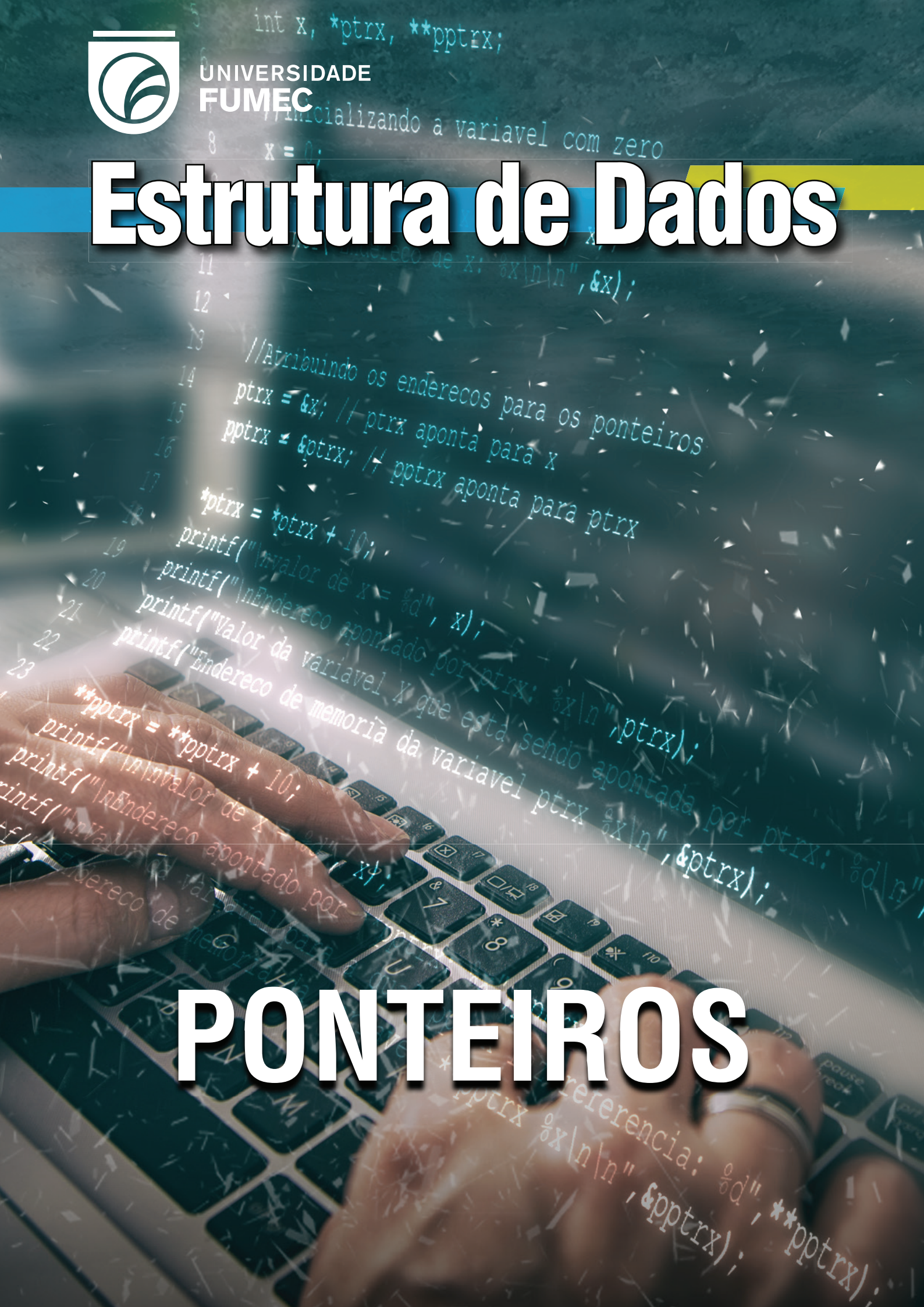
UNIVERSIDADE
FUMEC

Estrutura de Dados



PONTEIROS

APRESENTAÇÃO

Olá aluno(a), em programação você sabe o que é um ponteiro?

