



Profº Luiz Paulo Zanetti

E-mail: luizpaulozanetti@hotmail.com



**Curso Superior de Tecnologia em
Análise e Desenvolvimento de Sistemas**

**Disciplina
Linguagem de Programação**

Ponteiro

Endereço de Memória

Ponteiros

Introdução Endereços de memória

Para o computador, **não existe praticamente diferença alguma entre as variáveis**, para ele é tudo bit, é tudo **1 ou 0**.

Um meio usado nos hardwares para administrar esse número gigantesco de 1's e 0's, é através do **endereçamento de memória**.

Cada **trecho da memória** tem um **endereço único**. Não existem dois bits, em uma máquina, que tenha o **mesmo endereço de memória**. O ato de **selecionar, ou alocar, um espaço de memória em C** é feito no momento da declaração.

Ponteiros

Introdução Endereços de memória

Um ponteiro é um tipo de dado que serve para indicar, ou armazenar, um endereço de memória.

Um ponteiro não é um inteiro, é um tipo de dado que armazena o endereço em que o inteiro está alocado.

Um ponteiro não é um float ou double, ponteiro é um tipo de dado que armazena o endereço em que o float ou double está.

Um ponteiro não é um char, ponteiro é um tipo de dado que pode armazenar o endereço em que um caractere está.

Ponteiros

Introdução Endereços de memória

Não confundir ponteiros com outros tipos de dados.

Isso se deve ao fato dos ponteiros serem um tipo de abstração, criado especialmente para facilitar o trabalho da computação em baixo nível, da computação que mexe diretamente com a memória de seu computador, poder este que pouquíssimas linguagens possuem.

Ponteiros

Obter o endereço de memória de uma variável

Sempre que **declaramos uma variável** e usamos ela, **estamos trabalhando com seu valor**.

Por exemplo:

numero1 = 1;

numero2 = 2;

letra1 = 'a';

letra2 = 'b';

Fixe bem esse detalhe: esse é o valor que está armazenado na memória, essas variáveis são um conjunto de bits, um conjunto de informações, um valor.

Ponteiros

Obter o endereço de memória de uma variável

Agora vamos descobrir em qual **posição da memória** esses **valores estão**. Para isso, basta **colocarmos** o símbolo de E comercial **antes da variável: &**

Exemplo

Para saber o endereço da variável 'numero1', fazemos: **&numero1**

Para saber o endereço da variável 'numero2', fazemos: **&numero2**

Para saber o endereço da variável 'letra1', fazemos: **&letra1**

Para saber o endereço da variável 'letra2', fazemos: **&letra2**

Ponteiros

Obter o endereço de memória de uma variável

Para facilitar a visualização do usuário, podemos imaginar a memória como um vetor gigantesco de espaços, e esses espaços são numerados com números inteiros.

Embora seja um inteiro, não quer dizer que o valor seja inteiro. Todos os endereços são números inteiros, mas nem todo o valor armazenado dentro daquele endereço de memória é inteiro.

Vamos fazer um exemplo para entender melhor a diferença entre valor e endereço de uma memória.

Ponteiros

Obter o endereço de memória de uma variável

Crie um programa em C que declara dois números inteiros e dois caracteres do tipo char (todos devidamente inicializados).

Em seguida, mostre o VALOR de cada variável, bem como seu ENDEREÇO.

Depois, altere os valores das variáveis e mostre novamente o VALOR e ENDEREÇO de cada variável desta.

Ponteiros

Obter o endereço de memória de uma variável

Após rodar esse exemplo, você verá a clara **diferença** entre o **VALOR** e o **ENDEREÇO** de uma variável na memória.

O valor é aquela informação que você inicia, e endereço é um número inteiro ENORME.

O valor é aquela **informação que é alterada**, já o **endereço** de uma variável permanece **CONSTANTE** !

