**Ponteiro**

Um ponteiro é uma variável que armazena o endereço de memória de um objeto. Os ponteiros são usados extensivamente em C e C++ para três propósitos principais:

* para alocar novos objetos no heap,
* para passar funções para outras funções
* para iterar sobre os elementos em matrizes ou outras estruturas de dados.

**ponteiros** são muito úteis quando temos uma situação em que uma variável precisa ser acessada em diferentes partes do programa.

Em um caso como esse o código pode ter vários ponteiros em diversas partes do programa apontando para uma variável específica.

E o melhor de tudo é que se o dado que estiver no local de memória apontado sofrer alguma alteração, não vai ter problema, pois os ponteiro espalhados no programa apontam para o endereço de memória e não exatamente para o valor.

### Como declaro um ponteiro?

### A sintaxe de um ponteiro é a seguinte:

tipo \* nome\_Ponteiro;

No exemplo acima temos o tipo que é o tipo de dado da variável que vamos apontar, podendo ser int, float ou até mesmo uma struct.

Depois temos o \* (asterisco) que nesse caso **determina** que a variável é um ponteiro. E por fim temos “Nome\_Ponteiro” que, como o próprio nome diz, é o nome do ponteiro.

Seguindo esses passos teremos a declaração de um ponteiro como o apresentado abaixo:

#include <stdio.h>

#include <stdlib.h>

int main(void) {

int \* ptr;

return EXIT\_SUCCESS;

}

Mas claro, isso não é o suficiente para que possamos usar um ponteiro, a única coisa que fizemos foi declarar um ponteiro e nada mais.

Agora precisamos atribuir a ele um endereço de memória de uma variável qualquer do tipo int. Para fazer isso é necessário que criemos essa variável. Por exemplo:

int valor = 10;

Depois disso teremos o endereço de memória que será atribuído a nosso ponteiro, mas é claro essa atribuição não é simples. Ela precisa ser diferenciada e isso é feito usando o & (E comercial), com esse caractere conseguimos atribuir o endereço de memória de uma variável a um ponteiro.

Veja a sintaxe:

ponteiro = &valor;

Então vamos misturar tudo isso em um código para ver no que vai dar.

#include <stdio.h>

#include <stdlib.h>

int main(void) {

int \* ptr;

int valor = 10;

ptr = &valor;

printf("Endereço = %x", &valor);

printf("Endereço = %x", ptr);

printf("Valor = %d", \*ptr);

return EXIT\_SUCCESS;

}

Endereço = 5015936c

Endereço = 5015936c

Valor = 10

Veja que no código acima temos a estrutura de um código em linguagem C e nele criamos uma variável do tipo int chamada valor a quem atribuímos o valor 10.

int valor = 10;

Depois declaramos nosso ponteiro ptr e atribuímos a ele o **endereço** da variável valor.

int \* ptr;

ptr = &valor;

Veja bem, ponteiros só aceitam endereços de memória. Não adianta tentarmos atribuir algum valor primitivo, por exemplo.

E para se obter o endereço de uma variável usamos o operador &. Foi o que fizemos.

Feito isso usamos printf() para exibir o valor do endereço da variável valor:

printf("Endereço = %x", &valor);

printf("Endereço = %x", ptr);

Se o ponteiro ptr está armazenando o **endereço** da variável valor, então quando imprimirmos o ponteiro ptr teremos o mesmo resultado (o mesmo endereço) que imprimir &valor (que retorna o endereço de memória da variável), não é verdade? Pois bem, foi isso que aconteceu. Viu o resultado da execução que mostramos ali em cima?

Endereço = 5015936c

Endereço = 5015936c

Valor = 10

Por ultimo, exibimos o valor que existe na variável valor, tal valor que se acessado pelo ponteiro usamos a sintaxe \*ptr.

printf("Valor = %d", \*ptr);

Perceba que para acessar o endereço de memória é necessário duas coisas muito importantes:

* **Primeiro:** dentro de printf() use %x para exibir o endereço de memória, pois o mesmo se trata de um valor hexadecimal.
* **Segundo:** para acessar o endereço de memória de uma variável use & antes dela.

É possível ainda acessar o endereço de memória de um ponteiro e isso nada tem a ver com o endereço de memória da variável, para isso, assim como fizemos com a variável valor, podemos fazer:

printf("Endereço de memória do ponteiro = %x", &ptr);

Então, recapitulando, dentro de um printf() se utilizarmos:

* ptr estaremos acessando o endereço de memória associado ao ponteiro. Ou seja, o endereço de memória de uma variável.
* &ptr aí já estaremos acessando o endereço de memória do **ponteiro**.

Para acessar o conteúdo daquele endereço associado ao ponteiro é necessário mudar um pouco a abordagem.

* **Primeiro:** dentro de printf() use %d para que seja possível mostrar um inteiro.
* **Segundo:** use o operador \* (que nesse caso nada tem a ver com multiplicação, tudo bem?) antes do ponteiro para acessar seu valor: \*ptr.

Altere o exemplo para:

printf("Endereço = %x", &valor);

printf("Endereço = %x", ptr);

printf("Valor = %d", \*ptr);

printf("Valor = %d", valor);

### Ponteiro para ponteiros

Então vamos lá! Para entender ponteiros para ponteiros precisamos de uma situação da vida real.

Imagine que você foi para uma balada e encontrou uma pessoa legal, vocês conversaram e essa pessoa escreveu em um papel velho seu número de telefone. Você pegou o papel, mas como ele está sujo e meio engordurado você resolve pegar um papel limpo e anotar o telefone novamente (e você está sem celular!).

Esse processo pode ser identificado em programação como um ponteiro para ponteiro. O também chamado ponteiro do ponteiro é aquele ponteiro que aponta para outro ponteiro.

int \*ptr;

int \*\*pptr;

Veja que acima declaramos um ponteiro comum com apenas um \*asterisco e depois declaramos o ponteiro do ponteiro, que nesse caso, utiliza-se dois \*\* asteriscos.

Essa é a declaração, já a atribuição é a seguinte:

ptr = &valor;

pptr = &ptr;

Bem simples, enquanto o ponteiro simples aponta para uma variável, o ponteiro do ponteiro aponta para o ponteiro simples.

Vamos misturar tudo isso e ver no que dá:

#include <stdio.h>

#include <stdlib.h>

int main(void) {

int \* ptr;

int \*\* pptr;

int valor = 10;

ptr = &valor;

pptr = &ptr;

printf("Endereço de ptr = %x", &ptr);

printf("Endereço de pptr = %x", &pptr);

printf("Valor ptr = %d", \*ptr);

printf("Valor pptr = %d", \*\*pptr);

return EXIT\_SUCCESS;

}