

# *Tecnologia em Análise e Desenvolvimento de Sistemas - TADS*

## ***Estrutura de Dados***

*TADS / IFRS - 2022-2*

***Prof. Luciano Vargas Gonçalves***

*E-mail: [luciano.goncalves@riogrande.ifrs.edu.br](mailto:luciano.goncalves@riogrande.ifrs.edu.br)*





## Aula 2 – Ponteiros e Vetores

# Antes vamos ver Memória RAM

- **Endereços da memória:**

- A memória RAM de qualquer computador é uma sequência de bytes (cada linha da tabela é um Byte). Ex: 1 Pente de 1GB  $\approx$  1bilhão de Bytes ou linhas na tabela.
- Os bytes são numerados sequencialmente. O número de um byte é o seu **endereço (= address) escrito em decimal ou hexadecimal.**

- *Exemplo:*

- *Endereço (e943ab) de um Byte da memória*
    - *Pode armazenar 8 Bits de informação*

Endereço	Valor (Célula)
e943ab	
e943ac	
e943ad	
e943ae	
e943af	
e943b0	
e943b1	
e943b2	

Memória Ram -

# Memória RAM

- **Célula de Memória:**

- **Endereço de memória**

- Identifica uma célula de memória;
    - O seu uso é controlado pelo Sistema Operacional dentro dos processos do sistema;

- **Valor:**

- Conteúdo ou informação armazenado na memória, o valor dado a alguma informação (dados, ou programas);

Endereço	Valor
e943ab	
e943ac	
e943ad	
e943ae	
e943af	
e943b0	
e943b1	
e943b2	

Memória RAM - Ex

# Memória RAM

- Tipos de dados e Espaço ocupado
  - Um char ocupa 1 byte (8bits).
  - Um int ocupa 4 bytes (32bits) em alguns computadores e 8 bytes em outros (sizeof (int)).
  - Um double ocupa usualmente 64 bits (sizeof (double)).
  - O endereço da variável(informação) é o endereço do primeiro byte alocado.

Tipo inteiro	Tamanho	Valor mínimo	Valor Máximo
signed char	8	−128	127
short int	16	−32.768	32.767
int	32	−2.147.483.648	2.147.483.647
long int	32	−2.147.483.648	2.147.483.647
long long int	64	−9.223.372.036.854.775.808	9.223.372.036.854.775.807
unsigned char	8	0	255
unsigned short int	16	0	65535
unsigned int	32	0	4.294.967.295
unsigned long int	32	0	4.294.967.295
unsigned long long int	64	0	18.446.744.073.709.551.615

Bits

# Memória RAM

- Tipos de Dados

```
int v_inteiro = 10;
double v_double = 15.73;
char v_char = 'A';
long int v_longInt = 12;

printf("0 Inteiro %d ocupa na memoria %d Bytes \n",v_inteiro,sizeof(v_inteiro));
printf("0 Double %f ocupa na memoria %d Bytes \n",v_double,sizeof(v_double));
printf("0 Char %c ocupa na memoria %d Bytes \n",v_char,sizeof(v_char));
printf("0 LONG INT %d ocupa na memoria %d Bytes \n",v_longInt,sizeof(v_longInt));
```

```
Aula2
0 Inteiro 10 ocupa na memoria 4 Bytes
0 Double 15.730000 ocupa na memoria 8 Bytes
0 Char A ocupa na memoria 1 Bytes
0 LONG INT 12 ocupa na memoria 8 Bytes

--- TIPO ---|--- BYTES ---
char .....: 1 bytes
short.....: 2 bytes
int.....: 4 bytes
long.....: 8 bytes
float .....: 4 bytes
double.....: 8 bytes
long double.: 16 bytes

Process returned 0 (0x0)   execution time : 0.003 s
Press ENTER to continue.
```

4 bytes {

8 bytes {

1 bytes {

Endereço	Valor	Tipo de dado
e943a9	10	v_inteiro
e943ac		
e943ad	15.73	v_double
e943b4		
e943b5	'A'	v_char
e943b6		

