UNIVERSIDADE FUMEC

Estrutura de Dados I

Ponteiros

Profª Amanda Danielle Lima de Oliveira Tameirão

Objetivos

- Conceito de ponteiros
- Declaração
- Apontamento
- Aplicação

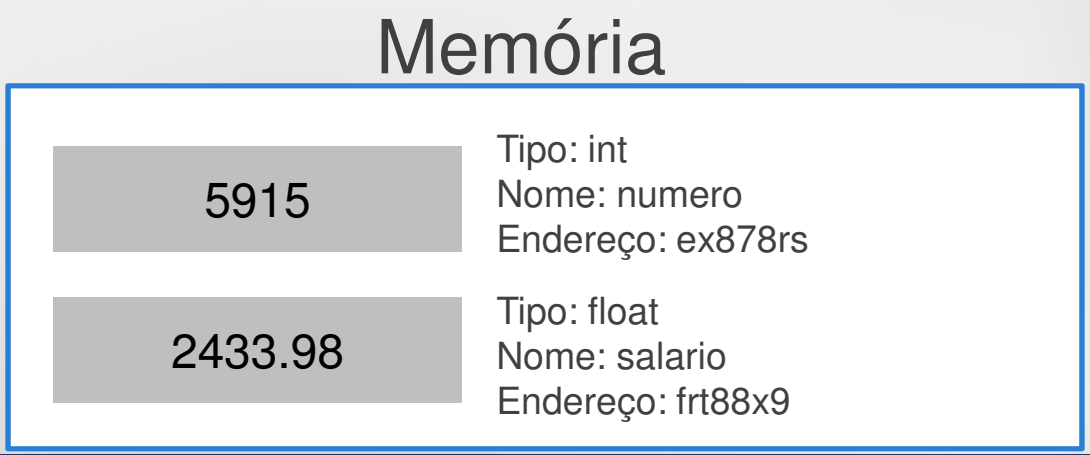
UNIVERSIDADE FUMEC

Conceito

Variável

Uma variável corresponde a um espaço de memória reservado para armazenar dados.

Todas são identificadas por nome e tipo, e também pelo endereço, ou seja, sua localização exata na memória.



Ponteiro
Conceito

Corresponde a uma variável que armazena o endereço de outra variável;

Assim é possível o acesso de modo indireto ao conteúdo da variável apontada.

UNIVERSIDADE FUMEC

Ponteiro
Vantagens de utilização

- Passagem de parâmetro por referência;
- Passagem de matrizes e vetores para outras funções;
- Manipular elementos de uma matriz ou vetor;
- Criar e utilizar estruturas complexas (árvores, listas, pilhas, etc);
- Alocar memória dinamicamente.

Ponteiro
Declaração

A forma geral de declaração é :

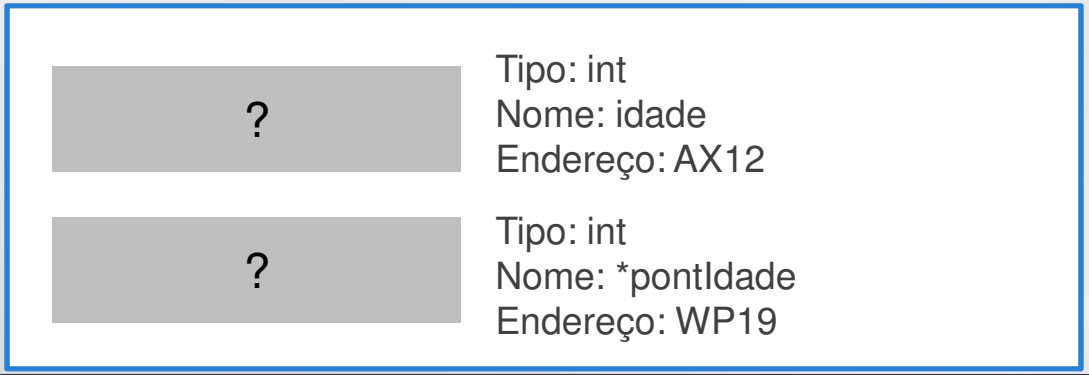
tipo_do_ponteiro *nome_do_ponteiro;

```
1  #include <stdio.h>
2  int main()
3  {
4      int idade; //Declaração de uma variável comum, denominada idade
5      int *pontIdade; //Declaração de uma variável ponteiro, denominada pontIdade
6
7      return 0;
8  }
```

