação de strings

---

- ▶ Temos abaixo a declaração da variável texto:

```
char texto[100];
```

- ▶ Se  $n$  é uma constante inteira, então os símbolos  $[n]$  após o nome da variável indicam que ela poderá ocupar até  $n$  posições de memória consecutivas.
- ▶ Logo, a variável texto poderá ocupar até 100 posições do tipo char (ou seja, 100 bytes).
- ▶ Como as posições de memória são consecutivas, cada uma delas pode ser identificada por um índice.

# Representação de strings

---

- ▶ Na linguagem C, os valores dos índices começam sempre em zero.
- ▶ Exemplo: considere a seguinte declaração:

```
char s[20] = "Linguagem C";
```

- ▶ A representação de s pode ser imaginada como:

L	I	N	G	U	A	G	E	M		C	\0			
S[0]	S[1]	S[2]	S[3]	S[4]	S[5]	S[6]	S[7]	S[8]	S[9]	S[10]	S[11]	...		S[19]

- ▶ Observe que s pode ocupar até 20 posições.

# Representação de Strings

---

- ▶ Tipos de strings
  - ▶ Strings constantes
    - ▶ Entre “ ” – (Ex.: `printf( “Joao é bonito!”)`)
  - ▶ Variáveis strings
    - ▶ `char nome[15]`

# Representação de Strings

---

- ▶ Escrita de strings
  - ▶ Comando `printf(<string a ser impressa>)`
    - ▶ Ex.: `printf("Digite nome: ");`
  - ▶ Comando `puts(<string a ser impressa>);`
    - ▶ Ex.: `puts(nome);`

Qual a diferença entre os dois comandos?

O comando `puts()` corresponde ao `printf()`, no entanto, ao imprimir algo na tela, também imprime um `'\n'`

# Representação de Strings

---

- ▶ Leitura de strings

- ▶ Comando `scanf(<string a ser lida>)`

- ▶ Não se usa o ‘&’ para leitura, pois o nome de uma matriz é o seu endereço inicial

- Ex.: `scanf(“%s”, nome);`

- ▶ Comando `gets(<string a ser lida>);`

- ▶ Ex.: `gets(nome);`

Qual a diferença entre os dois comandos?

O comando `scanf()` usa qualquer espaço branco para terminar uma entrada. Não existe forma de digitar um texto de múltiplas palavras em uma variável. Com o `gets()`, isso é permitido.



# Representação de Strings

---

## ► Escrita/Leitura (printf() e scanf())

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

```
<saída>
```

```
int main(){  
    char nome[20];  
    printf("Digite nome: ");  
    scanf("%s", &nome);  
    printf("Saudações! %s", nome);  
    system("pause");  
    return 0;  
}
```

```
Digite nome: Maria das Graças  
Saudações! Maria
```

# Representação de Strings

---

## ► Escrita/Leitura (puts() e gets())

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

```
int main(){  
    char nome[20];  
    puts("Digite nome: ");  
    gets(nome);  
    puts("Saudações!");  
    puts(nome);  
    puts(&nome[3]);  
    system("pause");  
    return 0;  
}
```

```
<saída>
```

```
Digite nome:  
Maria das Graças  
Saudações!  
Maria das Graças  
ia das Graças
```

# Representação de Strings

---

- ▶ Inicializando strings

- ▶ Pode ser visto como caracteres separados

- ▶ `char vogais[ ] = { 'A', 'E', 'I', 'O', 'U' };`

- ▶ Pode ser visto como um conjunto de caracteres formando palavras

- ▶ `char nome[ ] = "Ana";`

# Manipulação de Strings

---

- ▶ Funções fornecidas pelo Compilador 'C'
  - ▶ strlen(), strcat(), strcmp(), strcpy()
  - ▶ Todas estas funções estão dentro da biblioteca string.h

# Manipulação de Strings

---

## ▶ Funções strlen()

- ▶ Aceita um endereço de String como argumento e retorna o tamanho da String até um caractere nulo ( '\0' )
- ▶ O terminador nulo não é contado, ou seja, o tamanho da string deve ser um a mais do que o definido pela função strlen()
- ▶ Como usar: strlen(string);

# Manipulação de Strings

---

## ► Função para saber o tamanho de uma string

```
#include <stdio.h>
#include <string.h>
int main(){
    int total=0;
    char str[100];
    printf("Digite string:");
    gets(str);
    while(str[total]!='\0'){
        total+=1;
    }
    printf("Tamanho da string é: %d", total);
    system("pause");
    return 0;
}
```