# Memória RAM

- ***Operador '&', copia um endereço de memória***


- Operador '&' copia o endereço de uma variável que se encontra na memória;
- Operador '%x' ou '%p' imprimem um endereço;

```
#include <stdio.h>
int main()
{
    system("clear");
    int x = 10;
    double y = 20, z = 30;

    printf("\nEndereco de X = %x e o valor X= %d \n",&x,x);
    printf("\nEndereco de Y = %x e o valor Y= %f \n",&y,y);
    printf("\nEndereco de Z = %x e o valor Z= %f \n",&z,z);

    exit(0);
}
```

EndereçoY	Valor	Variável
e943a0	10	x
e943ad	20	y
e943b5	30	z
e943b6		
e943b7		
e943b8		



# Memória RAM

- **Operador '&'**

```
#include <stdio.h>
int main()
{
    system("clear");
    int x = 10;
    double y = 20, z = 30;

    printf("\nEndereco de X = %x e o valor X= %d \n",&x,x);
    printf("\nEndereco de Y = %x e o valor Y= %f \n",&y,y);
    printf("\nEndereco de Z = %x e o valor Z= %f \n",&z,z);

    exit(0);
}
```

Endereco de X = 4d984464 e o valor X= 10

Endereco de Y = 4d984468 e o valor Y= 20.000000

Endereco de Z = 4d984470 e o valor Z= 30.000000

Saída no terminal



# Ponteiros em C

- **Ponteiros**


- Os conceitos de endereço de memória e ponteiro são fundamentais em qualquer linguagem de programação, embora fiquem ocultos em algumas delas.
- Em C, esses conceitos são explícitos. O conceito de ponteiro não é fácil e, é preciso fazer algum esforço para dominá-lo.
- ***São variáveis capazes de armazenar apenas endereços de memória (hexadecimal);***
- ***Podem Acessar o conteúdo de uma posições de memórias indiretamente, apenas com o endereço da posição;***
- ***Podem alterar uma informação de uma posição de memória;***

# Ponteiro

- **Definição:**

- **Ponteiro (\*P) é uma variável com a capacidade de armazenar um endereço de memória (hexadecimal);**
- Só pode armazenar endereços de memória;
  - Na declaração usa o '\*' para indicar um ponteiro:
    - O ponteiro *tem o mesmo Tipo* da variável referenciada;
      - Ex: Variável **inteira** "x";
        - Poderá ser acessada por um ponteiro \*px também **inteiro**;
        - Exemplo:
          - **\*px = &x;**
            - Ponteiro \*px recebe e armazena o endereço da variável "x".

Endereço	Valor	Variável
e943ab	10	x
e943ac		
e943ad	Ponteiro	
e943ae		
e943af	e943ab	*px
e943b0		
e943b1		
e943b2		



# Ponteiro

- **Operadores ‘\*’, ‘&’, ‘%x’ ou ‘%p’**

- Operador **‘\*’** usado para declarar o ponteiro, e também para consultar o conteúdo do endereço referenciado;
- Operador **‘&’** copia o endereço de uma variável;
- Operador **‘%x’** ou **‘%p’** imprimem um endereço;

```
int x =10;
int *px;    //declaração
px = &x;    //atribuição/cópia do endereço
printf("Endereço de x = %p e o valor X=%d \n",px,*px);
```

Endereço	Valor	Variável
e943ab	10	x
e943ac		
e943ad	Ponteiro	
e943ae		
e943af		*px
e943b0	e943ab	
e943b1		
e943b2		

# Ponteiros

- Um ponteiro pode ter um valor especial - *NULL (nada)*
  - O ponteiro ***\*p = Null***, indica que o ponteiro inicia sem o apontamento para uma posição de memória;

Endereço	Valor	Variável
e943ab		
e943ac		
e943ad		
e943ae		
e943af	NULL	*px
e943b0		
e943b1		
e943b2		