Ponteiro
Declaração - Representação

```
1  #include <stdio.h>
2  int main()
3  {
4      int idade; //Declaração de uma variável comum, denominada idade
5      int *pontIdade; //Declaração de uma variável ponteiro, denominada pontIdade
6
7      return 0;
8  }
```

Memória



Ponteiro
Declaração - Tipo

<tipo> * <nome do ponteiro>;

O tipo, em ponteiros, indica que ele apontará somente para endereços de variáveis daquele tipo.

Por exemplo: Um ponteiro do tipo inteiro poderá armazenar endereços de memória de vairáveis inteiras.

Ponteiro
Declaração - Tipo

Você poderá declarar ponteiros utilizando qualquer tipo básico da linguagem, porém, ele só apontará para o tipo escolhido, nunca para mais de um tipo ao mesmo tempo.

Tipo
char
int
float
double

Ponteiro
Declaração - Operador

O Operador * permite que o compilador saiba que refere-se a uma variável ponteiro, ou seja, que armazenará endereços daquele tipo informado.

```
1  #include <stdio.h>
2  int main()
3  {
4      int idade; //Declaração de uma variável comum, denominada idade
5      int *pontIdade; //Declaração de uma variável ponteiro, denominada pontIdade
6
7      return 0;
8  }
```

Ponteiro
Declaração - Nome

O nome de uma variável do tipo ponteiro segue as mesmas regras de declaração de nomes de variáveis comuns.

Ponteiro
Operadores & e *

Para manipular ponteiros utilizamos os operadores unários & e *.

Ambos permitem a manipulação de ponteiros e a utilização dos mesmos.

Obs: Operadores unários efetuam suas tarefas sobre uma variável.

Ponteiro
Operador &

Ao ser aplicado na utilização de ponteiros **retornará** o **endereço de memória** desta.

Um exemplo comum é a função *scanf*(), em sua utilização aplicamos o & imediatamente antes da variável que armazenará a digitação, na verdade estamos informando ao compilador o endereço de memória que ele irá guardar aquele dado.

Ponteiro
Operador &

No exemplo abaixo é possível perceber que o comando scanf() passa o endereço de memória da variável utilizando o &.

```c
#include <stdio.h>
int main()
{
    int idade; //Declaração de uma variável comum, denominada idade
    int *pontIdade; //Declaração de uma variável ponteiro, denominada pontIdade

    printf("Idade: ");
    scanf("%d", &idade);

    return 0;
}
```

Ponteiro
Operador &

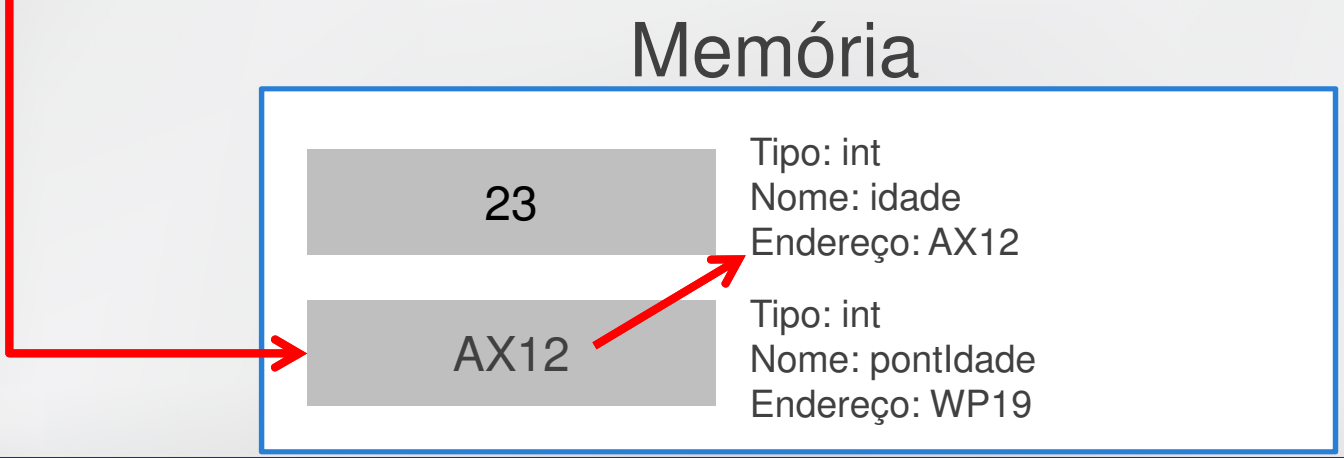
Para efetuarmos o apontamento, basta atribuirmos um endereço a um ponteiro.

```
1  #include <stdio.h>
2  int main()
3  {
4      int idade; //Declaração de uma variável comum, denominada idade
5      int *pontIdade; //Declaração de uma variável ponteiro, denominada pontIdade
6
7      pontIdade = &idade;
8
9      return 0;
10 }
```