# Manipulação de Strings

---

## ► Função strlen()

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

int main(){
    int tam;
    char str[100];
    printf("Digite string:");
    gets(str);
    tam=strlen(str);
    printf("Tamanho da string é: %d", tam);
    system("pause");
    return 0;
}
```

# Manipulação de Strings

---

## ► Função strlen()

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

int main() {
    int i;
    char frase[100];
    printf("Digite string:");
    gets(frase);
    for(i=0; i<=strlen(frase); i++) {
        if((frase[i]=='a') || (frase[i]=='e') || (frase[i]=='i') ||
            (frase[i]=='o') || (frase[i]=='u'))
            frase[i]=toupper(frase[i]);
    }
    puts(frase);
    system("pause");
    return 0;
}
```

---



# Manipulação de Strings

---

- ▶ **Função strcat()**
  - ▶ Concatena duas strings (uma no final da outra)
  - ▶ Recebe dois endereços de strings como argumento e copia a segunda string no final da primeira, sendo que a origem permanece a mesma
  - ▶ Ter cuidado em deixar espaço em branco, pois a função não trata isso
  - ▶ Como usar: `strcat(Destino_string, Origem_string);`

# Manipulação de Strings

---

## ► Função strcat()

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

int main() {
    char nome[] = {"Carlos "};
    char sobrenome[] = {"Manuel"};
    strcat(nome, sobrenome);
    puts(nome);
    system("pause");
    return 0;
}
```

---

# Manipulação de Strings

---

## ► Função strcpy()

- Copia uma string em outra
- Prestar atenção porque não concatena, e sim sobrescreve a string.
- Recebe dois endereços de strings como argumento e copia a segunda string na primeira, sendo que a origem não permanece mais a mesma
- Como usar: `strcpy(Destino_string, Origem_string);`

# Manipulação de Strings

---

## ► Função strcpy()

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

int main(){
    char str1[20], str2[20], str3[80];
    puts("Digite string 1: ");
    gets(str1);
    strcpy(str2, str1);
    strcpy(str3, "Voce digitou: ");
    puts(str3);
    puts(str1);
    system("pause");
    return 0;
}
```

# Manipulação de Strings

---

## ► Função strcpy()

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

int main(){
    char frase[80];
    int pos;
    puts("Digite string: ");
    gets(frase);
    puts("Digite posicao: ");
    scanf("%d", &pos);
    strcpy(&frase[pos], &frase[pos+1]);
    puts(frase);
    system("pause");
    return 0;
}
```

# Manipulação de Strings

---

## ▶ Função strcmp()

- ▶ Compara duas strings, recebendo o endereço de memória delas como argumentos
- ▶ Recebe dois endereços de strings como argumento e retorna um valor inteiro

**< 0, se String1 < String2;**

**= 0, se String1 = String2;**

**> 0, se String1 > String2;**

- ▶ Como usar: strcpy(Destino\_string, Origem\_string);

# Manipulação de Strings

---

## ► Sem a função strcmp()

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

int main() {
    char resp[]="branco";
    char r1[40];
    puts("Digite a cor do cavalo branco de Napoleão");
    gets(r1);
    while(r1!=resp) {
        puts("Resposta errada!!!! Tente novamente:");
        gets(r1);
    }
    puts("Resposta Correta");
    system("pause");
    return 0;
}
```

ESTE PROGRAMA NÃO ESTÁ CORRETO, POIS resp1 e resp SÃO ENDEREÇOS DE MEMÓRIA, ENTÃO: resp1!=resp SEMPRE

# Manipulação de Strings

---

## ► Função strcmp()

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

int main() {
    char resp[]="branco";
    char r1[20];
    puts("Digite a cor do cavalo branco de Napoleão");
    gets(r1);
    while(strcmp(r1,resp)!=0){
        puts("Resposta errada!!!! Tente novamente:");
        gets(r1);
    }
    puts("Resposta Correta");
    system("pause");
    return 0;
}
```



# Manipulação de Strings

---

## ► Função strcmp()

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

int main() {
    printf("%d \n", strcmp ("A", "A"));
    printf("%d \n", strcmp ("A", "B"));
    printf("%d \n", strcmp ("B", "A"));
    printf("%d \n", strcmp ("A", "E"));
    system("pause");
    return 0;
}
```

# Lista de Exercícios (Vetores)

---

- I. Uma seqüência de números gerados aleatoriamente para um dado de 6 faces é considerada adequada se qualquer valor não aparecer mais do que 2 lançamentos sucessivos. Por exemplo:

Seq. 1→	2	1	3	1	4	1	5	2	1
---------	---	---	---	---	---	---	---	---	---

Seq. 2→	5	4	6	2	1	3	3	3	4
---------	---	---	---	---	---	---	---	---	---

A Seq. 1 é considerada apesar do número 1 aparecer mais de uma vez, pois ele não aparece em mais de 2 lançamentos sucessivos. A Seq. 2 não é considerada adequada, pois o valor 3 aparece em mais de 2 lançamentos sucessivos. Construa um programa que verifica se uma seqüência aleatória é adequada ou não para a simulação de 100 lançamentos de um dado de 6 faces.

