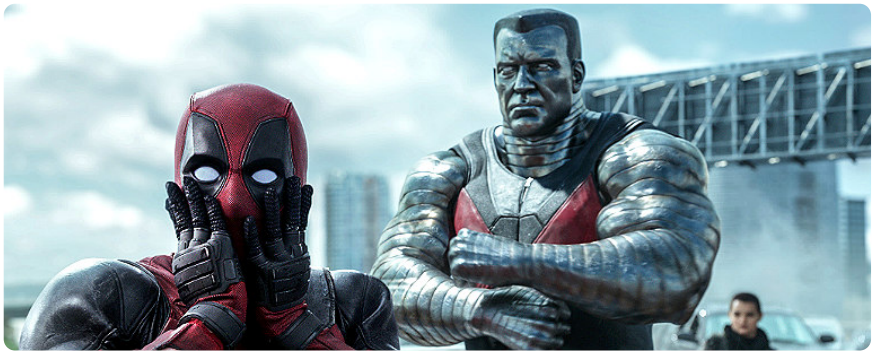
C (HTTPS://WWW.TREINAWEB.COM.BR/BLOG/CATEGORIA/DESENVOLVIMENTO-BACK-END/C/)

Ponteiros em C, uma abordagem básica e inicial

Redação Treinaweb (Https://Www.Treinaweb.Com.Br/Blog/Author/Redacao-Treinaweb/) 31 De Outubro De 2016

Nesse artigo vou falar um pouco sobre o terror dos estudantes de programação, a verdadeira bruxa que come criancinhas em noite de lua cheia ... Vou falar de **ponteiros**.



Mas calma, não se assuste! Vamos ver que esse assunto não é nenhum bicho de sete cabeças.

Pense em *ponteiros* como sendo aquele colega de trabalho "sacana" que não sabe nada a não ser apontar para você quando alguma pergunta é feita para ele. Quando seu chefe pergunta qualquer coisa o seu colega aponta para você responder, afinal é você quem têm na memória as informações.

Com ponteiros ocorre algo bem parecido, vamos ver uma explicação um pouco mais técnica sobre esse assunto.

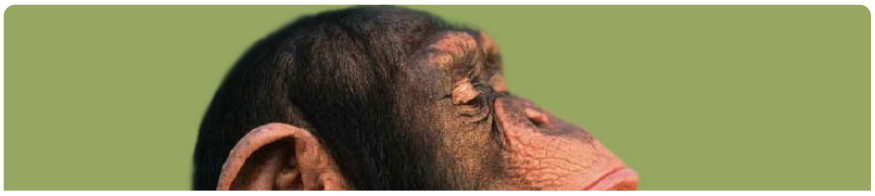
> **Podemos dizer que *ponteiros* ou apontadores (também podemos nos referir a eles desta forma), são variáveis que armazenam endereços de memória.**

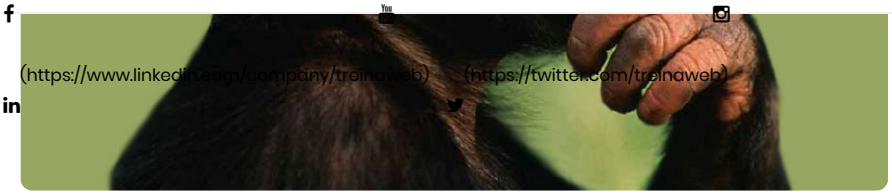
Mas claro, não é qualquer **endereço de memória**, nossos ponteiros armazenam endereços de outras variáveis.

Então, veja que aquilo que chamamos de **"apontar"** na realidade é simplesmente a denominação que damos a um ponteiro que contém o endereço de uma variável qualquer (e de qualquer tipo).

Agora você deve estar se perguntando:

> **Por que devo aprender isso, qual é o grande benefício?**





É simples: **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.

Deu pra perceber como o uso de ponteiros ajuda o programador? Dificilmente você vai escrever um código com menos do que algumas dezenas até centenas de páginas e poder usar vários ponteiros em uma aplicação desse tipo é mais que uma mão na roda.

Afinal como declaro um ponteiro?

Depois de ver como um ponteiro pode melhorar sua qualidade de vida, você deve estar se perguntando como declarar uma maravilha dessa em seus códigos.

É simples. 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;
```

Bem simples não é mesmo? Então vamos misturar tudo isso em um código para ver no que vai dar.