No código, estamos atribuindo o endereço da variável idade ao ponteiro

Ponteiro
Operador & - Representação

```
1  #include <stdio.h>
2  int main()
3  {
4      int idade = 23; //Declaração de uma variável comum, denominada idade
5      int *pontIdade; //Declaração de uma variável ponteiro, denominada pontIdade
6
7      pontIdade = &idade;
8
9      return 0;
10 }
```



Ponteiro
Operador *

O operador * também é conhecido como referência de ponteiros ou operador indireto.

Serve para manipular ou recuperar o valor da variável apontada.

Ao aplicar * em um ponteiro estamos fazendo **referência** ao **conteúdo da variável para a qual ele aponta**.

Ponteiro
Operador *

Veja o exemplo a seguir:

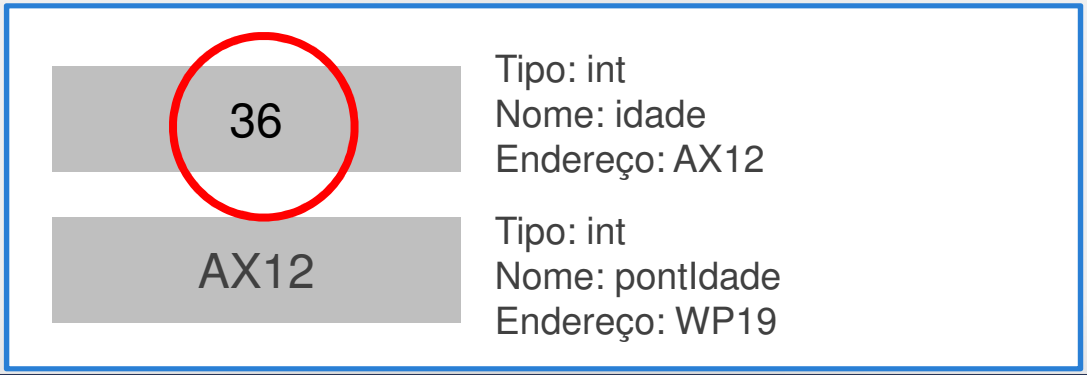
A partir do ponteiro foi possível alterar de 23 para 36 o valor da variável declarada.

```
1  #include <stdio.h>
2  int main()
3  {
4      int idade = 23; //Declaração de uma variável comum, denominada idade
5      int *pontIdade; //Declaração de uma variável ponteiro, denominada pontIdade
6
7      pontIdade = &idade;
8
9      *pontIdade = 36;
10
11     return 0;
12 }
```

Ponteiro
Operador & - Representação

```c
#include <stdio.h>
int main()
{
    int idade = 23; //Declaração de uma variável comum, denominada idade
    int *pontIdade; //Declaração de uma variável ponteiro, denominada pontIdade

    pontIdade = &idade;

    *pontIdade = 36;

    return 0;
}
```