# Lista de Exercícios (Vetores)

---

2. Elaborar um sistema de reservas de companhias áreas em que os assentos de cada voo são representados por um vetor. Inicialmente todos os assentos estão vagos e todos os elementos do vetor possuem valor 0. A cada reserva realizada a posição do vetor correspondente ao assento deverá apresentar o valor 1. Assim, para cada usuário que utiliza o sistema deverá ser apresentado o seguinte menu:

Favor digitar:

- 1 – Para verificar ocupação do avião.
- 2 – Realizar reserva de um assento.
- 3 – Cancelar reserva de um assento.

Caso o usuário digite a opção 1 deverá ser mostrada na tela quais posições do avião estão disponíveis (lembre-se que a posição  $v[0]$  do vetor corresponde ao assento número 1 do avião e assim por diante). Caso a opção 2 seja selecionada, então, deverá ser verificado se a posição está ocupada ou não. Se estiver, imprimir uma mensagem falando isso. Caso contrário, realizar a reserva e mostra a nova ocupação do avião. Por último, caso a opção 3 seja selecionada, então, verificar se o assento está ocupado. Se estiver, então, tornar o assento vazio e mostrar a nova ocupação do avião. Caso contrário, imprimir uma mensagem de que o assento já está disponível.

# Lista de Exercícios (Vetores)

---

3. Escreva um algoritmo que leia um vetor de 20 posições e mostre-o. Em seguida, troque o primeiro elemento com o último, o segundo com o penúltimo, o terceiro com o antepenúltimo, e assim sucessivamente. Mostre o novo vetor depois da troca.
4. Faça um algoritmo que leia um vetor de 500 posições de números inteiros e divida todos os seus elementos pelo maior valor do vetor. Mostre o vetor após os cálculos.
5. Uma locadora de vídeos tem guardada, em um vetor de 50 posições, a quantidade de filmes retirados por seus clientes durante o ano de 2004. Agora, esta locadora está fazendo uma promoção e, para cada 10 filmes retirados, o cliente tem direito a uma locação grátis. Faça um algoritmo que crie um outro vetor contendo a quantidade de locações gratuitas a que cada cliente tem direito.
6. Faça um algoritmo que leia dois vetores (A e B) de 50 posições de números inteiros. O algoritmo deve, então, subtrair o primeiro elemento de A do último de B, acumulando o valor, subtrair o segundo elemento de A do penúltimo de B, acumulando o valor, e assim por diante. Mostre o resultado da soma final.

# Lista de Exercícios (Vetores)

11. A **distância de Hamming** fornece o número de posições de dois vetores binários que apresentam valores diferentes. Por exemplo, sejam os vetores v e t de tamanho 10:

Valor	1	0	1	0	0	0	0	0	0
Índice	v[0]	v[1]	v[2]	v[3]	v[4]	v[5]	v[6]	v[7]	v[8]
Comparação	↔	↔	↔	↔	↔	↔	↔	↔	↔
Valor	1	1	0	0	1	0	1	0	0
Índice	t[0]	t[1]	t[2]	t[3]	t[4]	t[5]	t[6]	t[7]	t[8]
Diferença	0	1	1	0	1	0	1	0	0

Para os valores dos vetores binários v e t apresentados acima a **distância de Hamming** é igual a quatro, pois os elementos v[1], v[2], v[4] e v[6] são diferentes (ou seja, quatro componentes dos vetores diferem entre si). Construa um trecho de código que fornece a **distância de Hamming** para dois vetores binários v e t de tamanho n.

# Lista de Exercícios (Vetores)

---

12. Criar um programa que gera aleatoriamente os valores das notas de uma turma com 70 alunos e depois ordena as notas pelo Bubble Sort Simplificado. O Bubble Sort Simplificado consiste em ordenar em ordem crescente os elementos de um vetor. Para tanto, os valores maiores “afundam” (submergem) para a parte de baixo do vetor. O Bubble Sort Simplificado varre  $(n-1)$  vezes todo o vetor, comparando os elementos dois a dois,  $(n-1)$  vezes. A cada varredura, se um par está em ordem crescente, nada é feito. Caso contrário, os elementos no vetor são permutados. Um exemplo de execução é dado por:

```
Vetor inicial: 8 5 1  
Varredura 1:  
Comparacao 1: 8 5 1 → 5 8 1  
Comparação 2: 5 8 1 → 5 1 8  
  
Varredura 2:  
Comparacao 1: 5 1 8 → 1 5 8  
Comparação 2: 1 5 8 → 1 5 8
```