# Ponteiros

- **Alocação de memória direta “malloc”**

- Reserva uma posição de memória e retorna o endereço alocado. Faz-se necessário CASTING.
- Precisa definir o tamanho de memória a ser ocupada (“sizeof”)
- Espaço de memória sem uma variável associada;

```
int *px;    //declaração
px = (int*)malloc(sizeof(int)); //alocar memória
*px = 20;
printf("\nEndereco de x = %p e o valor X=%d \n",px,*px);
free(px); //desalocar memória
exit(0);
```

Casting para o tipo Inteiro

Endereço	Valor	Variável
e943ab	20	
e943ac		
e943ad		
e943ae		
e943af	e943ab	px
e943b0		
e943b1		
e943b2		

# Ponteiros

- Alocação de memória “malloc”

- Não existe uma variável associada a posição de memória;
- Necessita validar a alocação de memória (IF);

```
int *px;    //declaração
px = (int*)malloc(sizeof(int)); //alocar memória

Teste → if(px == NULL){ //validar alocação
    printf("Erro - Memória Insuficiente.\n");
    exit(0);
}
*px = 20;
printf("\nEndereco de x = %p e o valor X=%d \n", px, *px);
free(px); //desalocar memória
exit(0);
```

**Comando “Free” libera a memória**

Endereço	Valor	Variável
e943ab	20	??
e943ac		
e943ad		Ponteiro
e943ae		
e943af	e943ab	px
e943b0		
e943b1		
e943b2		

Ponteiro direto, sem variável

# Ponteiros em C

- O segundo argumento da função ***scanf*** é o endereço da posição na memória, onde devem ser depositados as informações lidas do dispositivo entrada padrão:

```
int i;  
scanf ("%d", &i);
```

Endereço onde será armazenada a informação

# Exercícios

- 1) Crie um programa em C, para ler duas variáveis inteira (A,B), referente a idade de duas pessoas, após crie uma variável S e um ponteiro para \*ps. Armazene a soma dos valores A e B em S, com o uso do ponteiro \*ps, Após:
  - 1) Mostre o resultado armazenado em S.
  - 2) Mostre o endereço armazenado pelo ponteiro \*ps e o endereço de S;
  - 3) Mostre o endereço do ponteiro \*ps, mostre o conteúdo referenciado por \*ps;
  - 4) Mostre o endereço das variáveis A e B;



# Exercícios

- 2) Crie um programa em C, para ler duas variáveis inteira (A,B), referente a idade de duas pessoas. Crie um ponteiro para \*ps. Armazene a soma dos valores A e B em um espaço de memória alocado dinamicamente (use Malloc) e referenciada pelo ponteiro \*ps, Após:
- 1) Mostre o resultado de A+B;
  - 2) Mostre o endereço armazenado no ponteiro \*Ps;
  - 3) Mostre o endereço do ponteiro \*Ps;
  - 4) Mostre o endereço das variáveis A e B;

# Exercícios

3) Crie um programa em C, para ler duas variáveis double (A,B), referente ao peso de duas pessoas;

Aloque uma porção de memória do tipo double, atribua o endereço da memória alocada ao ponteiro \*Ps, para armazenar a soma dos valores A e B;

1) Mostre o resultado de  $A + B$ ;

2) Mostre o endereço armazenado no ponteiro \*Ps, e o valor referenciado;

3) Crie uma variável double C, e copie o valor armazenado no ponteiro \*Ps para a variável C. Mostre o valor de C;

# Exercícios

- 4) Com base no exercício anterior (3), copie o endereço armazenado no ponteiro \*Ps, para o ponteiro \*Qs;  
Some 100 ao valor referenciado no ponteiro \*Qs;  
Copie o valor do ponteiro \*Qs para a variável A;  
Atribua o endereço da variável B, ao ponteiro \*Qs;  
Use o ponteiro \*Qs, para subtrair 10 do valor da variável B;  
Mostre o valor de B.