Memória



UNIVERSIDADE FUMEC

Ponteiro
Manipulação de ponteiros

```c
#include <stdio.h>
int main()
{
    int idade = 23; //Declaração de uma variável comum, denominada idade
    int *pontIdade; //Declaração de uma variável ponteiro, denominada pontIdade

    pontIdade = &idade;
    *pontIdade = 36;

    printf("\nA variável idade possui o valor %d.", idade); //Acesso direto
    printf("\nA variável idade possui o valor %d.", *pontIdade); //Acesso indireto

    printf("\nEndereço de memória da variável idade %p.", &idade); //Acesso direto
    printf("\nEndereço de memória da variável idade %p.", pontIdade); //Acesso indireto

    printf("\nEndereço de memória da variável pontIdade %p.", &pontIdade);

    return 0;
}
```

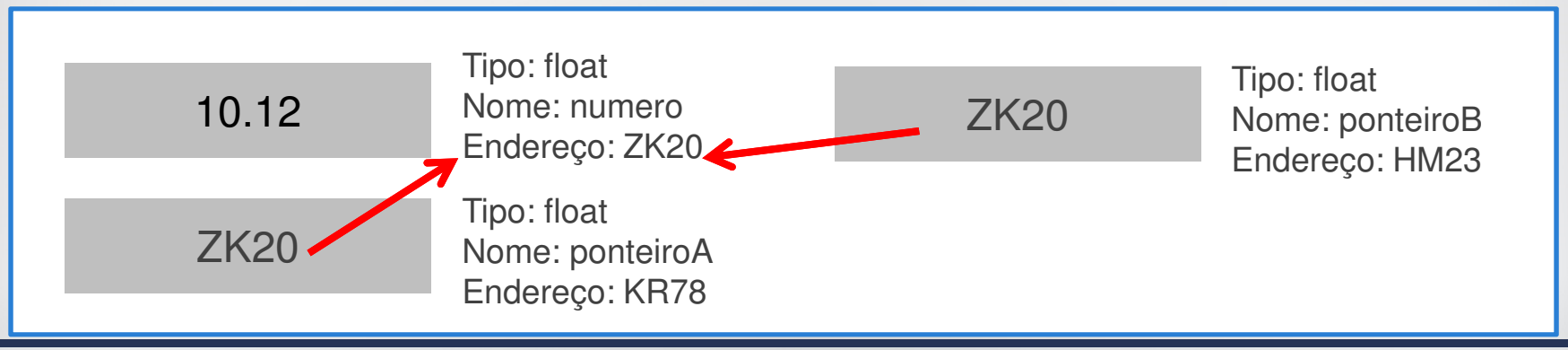
Ponteiro
Atribuição de ponteiros

Um ponteiro poderá receber o conteúdo de outro ponteiro, fazendo com que ambos apontem para a mesma variável.

```
1  #include <stdio.h>
2
3  int main()
4  {
5      float *ponteiroA, *ponteiroB; //Criação de dois ponteiros do tipo float
6      float numero = 10.12; //Criação de uma variável comum do tipo float
7      ponteiroA = &numero; /* A variável ponteiroA, do tipo ponteiro,
8                              aponta para o endereço da variável numero
9                           */
10     ponteiroB = ponteiroA; /* A variável ponteiroB, do tipo ponteiro,
11                               passou a apontar para o mesmo endereço que
12                               ponteiroA apontava (endereço da variável numero)
13                            */
14
15     return 0;
16 }
```

Ponteiro
Representação

```c
#include <stdio.h>

int main()
{
    float *ponteiroA, *ponteiroB; //Criação de dois ponteiros do tipo float
    float numero = 10.12; //Criação de uma variável comum do tipo float
    ponteiroA = &numero; /* A variável ponteiroA, do tipo ponteiro,
                            aponta para o endereço da variável numero
                         */
    ponteiroB = ponteiroA; /* A variável ponteiroB, do tipo ponteiro,
                              passou a apontar para o mesmo endereço que
                              ponteiroA apontava (endereço da variável numero)
                           */

    return 0;
}
```