Ponteiros

Obter o endereço de memória de uma variável

```
#include <conio.h>
#include <stdio.h>
void main()
{
int numero1=1,numero2=2;
char letra1='a',letra2='b';
clrscr();
printf("numero1: \n");
printf("Valor: %d\n", numero1);
printf("Endereco na memoria: %d\n\n", &numero1);
printf("numero2: \n");
printf("Valor: %d\n", numero2);
printf("Endereco na memoria: %d\n\n", &numero2);
printf("letra1: \n");
printf("Valor: %c\n", letra1);
printf("Endereco na memoria: %d\n\n", &letra1);
printf("letra2: \n");
printf("Valor: %c\n", letra2);
printf("Endereco na memoria: %d\n\n", &letra2);
printf("Alterando os valores...");
printf("pressione enter para continuar ...\n\n");
getch();
clrscr();
```

```
numero1=2112;
numero2=666;
letra1='A';
letra2='B';
printf("numero1: \n");
printf("Valor: %d\n", numero1);
printf("Endereco na memoria: %d\n\n", &numero1);
printf("numero2: \n");
printf("Valor: %d\n", numero2);
printf("Endereco na memoria: %d\n\n", &numero2);
printf("letra1: \n");
printf("Valor: %c\n", letra1);
printf("Endereco na memoria: %d\n\n", &letra1);
printf("letra2: \n");
printf("Valor: %c\n", letra2);
printf("Endereco na memoria: %d\n\n", &letra2);
getch();
}
```

Ponteiros

Tamanho da variável na memória: sizeof()

A **linguagem C** entende sua memória RAM como um **vetor** enorme de **bytes**.

Sempre que declaramos uma **variável** em C, **estamos guardando, selecionando ou alocando** um **espaço de bytes** desses, e dependendo do **tipo de variável**, o tamanho de memória é **reservada**.

Ponteiros

Tamanho da variável na memória: sizeof()

Embora as variáveis do **tipo *float* e *double*** sejam usadas para **representar números** em sua forma decimal, as variáveis do tipo ***double*** têm, como o próprio nome sugere, o **dobro de precisão**.

Ou seja, podemos colocar muito mais casas decimais em variáveis desse tipo. E para que isso aconteça, é óbvio que vamos precisar de **um espaço maior em memória**.

Ponteiros

Tamanho da variável na memória: `sizeof()`

Podemos descobrir quantos bytes certa variável ocupa através da função *sizeof()*.

Sempre que usamos a função *sizeof()*, ela retorna variáveis do tipo: *size_t*. Lembre-se bem desse tipo.

Vamos **usar** bastante em *strings* e de *alocação dinâmica de memória*.

Ponteiros

Tamanho da variável na memória: `sizeof()`

Faça um programa em C que mostra quantos bytes ocupam cada uma das variáveis: `char`, `int`, `float` e `double`.

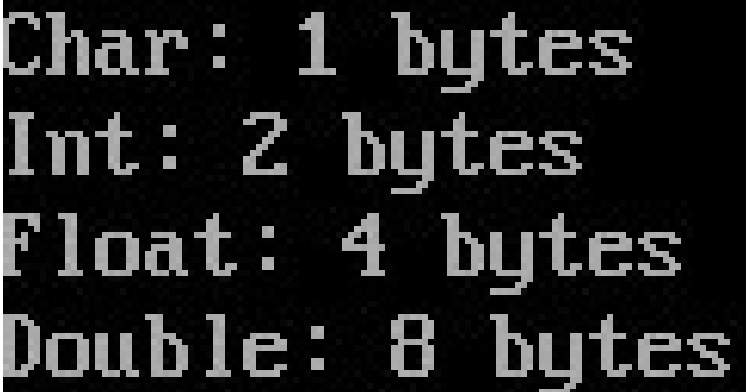
Existem **duas maneiras** de fazer isso, a primeira é simplesmente **colocando as palavras** reservadas **dentro do comando *sizeof()***.

A segunda maneira é **declarando variáveis** e colocando ela **dentro do comando *sizeof()***, como faremos no próximo exemplo.

Ponteiros

Tamanho da variável na memória: sizeof()

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    printf("Char: %d bytes\n", sizeof(char));
    printf("Int: %d bytes\n", sizeof(int));
    printf("Float: %d bytes\n", sizeof(float));
    printf("Double: %d bytes\n", sizeof(double));
    getch();
}
```



```
Char: 1 bytes
Int: 2 bytes
Float: 4 bytes
Double: 8 bytes
```

Ponteiros

Tamanho da variável na memória: sizeof()

Mostrar o endereço e número de bytes que cada variável ocupa.

Além de mostrar quantos bytes cada variável ocupa, mostre o endereço dela.

Para isso, declare 4 variáveis: uma char, um int, um float e um double.

Ponteiros

Tamanho da variável na memória: sizeof()

```
#include<stdio.h>
#include<conio.h>
void main()
{
char caractere;
int inteiro;
float Float;
double Double;
clrscr();
printf("Caractere: %d bytes \t em %d\n", sizeof(caractere), &caractere);
printf("  Inteiro: %d bytes \t em %d\n", sizeof(inteiro), &inteiro);
printf("  Float:   %d bytes \t em %d\n", sizeof(Float), &Float);
printf("  Double: %d bytes \t em %d\n", sizeof(Double), &Double);
getch();
}
```