Pois neste módulo conversaremos sobre variáveis ponteiros, são variáveis que armazenam o endereço de memória de outras e facilitam nossa programação em determinadas situações.

Deve ter ficado curioso(a), né? Então vamos compreender melhor esse assunto tão importante para estrutura de dados.

OBJETIVOS DE APRENDIZAGEM

Ao final desse módulo você deverá ser capaz de:

- Definir o que é um ponteiro;
- Aplicar ponteiros com variáveis comuns e compostas;
- Efetuar aritmética de ponteiros.

FICHA TÉCNICA

UNIVERSIDADE FUMEC

FUMEC VIRTUAL - SETOR DE EDUCAÇÃO A DISTÂNCIA

Gestão Pedagógica
Coordenação
Gabrielle Nunes Paixão
Transposição Pedagógica
Talita Miranda Leite

TECNOLOGIA DA INFORMAÇÃO

Produção Multimídia
Rodrigo Tito M. Valadares
Paulo Roberto Rosa Júnior

Infra-Estrututura e Suporte
Coordenação
Anderson Peixoto da Silva

AUTORIA

Prof.ª Amanda Danielle

BELO HORIZONTE - 2020

PONTEIROS

Introdução

RECAPITULANDO

Você já conhece o conceito de uma variável, que é o espaço de memória, reservado para armazenar dados específicos.

E, apenas para relembrar, também conhecemos suas características, que são: o nome, que a identifica no programa; o tipo, que define o que será armazenado dentro daquele espaço; o conteúdo, que corresponde ao que foi armazenado, e está diretamente ligado ao tipo definido; o escopo, que determina onde aquela variável poderá ser lida e/ou alterada; e, por fim, o endereço de memória, que determina onde a variável foi armazenada na memória.

Dentre estas características, existe uma, que neste caso é muito importante, O ENDEREÇO DE MEMÓRIA!! Ele permite que saibamos onde o valor está armazenado. Veja a seguir:

MEMÓRIA

5915	Tipo: int Nome: numero Endereço de memória: ex878rs
2433.98	Tipo: float Nome: salario Endereço de memória: frt88x9

Todas as variáveis que declaramos possuem as características citadas, mas existe uma variável "especial", que, além de possuir estas características, também nos permite guardar o endereço de memória de outra variável. A ela damos o nome de **ponteiro**.

Um ponteiro é uma variável, que armazena o endereço de outra variável ou um endereço de memória que está guardando algum dado. Assim, é possível acessar o conteúdo de forma indireta.

OBSERVAÇÃO

Quando um ponteiro está armazenando algum endereço de memória, dizemos que aquele ponteiro está "apontando" ou "referenciando" para algum lugar.

Vantagens de utilizar um ponteiro

Existem diversas vantagens para a utilização de ponteiros, que são:

- Efetuar uma passagem de parâmetro por referência;
- Passar matrizes e vetores para outras funções (que corresponde a uma passagem de parâmetro por referência);
- Manipular elementos de uma matriz ou vetor;
- Criar e utilizar estruturas complexas como árvores, listas, pilhas, filas, tabela hash, etc;
- E, por fim, podemos alocar memória dinamicamente.

Para declararmos um ponteiro a sintaxe é:

```
tipo_do_ponteiro *nome_do_ponteiro;
```

Agora vamos compreender cada parte desta declaração:

- **Tipo do ponteiro** deve ser o mesmo tipo da variável, ou espaço de memória, para onde ele aponta. Por exemplo, se um ponteiro aponta para uma variável, ou espaço de memória, *int*, o tipo dele também deve ser *int*.
- **Nome do ponteiro** segue a mesma regra para declaração de uma variável comum. Sempre precedido de *, para indicar que aquela variável só poderá armazenar endereços de memória, e não conteúdos diretos.

```
1  int main(){
2      //Declaração de uma variável comum denominada idade
3      int idade;
4      //Declaração de um variável ponteiro, denominada pontIdade
5      int *pontIdade;
6      return 0;
7  }
```

No código o operador * permite que o compilador saiba que a variável se refere a um ponteiro. Para declará-lo, é importante saber que é obrigatório utilizar o mesmo tipo da variável que o ponteiro irá apontar.