```
#include <stdio.h>
```

```
    int * ptr;
    int valor = 10;
    ptr = &valor;

    printf("Endereço = %x", &valor);
    printf("Endereço = %x", ptr);
    printf("Valor = %d", *ptr);

    return EXIT_SUCCESS;
}
```

Exemplo de um possível output para essa execução:

```
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

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);
```

Reiterando:

- `*ptr` – A variável `ptr` tem o endereço da variável `valor`, não é? É meio caminho andado para encontrar o valor dela, não acha? E para encontrar esse valor usamos o operador `*` antes do nome do ponteiro.
- `valor` – Estamos explicitamente imprimindo o conteúdo dessa variável `valor` do tipo `inteiro`.

É isso aí pessoal! simples, não é mesmo? Mas tudo que parece ser simples pode ser complicado.

Imagine que você queira se aventurar um pouco mais e sair da mesmice de ponteiros simples, se você esta nessa fase da vida, continue lendo ...

Ponteiro para ponteiros

Opa. Quem diria, você por aqui? Bom, se chegou até aqui é porque quer se aventurar, né? To sabendo.

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.

Doeu minha cabeça! Vamos ver a sintaxe para ficar mais fácil de entender?

```
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;
}
```

E no final exibimos os endereços e valores dos ponteiros. Veja que apesar de os ponteiros terem endereços diferentes, o valor apontado é o mesmo.

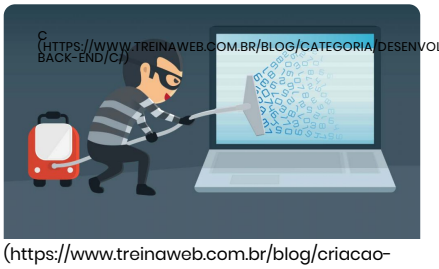
Muito legal, não é mesmo? Finalizo por aqui. Nos vemos nos próximos artigos.

C (HTTPS://WWW.TREINAWEB.COM.BR/BLOG/TAG/C/) # MEMÓRIA (HTTPS://WWW.TREINAWEB.COM.BR/BLOG/TAG/MEMORIA/) # PONTEIROS (HTTPS://WWW.TREINAWEB.COM.BR/BLOG/TAG/PONTEIROS/)

Artigo Anterior
Operações CRUD no ASP.NET MVC 5 com o ADO.NET (https://www.treinaweb.com.br/blog/operacoes-crud-no-asp-net-mvc-5-com-o-ado-net/)

Próximo Artigo
Os caminhos das certificações Microsoft (parte 1) (https://www.treinaweb.com.br/blog/os-caminhos-das-certificacoes-microsoft-parte-1/)

Artigos Relacionados



(https://www.treinaweb.com.br/blog/criacao-de-um-keylogger-em-c/)

Criação de um Keylogger em C (https://www.treinaweb.com.br/blog/criacao-de-um-keylogger-em-c/)

Redação Treinaweb
(Https://Www.Treinaweb.Com.Br/Blog/Author/Redacao-Treinaweb/)
9 De Janeiro De 2017

Curta nossa página

JUNTE-SE A MAIS DE 150.000 PROGRAMADORES

Receba o melhor conteúdo para DEVs do Brasil!

Digite seu melhor email

Cadastre-se

Quem somos

A TreinaWeb é a única escola Full Stack e DevOps do Brasil! Já são 12 anos formando desenvolvedores. Hoje, contamos com mais 350 cursos e cerca de 4.000 horas de conteúdo, formações e cursos completos.

A Treinaweb

Vantagens de estudar no Treinaweb
(https://www.treinaweb.com.br/vantagens-de-estudar-no-treinaweb?utm_source=blog&utm_medium=link&utm_campaign=tw-blog)
O que os nossos alunos dizem?
(https://www.treinaweb.com.br/depoimentos?utm_source=blog&utm_medium=link&utm_campaign=tw-blog)
Quanto Custa? (https://www.treinaweb.com.br/planos?utm_source=blog&utm_medium=link&utm_campaign=tw-blog)
Nossos Cursos (https://www.treinaweb.com.br/cursos-online?utm_source=blog&utm_medium=link&utm_campaign=tw-blog)
Formações (https://www.treinaweb.com.br/formacoes?utm_source=blog&utm_medium=link&utm_campaign=tw-blog)
Cursos para empresas
(https://www.treinaweb.com.br/contato/empresa?utm_source=blog&utm_medium=link&utm_campaign=tw-blog)
Ferramentas para desenvolvedores
(https://www.treinaweb.com.br/ferramentas-para-desenvolvedores?utm_source=blog&utm_medium=link&utm_campaign=tw-blog)
Sugira um novo artigo
(https://www.treinaweb.com.br/contato?