

ESTRUTURAS DE DADOS

PONTEIROS EM C



santo.oliani@fatectq.edu.br



<https://aula.olianijr.com.br>

O que é um ponteiro em programação?

- São variáveis, mas com uma diferença
- Enquanto variáveis armazenam um valor, ponteiros armazenam posição de memória

Mas .. Por que usar isto?

- Quanto trabalha-se com variáveis, em geral, tem-se o armazenamento de um único dado por posição.
- Já com os ponteiro é possível ter vários dados na memória do computador e navegar por estes dados somente apontando o endereço de memória de onde estão armazenados.

Declarando um ponteiro

Por mais que um ponteiro simplesmente aponte para uma posição de memória

Ele precisa saber qual é o tamanho da informação que ele irá apontar.

Declarando um ponteiro

Tabela de tipos em C

Tipo	Tamanho em Bytes	Faixa Mínima
char	1	-127 a 127
unsigned char	1	0 a 255
signed char	1	-127 a 127
int	4	-2.147.483.648 a 2.147.483.647
unsigned int	4	0 a 4.294.967.295
signed int	4	-2.147.483.648 a 2.147.483.647
short int	2	-32.768 a 32.767
unsigned short int	2	0 a 65.535
signed short int	2	-32.768 a 32.767
long int	4	-2.147.483.648 a 2.147.483.647
signed long int	4	-2.147.483.648 a 2.147.483.647
unsigned long int	4	0 a 4.294.967.295
float	4	Seis dígitos de precisão
double	8	Dez dígitos de precisão
long double	10	Dez dígitos de precisão

Declarando ponteiros

A declaração de um ponteiro deve seguir as mesmas características de uma variável convencional.

Mas... como o computador vai saber se estou ou não usando um ponteiro?

Declarando ponteiros

Usando o operador (*****) precedido da variável.

Assim um ponteiro pode ser declarado desta forma:

tipo *****variavel;

Exemplos de declaração de ponteiros

Ponto flutuante

```
float * x;
```

Caractere

```
char * y;
```

Inteiro

```
int * z;
```

Declaração de variáveis

Mas tome muito cuidado....

Ao contrário das variáveis.. Os ponteiros ao serem criados não dispõe de espaço de memória para armazenar a informação.

Mas e ai... Como vou salvar os meus dados?

Para o ponteiro é necessário solicitar ao sistema um espaço de memória para armazenar os dados.

Lembre-se.. Ponteiros armazenam endereços de memória onde o dado está e não o dado em si.

Solicitando ao sistema operacional um espaço de memória

Para isto é necessário utilizar um comando específico da linguagem C.

Este comando é o
malloc – memory allocation

Este comando está disponível na biblioteca **stdlib.h** (*STandard LIBrary*)

Sintaxe do malloc

O comando solicita ao sistema operacional um espaço de memória, porém, qual é o tamanho que será o meu dado?

Como vou organizar meus dados na posição de memória?

Descobrindo o tamanho de uma variável

Para descobrir o tamanho de uma variável é possível utilizar um comando:

`sizeof (tipo)`

Com isto é fácil descobrir qual é o tamanho em bytes que será necessário para armazenar aquele tipo de dado

Sintaxe do malloc

```
ponteiro = (tipo *) malloc (sizeof(tipo));
```

Recebe o endereço de memória ou NULL se não houver disponibilidade

Organiza e seta a posição de memória para receber o dado

Descobre o tamanho do dado
Solicita o espaço de memória ao sistema operacional

Depois do malloc tem-se a seguinte situação

00000
1H

ponteiro

000001H

Mas como armazenar um valor em um ponteiro?

Neste caso, pode-se utilizar o operador (*) para informar ao sistema que deseja armazenar um valor na posição de memória do ponteiro

O Assim tem-se: $*\text{ponteiro} = 10$

Neste exemplo, será salvo o número 10 na posição de memoria onde está o ponteiro.

Depois do malloc tem-se a seguinte situação



ponteiro

Mas o que acontece se escrever o ponteiro sem o operador (*)

Neste caso, você estará acessando o endereço da posição de memória onde seu dados estará salvo.

Isto significa que????

Quando acessar o ponteiro sem o operador (*), terá a seguinte resposta:

000001H



ponteiro

O valor de ponteiro é 000001H

Posso copiar o endereço de memória de
uma variável para outra?

Sim, perfeitamente.

Mas como?

Supondo a seguinte situação:

Foi criado uma variável do tipo inteiro
chamada de b e queira criar um
ponteiro p para está variável

Solução de apontamento para uma variável

```
1 //Biblioteca de Standard Input Output - Responsável pelas funções de entrada e saída
2 #include<stdio.h>
3 //Biblioteca de Standard Library - Contempla várias funções. Neste caso utilizada por causa de ponteiros
4 #include<stdlib.h>
5 //Definição do bloco principal
6 void main(){
7     //Declaração de variáveis locais
8     int b;
9     int *p;
10
11     //Atribuição de um valor para a variável b
12     b = 10;
13     p = &b;
14     //Escrevendo o endereço de memória do ponteiro.
15     printf("\nO Endereço da memoria da variavel p e %p", p);
16     printf("\nO valor armazenado na posicao de memoria apontada por p e %d", *p);
17     printf("\nO valor armazenado variavel b e %d", b);
18     system("pause");
19 }
```

Mas observando seu código... Não tem . Por que?