Por exemplo, se você tem uma variável do tipo *float*, o ponteiro que apontará para ela também deverá ser do tipo *float*.

ATENÇÃO

Apenas declarar uma variável ponteiro não garante que ela esteja apontando para algum lugar, para isto é preciso efetuar o apontamento. No código anterior, as variáveis foram apenas declaradas, mas o ponteiro não aponta para nenhum lugar.

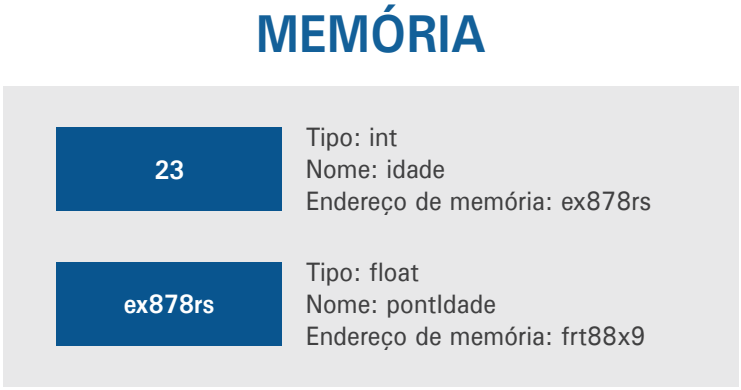
Operador &

Utilizamos o operador &, quando precisamos retornar o endereço de memória de uma variável, portanto, aplicado sobre uma variável, retorna o seu endereço de memória.

```
1  intmain(){
2      //Declaração de uma variável comum denominada idade
3      int idade = 23;
4      //Declaração de um variável ponteiro, denominada pontIdade
5      int *pontIdade;
6      /* Atribuição do endereço de memória da variável idade
7      para o ponteiro pontIdade */
8      pontIdade = &idade;
9      return 0;
10 }
```

A linha 8 do código possui uma atribuição de valores. Neste caso, o ponteiro *pontIdade* receberá o endereço de memória da variável *idade*.

Na memória a representação seria assim:



IMPORTANTE

Observe na Figura que o ponteiro, por ser uma variável, também possui um endereço de memória, ou seja, é armazenado em algum local.

Outro exemplo muito importante, que utiliza ponteiros, é o comando *scanf*, veja o código a seguir e sua forma de aplicação:

```
1  #include <stdio.h>
2  int main(){
3      int idade = 23;
4      printf("Digite a idade : ");
5      scanf("%d", &idade);
6      return 0;
7  }
```

O comando *scanf*, na linha 5, exige que o operador & fique imediatamente antes da variável que receberá o valor digitado. Neste caso, o que o operador faz é passar ao comando o endereço de memória da variável para que a digitação seja armazenada no lugar correto. O que ocorre, neste caso, é um apontamento. Apontamos diretamente para o endereço de memória em que haverá o armazenamento do que foi digitado.

Operador *

Este operador também é conhecido como referência de ponteiros ou operador indireto.