# Lista de Exercícios (Strings)

---

I. Criar um algoritmo que funcione através do MENU a seguir:

<p style="text-align: center;">#MENU#</p> <p>1 – Imprime o comprimento da frase 2 – Imprime os dois primeiros e os dois últimos caracteres da frase 3 – Imprime a frase espelhada 5 – Imprime o número de palavras existentes na frase 4 – Termina o algoritmo</p> <p>OPÇÃO:</p>
--

2. Uma palavra é denominada um **PALÍNDROMO** se for invertida e a leitura da mesma permanecer sem nenhuma alteração. Algumas palavras que são palíndromos são: **ABA, RADAR, RETER, REVER, RIR, ROTOR**, dentre outras. Construir um programa que detecte se uma palavra (string) digitada pelo usuário é ou não um palíndromo.

# Lista de Exercícios (Strings)

---

3. Existem PALÍNDROMOS que são formados por frases tais como:

(i) Socorram-me subi no onibus em Marrocos.

(ii) Omitiram radar maritmo.

Para mais exemplos consulte o seguinte endereço da internet: <http://pt.wikipedia.org/wiki/Pal%C3%ADndromo>.

Construir um programa que ignore os espaços e o caractere ‘-’ ao verificar se uma frase é um palíndromo.

4. Construir um programa que leia uma string S1 e uma substring S2 e depois forneça o número de ocorrências da substring S2 em uma string S1. Um exemplo de resposta que o programa deve fornecer é dado abaixo:

Digite a string: O rato roeu a roupa do rei de Roma e a rainha de raiva roeu o rato.  
Digite a substring: ra  
Numero de ocorrências: 4



# Lista de Exercícios (Strings)

---

5. Construir um programa que seja capaz de embaralhar uma string s1 com uma string s2 e colocar o resultado em uma string s3. Para embaralhar S1 com S2 é necessário preencher os índices pares de S3 com os elementos de S1 e os ímpares com os elementos de S2 até que os elementos de uma das duas strings termine e os demais elementos de S3 serão preenchidos com os elementos da string restante. Por exemplo:

Digite a string s1: local.  
Digite a string s2: misterio.  
Nova string s3: lmoicsatlerio.

# Lista de Exercícios (Strings)

---

6. Uma string é utilizada para representar uma das fitas de uma cadeia de DNA. Para tanto, as bases Adenina, Guanina, Citosina, Timina e Uracila são representadas pelas letras A, G, C, T e U, respectivamente. Deseja-se construir um programa que dada uma sequência de DNA é fornecida a sequência de RNA-m equivalente de acordo com a transformação indicada na Tabela 1.

**TABELA 1**

DNA	RNA-m
A	U
G	C
C	G
T	A

Teste o seu programa para a seguinte fita de uma cadeia de DNA:

A	T	C	C	G	T	T	A	A
---	---	---	---	---	---	---	---	---

# Lista de Exercícios (Strings)

---

7. Construir um programa que seja capaz de concatenar duas strings (s1 e s2) em uma terceira string s3. Por exemplo:

Digite a string s1: Quem canta os males espanta.  
Digite a string s2: Ha males que vem para o bem.  
Nova string s3: Ha males que vem para o bem. Quem canta os males espanta

8. Entrar com profissão de várias pessoas (loop infinito) e imprimir quantos são advogados (considerar advogado,ADVOGADO,Advogado).
9. Entrar com um nome e imprimi-lo tantas vezes quantos forem seus caracteres.
10. Criar um algoritmo que entre com uma palavra e imprima conforme o exemplo a seguir:

Palavra: PAZ

Impressão: P

A

Z

# Lista de Exercícios (Strings)

---

11. Faça um algoritmo em que o usuário digite uma senha com tamanho 8, onde serão combinados caracteres e números, mas a medida que o mesmo for digitando esta senha, ao invés de aparecer o que está sendo digitado, na tela seja mostrado \*\*\*\*\* (hide character).

# Referências

---

- ▶ Medina, Marco; Fertig, Cristina. Algoritmos e Programação: teoria e prática. São Paulo: Novatec Editora, 2006.
- ▶ Lopes, Anita; Garcia, Guto. Introdução à Programação: 500 algoritmos resolvidos. Rio de Janeiro: Editora Campus, 2002.
- ▶ Mizrani, Victorine Viviane. Treinamento em Linguagem C, Módulo I. Editora Makron Books.
- ▶ Transparências modificadas do professor Dr. Flavio Luiz Cardeal Pádua, do Centro Federal de Educação Tecnológica de Minas Gerais
- ▶ Transparências modificadas do professor Robson Fidalgo, da UFRPE.