# Ponteiros em C

- Há vários tipos de ponteiros:
  - ponteiros para caracteres, para inteiros, para registros etc.
  - O computador precisa saber de que tipo de ponteiro você está falando.
  - Para declarar um ponteiro “\*p” para um inteiro, diga
    - `int *p;`
  - Para declarar um ponteiro “\*p” para um registro ou estrutura, diga
    - `struct tipo *p;`
  - Um ponteiro de ponteiro “\*\*r” um ponteiro que apontará para outro ponteiro;
    - `Ponteiro de Ponteiro >> int **r;`

# Ponteiros em C

- Exemplo 1:

- Troca da valores entre duas variáveis x,y;

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

- ```
int main()
```

- ```
{
```

- ```
    system("clear");
```

- ```
    int x =10, y =20, aux;
```

- ```
    int *px, *py;
```

- ```
    px = &x;
```

- ```
    py = &y;
```

- ```
    printf("\n Valor X= %d e Y= %d\n",x,y);
```

- ```
    aux = *px;
```

- ```
    *px = *py;
```

- ```
    *py = aux;
```

- ```
    printf("\n Valor X= %d e Y= %d\n",x,y);
```

- ```
    exit(0);
```

- ```
}
```

troca



Valor X= 10 e Y= 20

Valor X= 20 e Y= 10

Troca efetuada

# Ponteiros (→) em C

- Ponteiro de Ponteiro (\*\*ppx)

Ponteiro (ppx) → Ponteiro (px) → variável (x)  
\*\*ppx → \*px → x

```
4  system("clear");
5  int x =10, y =20;
6  int *px, *py, **ppx;
7
8  px = &x;
9  py = &y;
10 ppx = &px;
11
12 printf("\n Valor X= %d ",x);
13 printf("\n Valor X= %d ",*px);
14 printf("\n Valor X= %d ",**ppx);
15
16 printf("\n Endereço X = %p",&x);
17 printf("\n Endereço apontado por px = %p",px);
18 printf("\n Endereço apontado por ppx =%p \n",ppx);
--
```

| Endereço | Valor  | Variável |
|----------|--------|----------|
| e943ab   | 10     | x        |
| e943ac   |        |          |
| e943ad   | 20     | y        |
| e943ae   |        |          |
| e943af   | e943ab | px       |
| e943b0   | e943ad | py       |
| e943b1   | e943af | ppx      |
| e943b2   |        |          |

# Ponteiros em C

- Ponteiro de Ponteiro (\*\*ppx)

```
system("clear");  
int x =10, y =20;  
int *px, *py, **ppx;
```

```
px = &x;  
py = &y;  
ppx = &px;
```

```
printf("\n Valor X= %d ",x);  
printf("\n Valor *PX= %d ",*px);  
printf("\n Valor *PPX= %d ",**ppx);
```

```
printf("\n Endereço X = %p ",&x);  
printf("\n Endereço apontado por PX = %p ",px);  
printf("\n Endereço apontado por PPX =%p \n",ppx);
```

Valor X= 10  
Valor \*PX= 10  
Valor \*PPX= 10

Acesso a mesma  
posição de  
memória

Endereço X = 0x7fff3dbfe388  
Endereço apontado por PX = 0x7fff3dbfe388  
Endereço apontado por PPX =0x7fff3dbfe390

Execução algoritmo

# Ponteiro de Ponteiro \*\*ppx

```
system("clear");  
int x =10, y =20;  
int *px, *py, **ppx;
```

```
px = &x;  
py = &y;  
ppx = &px;
```

```
printf("\n Valor X= %d e Y= %d",x,y);  
printf("\n Valor *px = %d ",*px);  
printf("\n Valor **ppx= %d ",**ppx);  
printf("\n Endereço X = %p e Y = %p",&x,&y);  
printf("\n Endereço apontado por px = %p ",px);  
printf("\n Endereço de px = %p ",&px);  
printf("\n Endereço apontado por ppx =%p \n",ppx);
```

```
px = &y;
```

```
printf("\n Valor X= %d e Y= %d",x,y);  
printf("\n Valor *px= %d ",*px);  
printf("\n Valor **ppx= %d ",**ppx);  
printf("\n Endereço X = %p e Y = %p",&x,&y);  
printf("\n Endereço apontado por px = %p ",px);  
printf("\n Endereço de px = %p ",&px);  
printf("\n Endereço apontado por ppx =%p \n",ppx);
```