No código, pode ser utilizado de duas formas:

A primeira forma, já foi ensinada, serve para declarar um ponteiro, indica que aquela variável não será comum, mas um ponteiro.	A segunda forma serve para manipular a variável apontada, de forma indireta, ou seja, através do ponteiro que aponta para aquela variável (referenciar).

```
1  #include <stdio.h>
2  int main(){
3      int idade = 23;
4      int *pontIdade;
5      pontIdade = &idade; //Apontar o ponteiro para a variável
6      *pontIdade = 36; //Referenciar a variável apontada
7      /* O valor da variável idade foi alterado de forma indireta,
8      passou de 23 para 36, através do ponteiro. Isto e possível pois
9      utilizamos o operador * */
10     //Manipulando as variáveis declaradas
11
12     // Formas de acessar o conteúdo da variável idade
13     // Acesso direto
14     printf ("\nA variável idade possui o valor %d.", idade);
15     // Acesso indireto
16         printf ("\nA variável idade possui o valor %d.", *pontIdade);
17
18     // Formas de acessar o endereço de memória da variável idade
19     // Acesso direto
20     printf ("\nEndereço de memória da variável idade %p.", &idade);
21         // Acesso indireto
22     printf ("\nEndereço de memória da variável idade %p.", pontIdade);
23
24     // Formas de acessar o endereço de memória da variável pontIdade
25     printf ("\nEndereço de memória da variável pontIdade %p.", &pontIdade);
26
27 return 0;
28 }
```

Apenas pare reforçar, na linha 6, o valor da variável *idade* é alterado a partir do ponteiro, já que ele possui o endereço de memória da variável *idade*, pode ir direto naquele endereço e manipular como quiser.

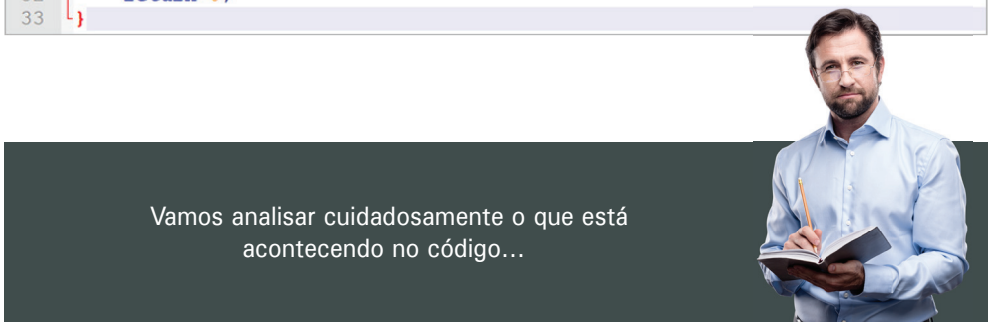
Para manipular os dados que precisa é necessário saber o que deseja acessar:

- Se deseja acessar o conteúdo do ponteiro, ou seja, o que esta dentro do espaço de memória dele, então **não** utilize *, apenas indique o nome do ponteiro e/ou a *string* de controle %p, afinal, ponteiros só armazenam endereços de memória
- Se deseja acessar o conteúdo da apontada, ou seja, o conteúdo do espaço de memória para onde o ponteiro aponta, então **utilize** *, e assim estará referenciando aquele espaço. Através do ponteiro estará acessando, e, se necessário, alterando, outro endereço de memória.

Ponteiro para ponteiro

Você também pode utilizar um ponteiro que apontará para outro ponteiro, neste caso, imagine que ambos terão acesso ao mesmo endereço de memória, porém, para manipular você precisa pensar bem em seus operadores.

```
1  #include<stdio.h>
2  intmain(){
3      int x = 23;// Declarar uma variável comum
4      int *px; // Declarar um ponteiro
5      int **ppx; // Declarar um ponteiro que apontar á para outro ponteiro
6      px = &x; // Referenciar uma variável comum para um ponteiro
7      ppx = &px; // Referenciar um ponteiro para outro ponteiro
8
9      // Imprimir o conteúdo de x
10     // Acesso direto a variável x
11     x = 12; //alterar diretamente o valor de x
12     printf("\n\nValor de x %d via x", x);
13
14     // Acesso indireto a variável x via px
15     *px = 24; //alterar indiretamente o valor de x
16     printf ("\nValor de x %d  via px", *px);
17
18     // Acesso indireto a variável x via ppx
19     **ppx = 66; //alterar indiretamente o valor de x
20     printf ("\nValor de x %d via ppx", **ppx);
21
22     // Imprimir o conteúdo de cada ponteiro
23     // Acesso direto a variável x
24     printf ("\n\nConteúdo de x e %d Endereço de memória de x %p", x , &x);
25
26     // Acesso indireto a variável x via px
27     printf ("\nConteúdo de px e %p Endereço de memória de px %p",
28             px , &px);
29     // Acesso indireto a variável x via ppx
30     printf ("\nConteúdo de ppx e %p Endereço de memória de ppx %p",
31             ppx , &ppx);
32     return 0;
33 }
```