O *malloc* cria uma posição de memória para poder ser apontado por um ponteiro.

Neste exemplo, não foi necessário criar uma posição de memória já que a variável **b** já foi definida.

Lembre-se, a proposta era apontar para outra variável e não alocar um espaço para armazenar um novo valor.

O que aconteceu na linha 13?

Neste passo, foi associado o endereço de memória da variável p e assim, ambos possuem o mesmo endereço de memória.

- O Note que o operador (&) é utilizado para recuperar o endereço de memória da variável sem ponteiro.
- O Assim temos a seguinte resposta com o programa executado.

Tela de execução do sistema implementado

```
O Endereco da memoria da variavel p e 0012FF60
O valor armazenado na posicao de memoria apontada por p e 10
O valor armazenado variavel b e 10
O Endereco da memoria da variavel b e 0012FF60
Press any key to continue . . .
```

Avaliando a resposta

Como foi possível observar, tanto a variável inteira **b** como o ponteiro **p** dispõe do mesmo endereço de memória

Mas o que acontece se modificar o valor de b?

```
1 //Biblioteca de Standard Input Output - Responsável pelas funções de entrada e saída
2 #include<stdio.h>
3 //Biblioteca de Standard Library - Contempla várias funções. Neste caso utilizada por causa de ponteiros
4 #include<stdlib.h>
5 //Definição do bloco principal
6 void main(){
7     //Declaração de variáveis locais
8     int b;
9     int *p;
10
11     //Atribuição de um valor para a variável b
12     b = 10;
13     p = &b;
14     //Escrevendo o endereço de memória do ponteiro.
15     printf("\nO Endereco da memoria da variavel p e %p", p);
16     printf("\nO valor armazenado na posicao de memoria apontada por p e %d", *p);
17     printf("\nO valor armazenado variavel b e %d", b);
18     printf("\nEndereco da memoria da variavel b e %p", &b);
19     printf("\nDigite um novo valor para b: ");
20     scanf("%d", &b);
21     printf("\nO Endereco da memoria da variavel p e %p", p);
22     printf("\nO valor armazenado na posicao de memoria apontada por p e %d", *p);
23     printf("\nO valor armazenado variavel b e %d", b);
24     printf("\nO Endereco da memoria da variavel b e %p\n", &b);
25
26     system("pause");
27 }
```

Resposta da modificação implementada

```
0 Endereco da memoria da variavel p e 0012FF60
0 valor armazenado na posicao de memoria apontada por p e 10
0 valor armazenado variavel b e 10
0 Endereco da memoria da variavel b e 0012FF60
Digite um novo valor para b: 25

0 Endereco da memoria da variavel p e 0012FF60
0 valor armazenado na posicao de memoria apontada por p e 25
0 valor armazenado variavel b e 25
0 Endereco da memoria da variavel b e 0012FF60
Press any key to continue . . . -
```

Ok.. Mais ainda não sei quando tenho que usar o ainda?

O é necessário quando deseja armazenar um valor em uma posição de memória.

Com isto, se simplesmente quiser criar um ponteiro e armazenar um valor na posição de memória a qual o ponteiro aponta eu uso o malloc.

Exemplo

Neste exemplo, será apresentado uma implementação de um ponteiro para armazenamento de uma variável do tipo float

O O valor da variável será informada pelo usuário

Em uma próxima etapa, será apresentado o valor digitado pelo usuário e o endereço de memória o qual o ponteiro está apontando

Implementação do Exemplo

```
1 //importando as bibliotecas
2 #include<stdio.h>
3 #include<stdlib.h>
4
5 //Definindo o bloco principal
6 void main(){
7     //Declaração das variáveis locais|
8     float *ponteiro; //Variável que será utilizada para apontar para o endereço de memória onde a informação será salva
9
10    //Como o ponteiro não armazena o dado e sim o endereço é necessário alocar um espaço de memória
11    ponteiro = (float*) malloc(sizeof(float));
12
13    //Com o endereço de memória é possível solicitar o dado para ser salvo.
14    printf("\nDigite o valor: ");
15    scanf("%f", &ponteiro);
16
17    //Apresentando o valor lido e a posição de memória do dado do usuário respectivamente
18    printf("\nO valor digitado foi %f", *ponteiro);
19    printf("\nA posição de memoria é %p\n", ponteiro);
20
21    system("pause");
22 }
```

Mas sempre que solicitar ao S.O. ele vai me dar uma posição de memória?

O Infelizmente não.