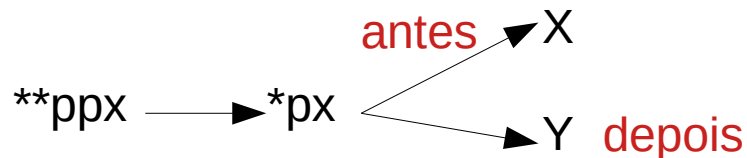
Troca no ponteiro \*p causa troca no \*\*r

Valor X= 10 e Y= 20  
Valor \*px = 10  
Valor \*\*ppx= 10  
Endereço X = 0x7ffedaef7848 e Y = 0x7ffedaef784c  
Endereço apontado por px = 0x7ffedaef7848  
Endereço de px = 0x7ffedaef7850  
Endereço apontado por ppx =0x7ffedaef7850

Valor X= 10 e Y= 20  
Valor \*px= 20  
Valor \*\*ppx= 20  
Endereço X = 0x7ffedaef7848 e Y = 0x7ffedaef784c  
Endereço apontado por px = 0x7ffedaef784c  
Endereço de px = 0x7ffedaef7850  
Endereço apontado por ppx =0x7ffedaef7850

\*px passa a apontar para Y

Execução algoritmo - Troca





# Passagem por Valor

- Troca de valores entre duas variáveis x,y;

```
#include <stdio.h>
void funcaoTroca(int px, int py){
    int aux = px;
    px = py;
    py = aux;
}
```

```
int main()
{
    int x =10, y =20;
    printf("\n Valor X= %d e Y= %d\n",x,y);

    funcaoTroca(x,y); //passagem por valor

    printf("\n Valor X= %d e Y= %d\n",x,y);

    exit(0);
}
```

Valor X=10 e Y=20

Valor X=10 e Y=20

Troca não realizada

# Passagem por Valor

- Troca de valores entre duas variáveis x,y;

```
#include <stdio.h>
void funcaoTroca(int px, int py){
    int aux = px;
    px = py;
    py = aux;
}

int main()
{
    int x =10, y =20;
    printf("\n Valor X= %d e Y= %d\n",x,y);

    funcaoTroca(x,y); //passagem por valor

    printf("\n Valor X= %d e Y= %d\n",x,y);

    exit(0);
}
```

Valor X=10 e Y=20

Valor X=10 e Y=20

| Endereço | Valor | Variável |
|----------|-------|----------|
| e943ab   | 10    | x        |
| e943ac   |       |          |
| e943ad   | 20    | y        |
| e943ae   |       |          |
| e943af   | 10    | px       |
| e943b0   | 20    | py       |
| e943b1   |       |          |
| e943b2   |       |          |

Troca não realizada

# Passagem por Valor

- Troca de valores entre duas variáveis x,y;

```
#include <stdio.h>
void funcaoTroca(int px, int py){
    int aux = px;
    px = py;
    py = aux;
}

int main()
{
    int x =10, y =20;
    printf("\n Valor X= %d e Y= %d\n",x,y);

    funcaoTroca(x,y); //passagem por valor

    printf("\n Valor X= %d e Y= %d\n",x,y);

    exit(0);
}
```

Valor X=10 e Y=20

Valor X=10 e Y=20

| Endereço | Valor | Variável |
|----------|-------|----------|
| e943ab   | 10    | x        |
| e943ac   |       |          |
| e943ad   | 20    | y        |
| e943ae   |       |          |
| e943af   | 20    | px       |
| e943b0   | 10    | py       |
| e943b1   |       |          |
| e943b2   |       |          |

Troca não realizada em X e Y

# Ponteiros e Passagem por Referência

- Troca de valores entre duas variáveis x,y;

```
#include <stdio.h>
void funcaoTroca(int *px, int *py){
    int aux;
    aux = *px;
    *px = *py;
    *py = aux;
}

int main()
{
    int x =10, y =20;
    printf("\n Valor X=%d e Y=%d\n",x,y);

    funcaoTroca(&x,&y); //passagem por referência

    printf("\n Valor X=%d e Y=%d\n",x,y);

    exit(0);
}
```