Na linha 11 é possível alterar o valor da variável x apenas inserindo conteúdo nela, através do operador de atribuição. Se fosse necessário utilizar o comando *scanf* deveríamos colocar `scanf("%d", &x);` afinal, para pegarmos o endereço de memória de x é necessário utilizar o operador &.

Na linha 15 é possível alterar o valor da variável x através de px, neste caso, para referenciá-la é necessário utilizar o * sobre px, assim estaremos inserindo valor diretamente no espaço de memória de x. Se fosse necessário utilizar o comando *scanf* deveríamos colocar `scanf("%d", px);` afinal, o endereço de memória de x já está dentro de px.

Na linha 19 é possível alterar o valor da variável x através de ppx, neste caso, para referenciá-la é necessário utilizar dois * sobre ppx, afinal, ele terá que referenciar px e depois referenciar x. Imagine que está "subindo dois níveis" para chegar ao conteúdo final. Se fosse necessário utilizar o comando scanf deveríamos colocar `scanf("%d", *ppx);` afinal, o endereço de memória de x já está dentro de px.

Vetores x Ponteiros

Já sabemos como os vetores e matrizes são divididos na linguagem C, sempre utilizam indexadores para diferenciar os conteúdos, mas ao pensar em estruturas, temos que nos perguntar, "Como é dividido o endereço de memória de um vetor?"

Bem... em vetores, os endereços de memória são contíguos, ou seja, sequenciais. Portanto, se você possui um vetor com 5 posições, de qualquer tipo, a primeira posição estará no endereço X, então, a quinta posição estará no endereço X + 4, quatro endereços após o primeiro, totalizando cinco.

E se utilizássemos um ponteiro apontando para um vetor? Como seria?

Se necessitarmos atribuir o endereço de memória de um vetor, basta atribuirmos o endereço do primeiro elemento, e neste caso, o programa deve manipular o endereçamento pela quantidade de elementos de um vetor.

O endereço de memória do primeiro elemento de um vetor pode ser acessado pelas seguintes sintaxes:

• &nome_do_vetor[0] – Neste caso você está acessando o endereço da primeira posição, como se fosse uma variável comum.

• nome_do_vetor – Neste caso o nome de um vetor possui o endereço de memória da primeira posição dele.

Portanto, se o seu vetor tem o nome de *vetSalario*, para obter o endereço da primeira posição você pode utilizar:

&vetSalario[0] //Acessando a primeira posição

OU

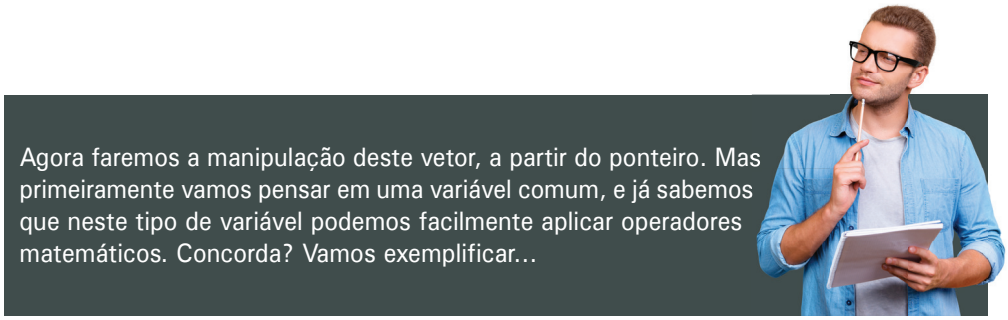
vetSalario //O nome de um vetor carrega o endereço da primeira posição