Ponteiro
Ponteiro para ponteiro

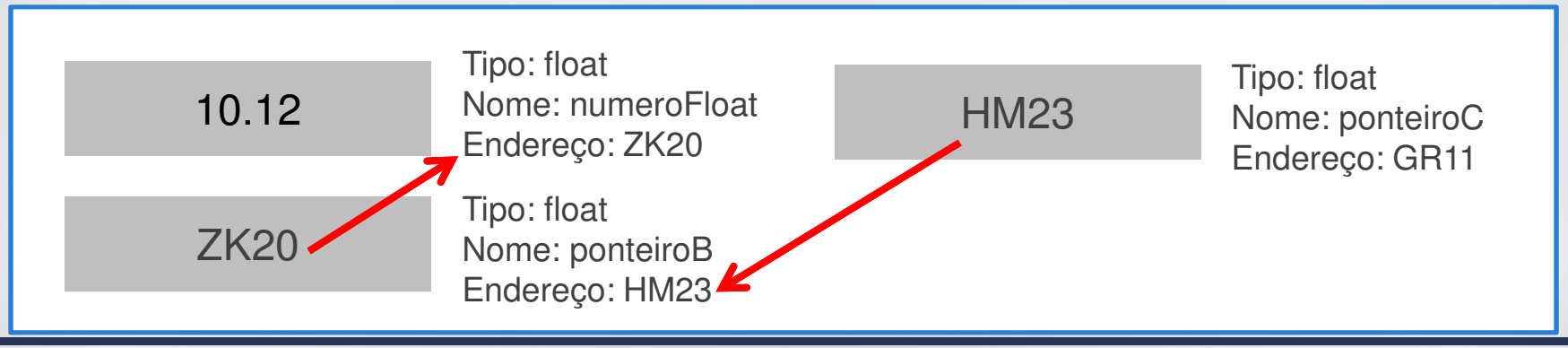
Um ponteiro também poderá apontar para outro ponteiro, permitindo que ele altere o conteúdo apontado pelo outro ponteiro.

```
1  #include <stdio.h>
2  int main(){
3      float numeroFloat = 10.12, //Variável comum
4            *ponteiroB = &numeroFloat, //ponteiro referenciando a variável comum
5            **ponteiroC = &ponteiroB; //ponteiro referenciando outro ponteiro
6
7      printf("\nVariável comum Conteúdo %f - Endereço %p", numeroFloat, &numeroFloat);
8      printf("\nPonteiroB Conteúdo %p - Endereço %p", ponteiroB, &ponteiroB);
9      printf("\nPonteiroC Conteúdo %p - Endereço %p", ponteiroC, &ponteiroC);
10     return 0;
11 }
```

Ponteiro
Representação

```c
#include <stdio.h>
int main(){
    float numeroFloat = 10.12, //Variável comum
          *ponteiroB = &numeroFloat, //ponteiro referenciando a variável comum
          **ponteiroC = &ponteiroB; //ponteiro referenciando outro ponteiro

    printf("\nVariável comum Conteúdo %f - Endereço %p", numeroFloat, &numeroFloat);
    printf("\nPonteiroB Conteúdo %p - Endereço %p", ponteiroB, &ponteiroB);
    printf("\nPonteiroC Conteúdo %p - Endereço %p", ponteiroC, &ponteiroC);
    return 0;
}
```

Memória



UNIVERSIDADE FUMEC

Vetores

Vector **A**

Conteúdo	12	9	10	16	25	13	20	14		14
Posição	0	1	2	3	4	5	6	7	8	9

UNIVERSIDADE FUMEC

Vetor

Variáveis homogêneas permitem o armazenamento de várias informações em uma mesma estrutura.

Na linguagem C, declarar vetor indica posicionar a variável em um lugar de memória onde os **endereços** são **contínuos**, ou seja, a primeira posição do vetor (vetor[0]) estará no endereço x, portanto, a segunda posição (vetor[1]) estará na posição x+1, ...