Ponteiros \*px e \*py

Passagem por Referência

Valor X=10 e Y=20  
Valor X=20 e Y=10

Troca realizada

# Ponteiros e Passagem por Referência

- Troca de valores entre duas variáveis x,y;

```
#include <stdio.h>
void funcaoTroca(int *px, int *py){
    int aux;
    aux = *px;
    *px = *py;
    *py = aux;
}

int main()
{
    int x =10, y =20;
    printf("\n Valor X=%d e Y=%d\n",x,y);

    funcaoTroca(&x,&y); //passagem por referência

    printf("\n Valor X=%d e Y=%d\n",x,y);

    exit(0);
}
```

| Endereço | Valor  | Variável |
|----------|--------|----------|
| e943ab   | 10     | X        |
| e943ac   |        |          |
| e943ad   | 20     | y        |
| e943ae   |        |          |
| e943af   | e943ab | px       |
| e943b0   | e943ad | py       |
| e943b1   |        |          |
| e943b2   |        |          |

# Ponteiros e Passagem por Referência

- Troca de valores entre duas variáveis x,y;

```
#include <stdio.h>
void funcaoTroca(int *px, int *py){
    int aux;
    aux = *px;
    *px = *py;
    *py = aux;
}

int main()
{
    int x =10, y =20;
    printf("\n Valor X=%d e Y=%d\n",x,y);

    funcaoTroca(&x,&y); //passagem por referência

    printf("\n Valor X=%d e Y=%d\n",x,y);

    exit(0);
}
```

| Endereço | Valor  | Variável |
|----------|--------|----------|
| e943ab   | 20     | X        |
| e943ac   |        |          |
| e943ad   | 10     | y        |
| e943ae   |        |          |
| e943af   | e943ab | px       |
| e943b0   | e943ad | py       |
| e943b1   |        |          |
| e943b2   |        |          |

# Exercícios

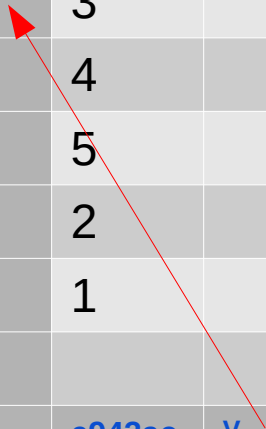
- Crie uma função para receber dois valores inteiros (A e B) e uma referência para inteiro (\*pc). O ponteiro \*pc retornará o resultado da soma de A e B
- Crie uma função para receber duas referências para inteiros (\*pa,\*pb), divida o valor \*pa pelo valor de \*pb, atribua o resultado a variável A;
-

# Vetores e Strings

- Vetores e Strings são ponteiros em C:
  - Um vetor é uma porção de memória contígua de mesmo tipo e o endereço do primeiro elemento é o mesmo do vetor; Os demais elementos estão equidistantes em função do seu tamanho (memória ocupada);
  - Exemplo

```
int v [5] = {3,4,5,2,1};
```

| Endereço | Valor  | Variável |
|----------|--------|----------|
| e943ab   |        |          |
| e943ac   | 3      |          |
| e943b0   | 4      |          |
| e943b4   | 5      |          |
| e943b8   | 2      |          |
| e943bc   | 1      |          |
| e943bf   |        |          |
| e943c0   | e943ac | v        |



Execução



# Vetores e Strings

```
void imprimeVetor(int vt[], int n){  
    for (int i=0;i<n;i++)  
        printf("\n v[%d] = %d ",i,vt[i]);  
    printf("\n");  
}
```

Função sem ponteiro

```
void imprimeVetorPonteiro(int *p, int n){  
    for (int i=0;i<n;i++)  
        printf("\n v[%d] = %d ",i,*(p+i));  
    printf("\n");  
}
```

