

# Estruturas de dados

## Vetores com ponteiros e passagem de parâmetro por referência



# Retomando...

- Ponteiros:
  - Armazenam endereços de memória nos quais existem variáveis;
  - Podem ser manipulados pelos operadores de endereço (&) e conteúdo (\*);

```
int nota;  
int *pNota;  
pNota = &nota;
```



# Ponteiros e vetores

- A forma de acesso a vetores vista anteriormente utilizava um índice para acessar cada elemento do vetor. Exemplo:  

```
vet[0] = 3;  
printf ("%d", vet[0]);
```
- **O nome de um vetor é, na realidade, um ponteiro para o primeiro elemento do vetor.**
- Pode-se usar o nome do vetor (ponteiro) como uma forma de acesso alternativa e mais ágil que a forma usual (via índices).



# Ponteiros e vetores

- Dado um vetor `vet[4]` de inteiros, para acessar o seu primeiro elemento podemos utilizar os seguintes comandos:

```
aux = vet[0];
```

```
aux = *vet;
```

Atribui o valor armazenado na primeira posição do vetor para a variável `aux`.

- Para acessar o segundo elemento:

```
aux = vet[1];
```

```
aux = *(vet + 1);
```

Atribui o valor armazenado na segunda posição do vetor para a variável `aux`.

&90	&94	&98	&102	&106
2	4	6	8	
&200	&204	&208	&212	&216

# Ponteiros e vetores – Exercício 1



Sistemas para Internet  
UFSM

- Crie um vetor de inteiros com 5 posições e adicione valores digitados pelo usuário. Utilizando o método visto anteriormente, mostre o conteúdo do vetor na tela.

# Ponteiros e vetores – Exercício 1



Sistemas para Internet  
UFSM

```
1 #include<stdio.h>
2 #include<conio.h>
3
4 int main(){
5     int v[5], i;
6     for (i = 0; i < 5; i++)
7         scanf("%d", (v + i));
8
9     for (i = 0; i < 5; i++)
10         printf("%d ", *(v + i));
11     return 0;
12 }
```

# Ponteiros e vetores

```
char str[15], *p1;  
p1 = str;
```

- O endereço de memória do primeiro elemento do vetor *str* foi associado a *p1*.
- Na linguagem C, o nome de um vetor sem um índice é o endereço para o início do vetor.
- É o mesmo que usar o seguinte comando:  

```
p1 = &str[0];
```



# Ponteiros e vetores

- Portanto, o acesso ao 5º elemento de um vetor pode ser feito das seguintes maneiras:

**str[4];**

ou

**\*(p1 + 4);**

- Frequentemente, o uso de aritmética é mais rápido do que o uso de indexação;



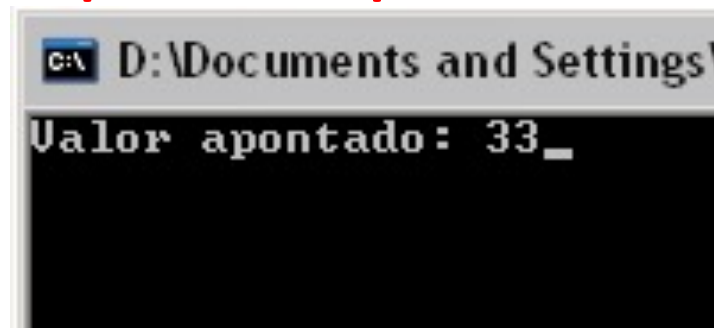


# Ponteiros e vetores

- Exemplo

```
int main ()
{
    int *p, vet[4] = {11, 22, 33, 44};
    p = &vet[0];
    printf ("Valor apontado: %d",    *(p + 2)    );
    return 0;
}
```

**O que será impresso na tela?**





## Ponteiros – Exercício 2

- Crie um vetor chamado vet com 5 posições;
- Escreva uma função para atribuir ao vetor os valores recebidos a partir do teclado;
- Crie funções (sempre usando ponteiros) para realizar as seguintes operações:
  - Localizar o maior valor no vetor;
  - Localizar o menor valor no vetor;
  - Mostrar os números pares do vetor.