Vetor - Apontamento

O nome de um ponteiro armazena o primeiro endereço deste, por isso, podemos utilizar o nome ou a primeira posição, precedida de &.

```
1  #include <stdio.h>
2  int main()
3  {
4      double salario[5]; //Declaração de um vetor com 5 posições, denominado salario
5      double *pontSalario; //Declaração de uma variável ponteiro, denominada pontSalario
6
7      pontSalario = &salario[0];  //Essa linha faz o mesmo que a linha 11 deste código.
8
9      //OU
10
11     pontSalario = salario; //Essa linha faz o mesmo que a linha 7 deste código.
12
13     return 0;
14 }
```

UNIVERSIDADE FUMEC

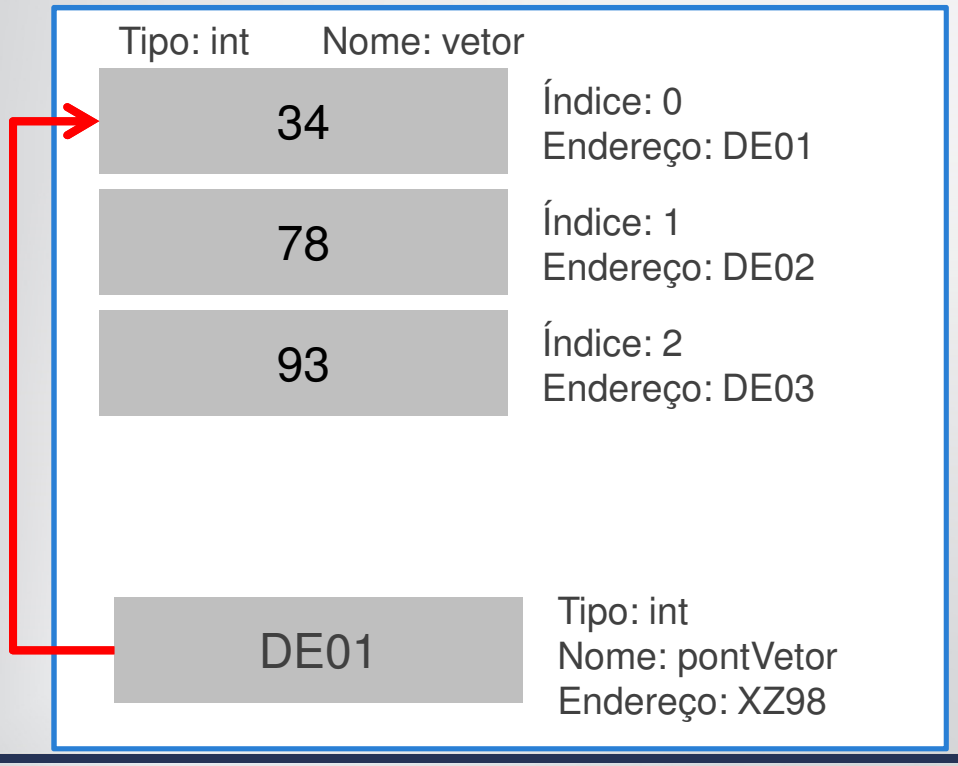
Manipulação

Ao manipular ponteiros em variáveis homogêneas lembre-se que você está apontando para o espaço de memória.

Portanto, ao incrementar ou decrementar um ponteiro é o mesmo que alterar o espaço de memória apontado.

Manipulação

Veja o exemplo a seguir, considere a alteração de ponteiros.