Função Com ponteiro

*ponteiro +  
incremento*

```
> void somaDez(int *p, int n){ ...
```

```
int main()
```

```
{  
    int v [5] = {3,4,5,2,1};  
    int n = 5;  
  
    imprimeVetor (v,n); //passagem por referencia  
    imprimeVetorPonteiro (v,n); //passagem por referencia
```

v[0] = 3  
v[1] = 4  
v[2] = 5  
v[3] = 2  
v[4] = 1

v[0] = 3  
v[1] = 4  
v[2] = 5  
v[3] = 2  
v[4] = 1

Execução

| Endereço | Valor  | Variável |
|----------|--------|----------|
| e943ab   |        |          |
| e943ac   | 3      | v[0]     |
| e943b0   | 4      | v[1]     |
| e943b4   | 5      | v[2]     |
| e943b8   | 2      | v[3]     |
| e943bc   | 1      | v[4]     |
| e943bf   |        |          |
| e943c0   | e943ac | v        |
| e943c1   | e943ac | p        |

# Vetores e Strings

```
void imprimeVetor(int vt[], int n){...
```

```
void imprimeVetorPonteiro(int *p, int n){...
```

```
void somaDez(int *p, int n){  
    for (int i=0;i<n;i++){  
        *(p+i)+=10;  
    }  
}
```

Função acrescenta 10 a cada elemento V

```
int main()  
{  
    int v [5] = {3,4,5,2,1};  
    int n = 5;  
  
    imprimeVetor (v,n); //passagem por referencia  
    imprimeVetorPonteiro (v,n); //passagem por referencia  
    somaDez(v,n);  
    imprimeVetor (v,n); //passagem por referencia
```

v[0] = 3  
v[1] = 4  
v[2] = 5  
v[3] = 2  
v[4] = 1

v[0] = 3  
v[1] = 4  
v[2] = 5  
v[3] = 2  
v[4] = 1

v[0] = 13  
v[1] = 14  
v[2] = 15  
v[3] = 12  
v[4] = 11

Execução

# Vetores e Strings

```
#include <stdio.h>
void imprimeString(char st[], int n){
    for (int i=0;i<n;i++)
        printf("\n S[%d] = %c ",i,st[i]);
    printf("\n");
}
void imprimeStringPonteiro(char *p, int n){
    for (int i=0;i<n;i++)
        printf("\n S[%d] = %c ",i,*(p+i));
    printf("\n");
}
void codigoAsc(char *p, int n){
    for (int i=0;i<n;i++)
        printf("\n%c = %d",*(p+i),*(p+i));
}
int main()
{
    char v [] = "Estrutura de Dados";
    int n = sizeof(v);
    imprimeString (v,n); //passagem por referencia
    imprimeStringPonteiro (v,n); //passagem por referencia
    codigoAsc(v,n);
    exit(0);
}
```

Função com Vetor de Char;

Função ponteiro de Char;

# Vetores e Strings

```
#include <stdio.h>
void imprimeString(char st[], int n){
    for (int i=0;i<n;i++){
        printf("\n S[%d] = %c ",i,st[i]);
        printf("\n");
    }
}
void imprimeStringPonteiro(char *p, int n){
    for (int i=0;i<n;i++){
        printf("\n S[%d] = %c ",i,*(p+i));
        printf("\n");
    }
}
void codigoAsc(char *p, int n){
    for (int i=0;i<n;i++){
        printf("\n%c = %d",*(p+i),*(p+i));
    }
}
int main()
{
    char v [] = "Estrutura de Dados";
    int n = sizeof(v);
    imprimeString (v,n); //passagem por referencia
    imprimeStringPonteiro (v,n); //passagem por referencia
    codigoAsc(v,n);
    exit(0);
}
```

## Saídas do programa

S[0] = E  
S[1] = s  
S[2] = t  
S[3] = r  
S[4] = u  
S[5] = t  
S[6] = u  
S[7] = r  
S[8] = a  
S[9] =  
S[10] = d  
S[11] = e  
S[12] =  
S[13] = D  
S[14] = a  
S[15] = d  
S[16] = o  
S[17] = s  
S[18] =