## Ponteiros – Exercício 2

```
1 #include<stdio.h>
2
3 void lerVetor (int *p);
4 void mostraVetor (int *p);
5 void mostraMenor (int *p);
6 void mostraMaior (int *p);
7 void mostraPares (int *p);
8 int main(){
9     int vet[5], *p;
10    p = vet;
11    lerVetor (p);
12    mostraVetor(p);
13    mostraMenor(p);
14    mostraMaior(p);
15    mostraPares(p);
16    return 0;
17 }
18
19 void lerVetor (int *p){
20     int i;
21     printf("\n Informe os valores do vetor:\n");
22     for (i = 0; i < 5; i++)
23         scanf("%d", p + i);
24 }
```



## Ponteiros – Exercício 2

```
26 void mostraVetor (int *p){
27     int i;
28     printf("\n Vetor: ");
29     for (i = 0; i < 5; i++)
30         printf("%d ", *(p + i));
31 }
32
33 void mostraMenor (int *p){
34     int i, menor;
35     menor = *p;
36     for (i = 1; i < 5; i++){
37         if (*(p + i) < menor)
38             menor = *(p + i);}
39     printf("\n Menor valor: %d", menor);
40 }
```



## Ponteiros – Exercício 2

```
42 void mostraMaior (int *p){
43     int i, maior;
44     maior = *p;
45     for (i = 1; i < 5; i++){
46         if (*(p + i) > maior)
47             maior = *(p + i);}
48     printf("\n Maior valor: %d", maior);
49 }
50
51 void mostraPares (int *p){
52     int i;
53     printf("\n Valores pares: ");
54     for (i = 0; i < 5; i++)
55         if (*(p + i) % 2 == 0)
56             printf("%d ", *(p + i));
57 }
```



Sistemas para Internet  
UFSM

---

# Passagem de parâmetros por referência

# Passagem de parâmetros

- Por valor:
  - Na chamada da função uma **cópia** dos valores dos argumentos é enviada para os parâmetros da declaração da função.
  - Alterações feitas nestas cópias não geram nenhuma alteração no valor original das variáveis que foram utilizadas na chamada da função.

```
void teste (int a, int b)
{
    printf("%d \n",a);
    printf("%d \n",b);
}
```

```
int main(){
{
    int x = 5;
    int y = 2;
    teste(x, y);
    return 0;
}
```

a = x;  
b = y;





# Passagem de parâmetros

- Por referência:
  - Na chamada da função, deve-se passar os endereços de memória dos argumentos para os parâmetros declarados na função.

```
Teste (&x, &y) ;
```

- Na declaração da função, os parâmetros que recebem os argumentos devem ser **ponteiros**.

```
void Teste (int *a, int *b) {...}
```

- Alterações feitas dentro da função, afetam os valores das variáveis usadas na chamada (argumentos).





# Passagem de parâmetros

- Por referência:
  - Possibilita retornar quantos valores forem necessários, diretamente pelos parâmetros.

```
void teste (int *a, int *b) 3
{
    *a = 307;
    4 *b = 212;
}

int main() {
    int x = 5;
    1 int y = 2;
    2 teste(&x, &y);
    5 printf("%d \n", x);
    printf("%d \n", y);
    return 0;
}
```

1. Declaração de **x** e **y** e atribuição valores para eles.
2. Chamada da função **teste** passando os endereços de **x** e **y**.
3. Declaração da função **teste** com ponteiros **\*a** e **\*b** como parâmetros contendo os endereços de **x** e **y**.
4. Alteração do conteúdo das variáveis apontadas pelos ponteiros **a** e **b**.
5. Imprime os valores de **x** e **y**.



# Passagem de parâmetros

- Por referência:
  - Observação: Matrizes e *strings* já são consideradas passagem de parâmetros por referência.
  - Não é necessário utilizar o operador de endereço (&) na chamada da função, pois o endereço inicial (índice zero) será repassado para a função.

```
void teste (char x[10])  
{  
    ...  
}
```

```
int main(){  
{  
    char aula[10] = {"PARAMETRO"};  
    teste (aula);  
    return 0;  
}
```

```
void teste(char *pAula)  
{  
    ...  
}  
  
int main()  
{  
    char aula[10] = {"PARAMETRO"};  
    teste(aula);  
}
```

	&50	&51	&52	&53	&54	&55	&56	&57	&58	&59
aula =	P	A	R	A	M	E	T	R	O	\0



# Passagem de parâmetros

- Exemplo

```
void exReferencia (int a[5])
{
    int i;
    for (i=0; i<5; i++)
    {
        a[i] = i;
    }
}
```

```
int main(){
{
    int i, x[5] = {3,4,5,6,7};
    exReferencia(x);
    for (i=0; i<5; i++)
    {
        printf("%d \n", x[i]);
    }
    return 0;
}
```

```
void exReferencia(int *pX)
{
    int i;
    for(i=0; i<5; i++)
    {
        *(pX + i) = i;
    }
}
```

```
, int main()
{
    int i, x[5] = {100,200,300,400,500};
    exReferencia(x);
    for(i=0; i<5; i++)
    {
        printf("%d ", x[i]);
    }
    return 0;
}
```

# Exercício

- Desenvolver um algoritmo que leia dois valores inteiros. Estes valores devem ser passados por referência para uma função que verificará se são pares. Caso sejam pares, deve-se adicionar 5 a cada um deles. Caso contrário, adicionar 1. Os novos valores das variáveis devem ser apresentadas na função main.