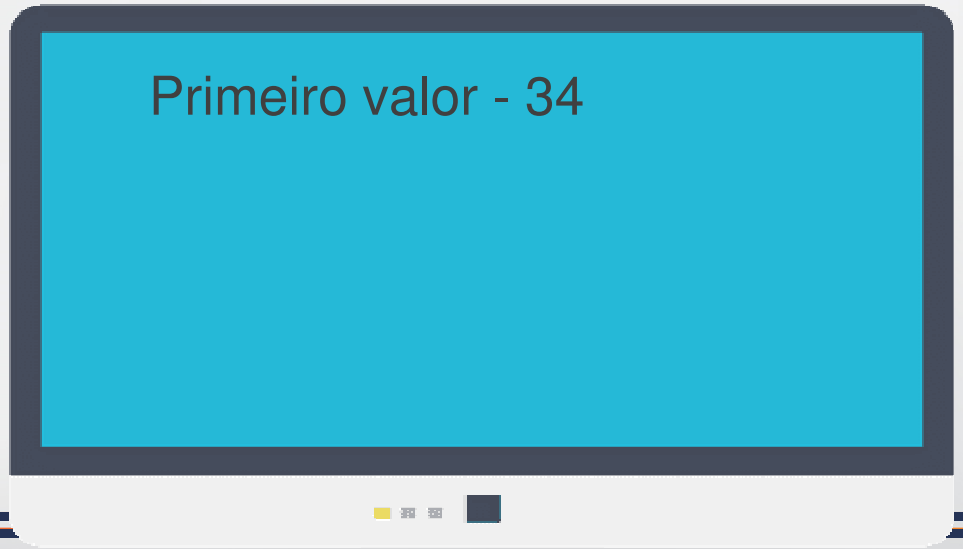

```c
#include <stdio.h>

int main()
{
    int vetor[] = {34, 78, 93};
    int *pontVetor;

    pontVetor = &vetor[0];

    printf("\nPrimeiro valor - %d", *pontVetor);

    pontVetor++;
    printf("\nSegundo valor - %d", *pontVetor);

    (*pontVetor)++;
    printf("\nTerceiro valor - %d", *pontVetor);

    *(pontVetor++);
    printf("\nQuarto valor - %d", *pontVetor);

    return 0;
}
```

Manipulação

vetor int
34
78
93

pontVetor = &vetor[0];

pontVetor int
Armazena o endereço de memória da **primeira** posição

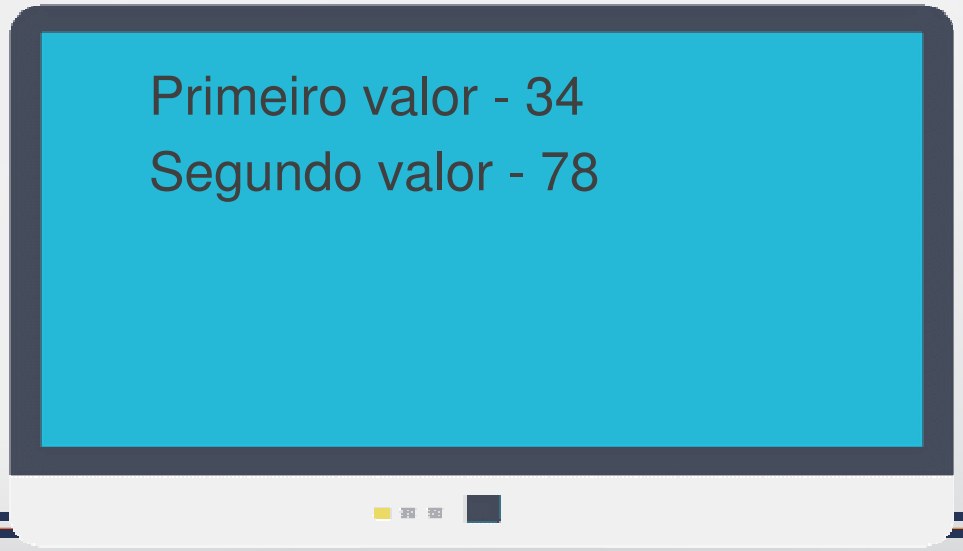


Manipulação

vetor int
34
78
93

pontVetor++;