S[0] = E  
S[1] = s  
S[2] = t  
S[3] = r  
S[4] = u  
S[5] = t  
S[6] = u  
S[7] = r  
S[8] = a  
S[9] =  
S[10] = d  
S[11] = e  
S[12] =  
S[13] = D  
S[14] = a  
S[15] = d  
S[16] = o  
S[17] = s  
S[18] =

E = 69  
s = 115  
t = 116  
r = 114  
u = 117  
t = 116  
u = 117  
r = 114  
a = 97  
= 32  
d = 100  
e = 101  
= 32  
D = 68  
a = 97  
d = 100  
o = 111  
s = 115

imprimeString()

imprimeStringPonteiro()

codigoAsc()

# Ponteiros em C

- Exercícios

- 1) Escreva uma função **menorVetor** que receba um vetor inteiro **v[0..n-1]**, a quantidade de elementos do vetor(n), o endereço de uma variável inteira **menor\_valor**(passagem por referência), e deposite nesta variável o valor do menor elemento do vetor.

```
void menorVetor(int *vt, int n, int *menor){
```

- 2) Escreva uma função **maiorVetor** que receba um vetor inteiro **v[0..n-1]**, a quantidade de elementos do vetor(n), o endereço de uma variável inteira **maior\_valor**(passagem por referência), e deposite nesta variável o valor do **maior** elemento do vetor.

```
void maiorVetor(int *vt, int n, int *maior){
```

- 3) Escreva uma função **menorMaiorVetor** que receba um vetor inteiro **v[0..n-1]**, a quantidade de elementos do vetor(n), e o endereço de duas variável inteira **menor\_valor** e **maior\_valor**(passagem por referência), e deposite nestas variáveis o valor do menor e maior elemento do vetor. Utilize as funções **menorVetor** e **maiorVetor**;

```
void menorMaiorVetor(int *vt, int n, int *menor, int *maior){
```

# Ponteiros em C

- Exercícios

4) Escreva uma função **menorVetor** que receba um vetor inteiro  **$v[0..n-1]$** , e a quantidade de elementos do vetor( **$n$** ), a função retorna o endereço do menor elemento do vetor.

```
int* menorVetor(int *vt, int n){
```

5) Escreva uma função **maiorVetor** que receba um vetor inteiro  **$v[0..n-1]$** , e a quantidade de elementos do vetor( **$n$** ), a função retorna o endereço do maior elemento do vetor.

```
int* maiorVetor(int *vt, int n){
```

6) Escreva uma função **menorMaiorVetor** que receba um vetor inteiro  **$v[0..n-1]$** , a quantidade de elementos do vetor( **$n$** ), e o endereço de duas variável inteira **menor\_valor** e **maior\_valor**(passagem por referência), e deposite nessas variáveis o valor do menor e

```
void menorMaiorVetor(int *vt, int n, int *menor, int *maior){
```

# Ponteiros em C

- Exercício 2
  - Um ponteiro pode ser usado para dizer a uma função onde ela deve depositar o resultado de seus cálculos. Escreva uma função **Converte** que converta minutos em horas-e-minutos.
    - A função recebe um inteiro **mnts** e os endereços de duas variáveis inteiras, digamos **h** e **m**, e atribui valores a essas variáveis, de modo que **m** seja menor que **60** e **que  $60 * h + m$  seja igual a **mnts****. Escreva também uma função main que use a função **Converte**.

# Ponteiros em C

- Exercício 3

- Escreva um função para receber uma String(vetor de char ), a quantidade elementos do vetor e um poteiro para armazenar todas as vogais presentes na String.
- Escreva um função para receber uma String(vetor de char ), a quantidade elementos do vetor e um poteiro para armazenar todas as consoantes presentes na String.
- Escreva um função para receber uma String(vetor de char ), e a quantidade elementos do vetor e retorne a quantidade de vogais na string.
- Escreva um função para receber uma String(vetor de char ), e a quantidade elementos do vetor e retorne a quantidade de consoantes na string.
- Crie um programa para utilizar as funções acima.