Veja o código a seguir:

```
1  #include<stdio.h>
2  int main(){
3      //Vetor com 10 posições double, denominado salário
4      double salario[10];
5
6      //ponteiro double , denominado pontSalario
7      double *pontSalario;
8
9      //Apontar o ponteiro para o vetor.
10     //A linha abaixo faz o mesmo que a linha 16
11     pontSalario = &salario[0];
12
13     //OU
14
15     //A linha abaixo faz o mesmo que a linha 11
16     pontSalario = salario;
17
18     return 0;
19 }
```

IMPORTANTE

Obviamente, se as linhas 11 e 16 efetuam a mesma execução, não é necessário escrever ambas no mesmo código, basta escolher uma sintaxe. O importante é efetuar o apontamento do ponteiro para o primeiro endereço de memória.



Imagine que temos uma variável com o nome de idade, e com o conteúdo 23. Se aplicarmos o comando idade++; incrementaremos 1 nesta variável, e, portanto, ela será acrescida de 1 item e passará de 23 para 24.

Bem, agora imagine que estamos manipulando uma variável do tipo ponteiro, que aponta para o primeiro endereço de um vetor. Como exemplo utilizaremos a variável *pontSalario*, do código, que aponta para o primeiro endereço do vetor salario, então, se aplicarmos o seguinte comando *pontSalario*++; e se seguirmos a mesma lógica de uma variável comum, incrementaríamos 1 nesta variável. Correto?

Sim, seu pensamento está correto, mas, sendo essa variável um ponteiro, incrementar 1 indica que ela apontará para o próximo endereço de memória disponível, ou seja, se ela pontava para X1 apontará para X2, e assim sucessivamente.

Aplicar comandos matemáticos para manipular vetor é conhecido como aritmética de ponteiros.

Veja um código com essa aplicação:

```
1  #include<stdio.h>
2  int main(){
3      int vetor[] = {34, 78, 93};//Declarar um vetor com 3 posições
4      int*pontVetor;//Declaração de um ponteiro
5
6      //Atribuir o primeiro endereço de memória ao ponteiro
7      pontVetor = vetor;
8
9      //Conteúdo e o endereço da primeira posição (através do ponteiro)
10     printf ("\n1° Valor - %d está na posição %p.", *pontVetor,
11     pontVetor);
12     pontVetor++;//Acrescenta 1 no ponteiro
13
14     //Conteúdo e o endereço da segunda posição (através do ponteiro)
15     printf ("\n2° valor - %d está na posição %p.", *pontVetor,
16     pontVetor);
17     (*pontVetor)++;//Acrescenta 1 na variável apontada pelo ponteiro
18
19     //Conteúdo e o endereço da segunda posição acrescido de 1
20     printf ("\n3° valor - %d está na posição %p.", *pontVetor,
21     pontVetor);
22     *( pontVetor++); //Acrescenta 1 no ponteiro
23
24     //Conteúdo e o endereço da terceira posição
25     printf ("\n4° valor - %d está na posição %p.", *pontVetor,
26     pontVetor);
27
28     return 0;
29 }
```