Mas e dai como sei se deu certo?

O Neste caso é possível incluir após a alocação uma verificação com o valor **NULL**.

NULL???

Sim, todo tipo de dados tem um valor nulo baseado na sua proposta de ação.

O No caso dos ponteiros, NULL representa que não foi possível ou não tem valor definido a aquele ponteiro.

Assim tem-se

```
1 //importando as bibliotecas
2 #include<stdio.h>
3 #include<stdlib.h>
4
5 //Definindo o bloco principal
6 void main(){
7     //Declaração das variáveis locais
8     float *ponteiro; //Variável que será utilizada para apontar para o endereço de memória onde a informação será salva
9
10    //Como o ponteiro não armazena o dado e sim o endereço é necessário alocar um espaço de memória
11    ponteiro = (float*) malloc(sizeof(float));
12
13    //verifica se conseguiu ou não a posição de memória.
14    if(ponteiro == NULL){
15        printf("Sem posicao de memoria disponivel. O programa sera finalizado.");
16        exit(1); //finaliza o programa
17    }
18
19    //Com o endereço de memória é possível solicitar o dado para ser salvo.
20    printf("\nDigite o valor: ");
21    scanf("%f",&ponteiro);
22
23    //Apresentando o valor lido e a posição de memória do dado do usuário respectivamente
24    printf("\nO valor digitado foi %f", *ponteiro);
25    printf("\nA posição de memoria e %p\n", ponteiro);
26
27    system("pause");
28 }
```

O que mais é possível com ponteiros???

Na verdade é possível um infinidade de aplicações com este tipo de sistema, como por exemplo:

- O Criar um vetor unidimensional ou bidimensional em tempo de execução;
- O Criar estruturas encadeadas (Listas, pilhas, Filas, arvores binárias, etc.)
- O Passagens de parâmetros por referência..
- O E tudo mais que a criatividade permitir

Criando vetores unidimensionais com ponteiros

```
1 //importando as bibliotecas
2 #include<stdio.h>
3 #include<stdlib.h>
4
5 //Definindo o bloco principal
6 void main(){
7     //Declaração das variáveis locais
8     float *ponteiro; //Variável que será utilizada para apontar para o endereço de memória onde a informação será salva
9     int i,tam; //Esta variável armazena o número de elementos do vetor que o usuário deseja salvar
10
11    //Solicitando ao usuário o número de posições que o vetor
12    printf("\nDigite o número de posicoes que terá o vetor");
13    scanf("%d",&tam);
14
15    //Multiplicando as posições pelo sizeof o sistema cria um vetor unidimensional
16    ponteiro = (float*) malloc( tam * sizeof(float));
17
18    //A partir daqui trabalha-se como se fosse um vetor normal
19    for(i=0; i<tam; i++){
20        printf("\nDigite o valor do elemento[%d]: ", i);
21        scanf("%f",&ponteiro[i]);
22    }
23
24    //Escrita dos valores armazenados no vetor
25    for(i=0; i<tam; i++){
26        printf("\nelemento[%d] Memoria(%p): %.2f ",i,&ponteiro[i],ponteiro[i]);
27    }
28
29    system("pause");
30 }
```

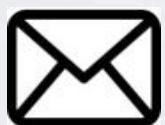
E como fica se for uma
estrutura???

```
1 //importando as bibliotecas
2 #include<stdio.h>
3 #include<stdlib.h>
4
5 //Definindo uma estrutura
6 typedef struct ponto{
7     int x,y;
8 };
9
10 //Definindo o bloco principal
11 void main(){
12     //Declaração das variáveis locais
13     ponto *pontEstrura; //Variável que será utilizada para apontar para o endereço de memória onde a informação será salva
14
15     //Alocando posição de memória para a estrutura
16     pontEstrura = (ponto*) malloc( sizeof(ponto));
17
18     //Com o endereço de memória é possível solicitar o dado para ser salvo.
19     printf("\nDigite o valor de x: ");
20     scanf("%d",&pontEstrura->x);
21
22     //Outra forma de acessar um ponteiro de uma estrutura
23     printf("\nDigite o valor de y: ");
24     scanf("%d",&(*pontEstrura).y);
25
26     //Apresentando o valor lido e a posição de memória do dado do usuário respectivamente
27     printf("\nO valor digitado de x foi %d", (*pontEstrura).x);
28     printf("\nO valor digitado de y foi %d", pontEstrura->y);
29     printf("\nA posicao de memoria e %p\n", pontEstrura);
30
31     system("pause");
32 }
```

Acessando partes da estruturas

- o Como pode ser observado, nas linhas 20 e 24, apresentam duas formas de acessar um ponteiro:
 1. `pontEstrura->x`
 2. `(*pontEstrura).y`
- o As duas formas são completamente iguais. Neste caso, usa-se a primeira forma pela sua simplicidade.

OBRIGADO



santo.oliani@fatectq.edu.br



<https://olianijr.com.br>