pontVetor int
Armazena o endereço de memória da **segunda** posição



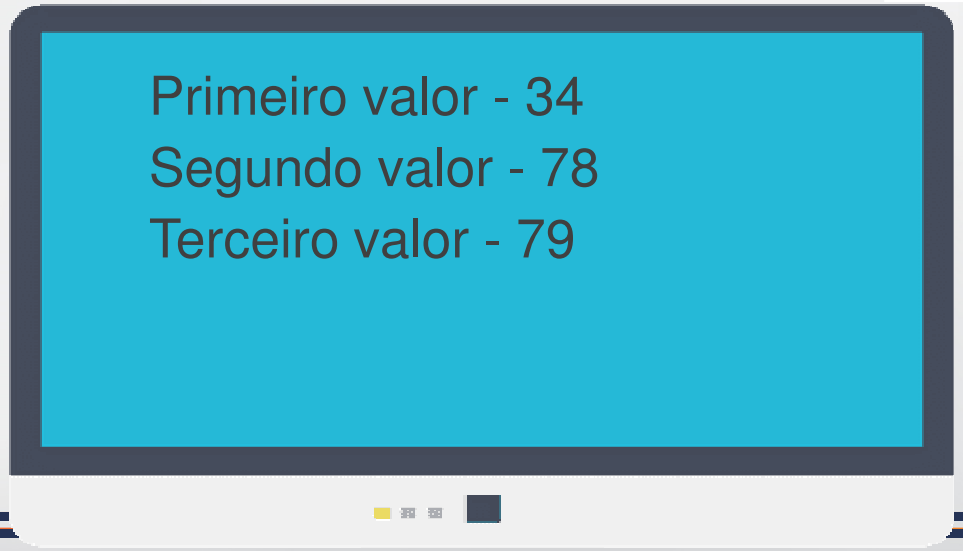
Manipulação

vetor int
34
79
93

Mantém o mesmo endereço de memória mas incrementa uma posição no vetor[1]

(*pontVetor)++;

pontVetor int
Armazena o endereço de memória da **segunda** posição

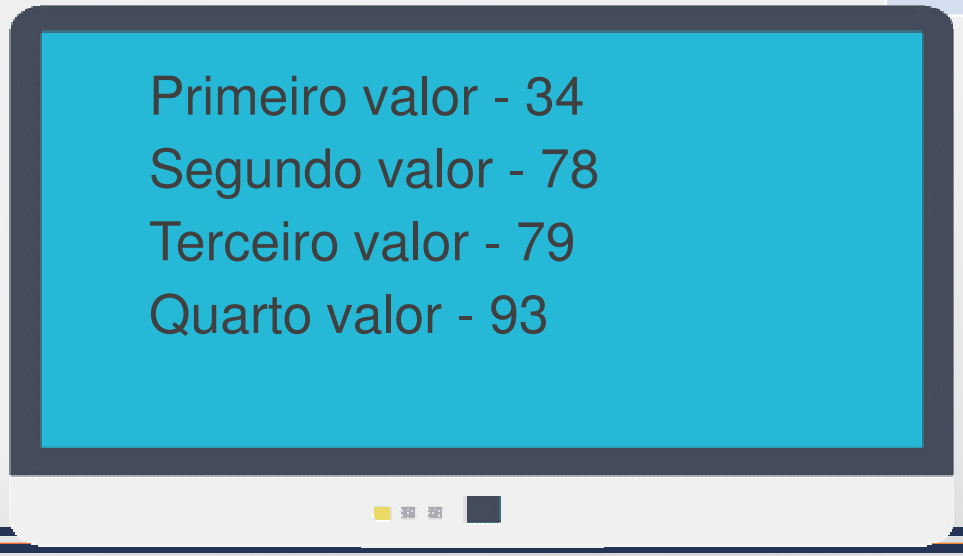


Manipulação

vetor int
34
79
93

*(pontVetor++);

pontVetor int
Armazena o endereço de memória da **terceira** posição



UNIVERSIDADE FUMEC

Exercícios Ponteiros

Bibliografia

- CORMEN, T. H. et al. **Algoritmos**: Teoria e Prática. 1ed. Rio de Janeiro: Campus, 2002. 916pp.
- ZIVIANI, N. **Projeto de algoritmos**: com implementações em Pascal e C. 2ed. São Paulo: Pioneira Thomson Learning, 2004. 552 p.
- TENENBAUM, A. M.; et al. **Estruturas de dados usando C.** 1ed. São Paulo: Pearson Education, 1995. 884 p.
- ASCENCIO, A. F. G; CAMPOS, E. A. V. **Fundamentos da Computação de Computadores:** Algoritmos, Pascal, C/C++ e Java**.** 2ed. São Paulo: Pearson Education, 2007. 434 p.
- Apostila criada para o curso de C da UFMG – Disponível em http://www.inf.ufsc.br/~fernando/ine5412/C_UFMG.pdf.
- Notas de aula do Prof. Flávio Lapper
- Notas de aula do Prof. Rafael Nunes
- Notas de aula do Prof. Ricardo Terra