```
Caractere: 1 bytes      em -11
Inteiro:   2 bytes      em -14
Float:     4 bytes      em -18
Double:    8 bytes      em -26
```

Ponteiros

Como declarar ponteiros em Linguagem C

Para declarar um ponteiro, ou apontador, em C basta colocarmos um **asterisco** - * - **antes do nome** desse ponteiro.

Sintaxe:

tipo ***nome_do_ponteiro**;

Por exemplo:

int ***ponteiro_pra_inteiro**;

float ***ponteiro_pra_float**;

char ***ponteiro_pra_char**;

Ponteiros

Como declarar ponteiros em Linguagem C

“Se os ponteiros armazenam endereço, e endereço são apenas números, por quê ter que declarar ponteiros com os tipos (int, float, char etc) ?”

A resposta é o tamanho que as variáveis ocupam em memória.

As variáveis ocupam posições vizinhas e contíguas (em seqüência) de memória (exceto, claro, o tipo char, que ocupa só 1 byte, ou seja, só um bloco).

Ponteiros

Como declarar ponteiros em Linguagem C

Vamos pegar o exemplo da **variável inteira**. Em minha máquina, ela **ocupa 4 bytes**.

Ou seja, **4 blocos de memória**, cada bloco com um **endereço**.

Mas o ponteiro **armazena apenas um endereço de memória**, e não 4.

Então, o **ponteiro** irá sempre armazenar o **endereço do primeiro bloco, do primeiro byte**.

Ponteiros

Como declarar ponteiros em Linguagem C

Se o C sabe quantos bytes cada variável ocupa, que elas são blocos vizinhos de memória e o ponteiro sabe o endereço do primeiro bloco, ele vai saber dos outros também!

É por isso que precisamos dizer o tipo de variável, antes de declarar o ponteiro.

Se for um ponteiro de inteiro, estamos dizendo: “Ponteiro, guarde esse endereço e os próximos 3, pois o inteiro tem 4 bloco”.

Se for um double: “Ponteiro, armazene o primeiro endereço, e saiba que os próximos 7 blocos são dessa mesma variável.”

Ponteiros e Vetores em Linguagem C

Quando **declaramos um vetor**, estamos **declarando um conjunto de variáveis** também contíguas, e cada uma dessas variáveis **ocupam vários bytes** (ou só 1 byte, se for char). Então, um **vetor é um conjunto maior ainda de bytes**, de blocos de memória.

Como você sabe, quando **apontamos um ponteiro para uma variável**, esse ponteiro **armazena o endereço do primeiro byte**, do menor endereço, da variável.

A relação com vetores é análoga: o nome do vetor é, na verdade, o endereço do primeiro elemento desse vetor.

Ponteiros e Vetores em Linguagem C

Se declararmos um **vetor** de nome *casa*, não importando o **número de elementos**, se **imprimirmos** o nome *casa* dentro de um `printf`, **veremos o endereço da primeira variável daquele vetor**.

Podemos ver um vetor como um ponteiro.

Isso explica o fato de que quando **passamos um vetor para uma função**, essa função **altera o valor** do vetor. Isso ocorre pois **não estamos passando uma cópia do vetor** (como acontece com as variáveis).

Ponteiros e Vetores em Linguagem C

Isso ocorre porque quando passamos o nome do vetor, **estamos passando um ponteiro para função.**

Ou seja, **estamos passando um endereço, onde a função vai atuar.**

E endereço de memória é o mesmo, dentro ou fora de uma função.

Crie um programa que mostre que o nome de um vetor.

Ponteiros e Vetores em Linguagem C

```
#include<stdio.h>
#include<conio.h>
void main()
{
int teste[10];
clrscr();
printf("Imprimindo o vetor 'teste': %d\n", teste);
printf("Imprimindo o endereço do primeiro elemento: %d\n", &teste[0]);
getch();
}
```

Imprimindo o vetor 'teste': -30

Imprimindo o endereço do primeiro elemento: -30

Ponteiros e Vetores em Linguagem C

Para declararmos um ponteiro *ptr* para um vetor *vet* []

Fazemos:

ptr = vet;

Pois o nome do vetor é um ponteiro (que não muda) para o primeiro elemento.

Fazemos:

ptr = &vet[0];

Como inicializar um ponteiro em Linguagem C

Por exemplo, se quisermos **armazenar o endereço do inteiro** '*numero*' no ponteiro '*numeroPtr*', fazemos:

```
int numero = 5;  
int *numeroPtr = &numero;
```

Agora **nosso ponteiro** está **apontando para a variável *numero***, pois o **ponteiro guardou o endereço do inteiro na sua posição de memória**.

Muito cuidado! Ponteiros armazenam endereços, e não valores. Ou seja, se fizer:

```
int *numeroPtr = numero;    ☹ ERRADO!!!!
```

Como inicializar um ponteiro em Linguagem C

É sempre bom inicializarmos os ponteiros, pois senão eles podem vir com lixo e você se esquecer, posteriormente, de inicializar.

Então, quando for usar, pensará que está usando o ponteiro de modo correto, mas estará usando o ponteiro com ele apontando para um lixo (endereço qualquer de memória).

Como inicializar um ponteiro em Linguagem C

Uma boa prática é apontar os ponteiros para a primeira posição de memória, que é conhecida como **NULL**.

Sempre que terminar de usar um ponteiro, coloque ele pra apontar para a posição **NULL**.

Para fazer isso, faça:

tipo *nome_do_ponteiro=NULL;

Como inicializar um ponteiro em Linguagem C

Declare um inteiro e uma variável do tipo double. Em seguida, crie dois ponteiros apontando para essas variáveis e mostre o endereço de memória das variáveis, mostre o endereço de memória que cada ponteiro armazenou. Por fim, coloque esses ponteiros para a primeira posição (**NULL**), de memória.

Para saber o endereço de uma variável dentro do printf, colocamos o %d e depois '&nome_variavel'. Para saber que endereço um ponteiro armazena no printf, também colocamos o %d entre as aspas, e fora colocamos apenas o nome do ponteiro.

Como inicializar um ponteiro em Linguagem C

```
#include <stdio.h>
#include <conio.h>
void main()
{
    clrscr();
    int inteiro;
    int *inteiro_ptr = &inteiro;

    double double1;
    double *double_ptr = &double1;

    printf("Endereco da variavel 'inteiro':\n", &inteiro);
    printf("Endereco armazenado no ponteiro\n'inteiro_ptr': %d\n\n", inteiro_ptr);
```

```
printf("Endereco da variavel 'double1':\n", &double1);
printf("Endereco armazenado no ponteiro\n'double_ptr': %d\n\n", double_ptr);
```

```
printf("Apos o uso dos ponteiros, vamos\naponta-los para NULL\n\n");
```

```
inteiro_ptr = NULL;
```

```
double_ptr = NULL;
```

```
printf("Endereco armazenado no ponteiro\n'inteiro_ptr': %d\n", inteiro_ptr);
printf("Endereco armazenado no ponteiro\n'double_ptr': %d\n", double_ptr);
getch();
}
```

```
Endereco da variavel 'inteiro': -12
Endereco armazenado no ponteiro 'inteiro_ptr': -12

Endereco da variavel 'double1': -20
Endereco armazenado no ponteiro 'double_ptr': -20

Apos o uso dos ponteiros, vamos aponta-los para NULL

Endereco armazenado no ponteiro 'inteiro_ptr': 0
Endereco armazenado no ponteiro 'double_ptr': 0
```

Obtendo o valor apontado pelo ponteiro: *

Para obtermos o valor da variável na qual o ponteiro aponta, devemos colocar um asterisco - * - antes do ponteiro, assim, o ponteiro irá mostrar o valor da variável (a variável que ele aponta), e não mais seu endereço.

Por exemplo, vamos supor que tenhamos declarado a variável inteira *‘numero’*:

```
int numero = 1;
```

Agora vamos criar um ponteiro *‘ptr_int’* e fazê-lo apontar para *‘numero’*:

```
int *ptr_int = &numero;
```

Obtendo o valor apontado pelo ponteiro: *

Agora *ptr_int* aponta para *numero*.

Para saber o valor de *numero* através do ponteiro, usamos: **ptr_int*

Veja bem:

ptr_int -> armazena o endereço da variável *numero*

**ptr_int* -> se refere ao valor da variável, ou seja, ao valor da variável *numero*.

Mostrando o valor das variáveis apontadas por ponteiros

Crie um programa em linguagem C que peça ao usuário três números inteiros e armazene em três variáveis inteiras através do uso de um ponteiro.

Após o usuário inserir cada número, mostre o número exibido, porém mostre através do ponteiro.