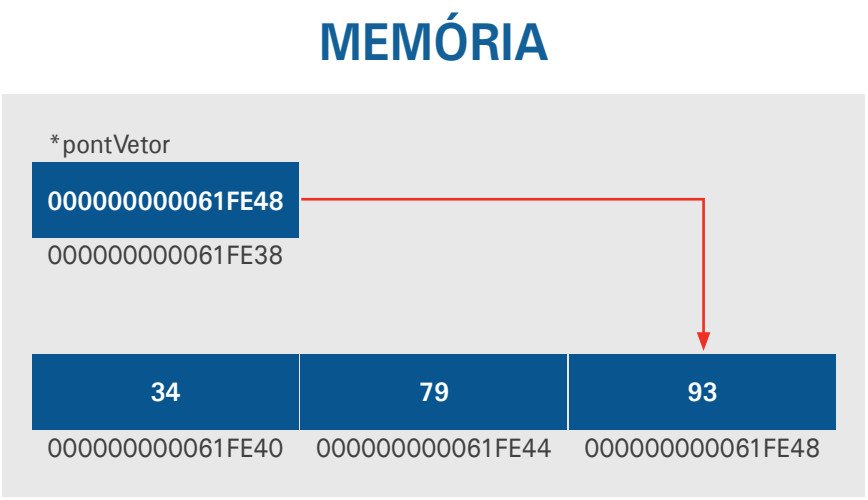
IMPORTANTE

Observações importantes no código:

Na linha de execução 3 existe um vetor não dimensionado declarado, como sua inicialização foi efetuada com 3 valores, o código "compreende" que o tamanho deste vetor é de 3 posições na memória, e, respectivamente, coloca os valores 34 – 78 – 93, em posições consecutivas.

Na linha 22, o operador * não tem efeito sobre o ponteiro, afinal, não existe retorno do valor da variável apontada.

Representação final na memória:



A Figura demonstra como é a finalização do código anterior, observe que, a segunda posição ficará com o valor 79, devido ao acréscimo efetuado na linha 17 do código.

E, apesar do ponteiro iniciar o código apontando para a primeira posição, ele termina apontando para a última, e neste caso apontar significa guardar o endereço de memória (a seta é apenas representativa).

Uma coisa importante para saber é que, nesta representação um inteiro (int) precisa de 4 *bytes* na memória, ou seja, cada variável inteira gasta 4 *bytes* em seu armazenamento, por isso os endereços de memória variam de 4 em 4. A cada 4 *bytes* começa outra variável, mas isto depende do tipo que você está utilizando. Cada tipo possui sua necessidade de armazenamento.

Síntese

Neste módulo conversamos sobre um tipo especial de variável conhecida como ponteiros.

Você descobriu que ponteiros são variáveis que armazenam endereço de memória. E por serem variáveis possuem as mesmas características desta (nome, tipo, conteúdo, escopo e endereço).

Todo ponteiro deve ser do mesmo tipo do espaço de memória que irá apontar, e dois operadores importantes para manipulá-los são: o *, que permite declaração e referenciamento; e o &, que permite acesso ao endereço de memória.

Também é possível utilizar ponteiros para armazenar endereço de memória de outro ponteiro, assim, teremos acesso ao mesmo endereço armazenado.

Em caso de aplicação de ponteiros em vetores, inicializa-se o primeiro endereço de memória para o ponteiro, a partir daí, dependendo da sua necessidade, é possível "caminhar" no vetor, apenas sabendo seu tamanho. Para este acesso utilize aritmética de ponteiros, que são operadores simples, matemáticas (soma e subtração), que efetuamos no ponteiro, fazendo com que ele "ande" um, dois ou mais endereços de memória para acessar o conteúdo necessário.

É importante saber que, utilizar ponteiros é dar "força" de alteração ao seu código. Tome cuidado com o apontamento incorreto e aproveite o ponteiro para acessar endereços de memória livremente.

Referências

CELES FILHO, W. **Introdução a estrutura de dados:** com técnicas de programação em c. Rio de Janeiro: Elsevier, 2004. 294 p.

DAMAS, Luis. **Linguagem C**. 10. ed. Rio de Janeiro: Ltc, 2016.

MANZANO, José Augusto N G. **Linguagem C:** Acompanhada de uma xícara de café. São Paulo: Érica, 2015.

MIZRAHI, Victorine Viviane. **Treinamento em linguagem C**. São Paulo: Pearson Prentice Hall, 2008. 407 p.