PONTEIROS

É um assunto muito discutido no estudo da linguagem C, mas é algo que vocês já conhecem, já fazem uso, só faltam ser apresentados. Essa notícia já nos tranquiliza, né? Já começamos o assunto bem!

Então, *vamos revisar alguns elementos que aprendemos no início da disciplina*?

Quando declaramos uma variável, de qualquer tipo, por exemplo:

```
int x;
```

Ao executarmos o programa (.exe) do nosso código, o Sistema Operacional consultará a memória RAM, que é uma memória não permanente, diferente do HD e SSD. Então, após consultar se há memória disponível, o Sistema Operacional reserva esse espaço de memória para essa variável. E após alocar este espaço de memória ele guarda o endereço que reservou, para quando manipulamos a variável conseguimos atualizar as informações armazenadas.

Um detalhe nesse processo, é que o Sistema Operacional associa o nome da variável x, com o endereço de memória alocada. Para nós, programadores, é muito mais fácil conhecer a variável pelo nome que escolhemos para ela, por exemplo "x" do que seu endereço, por exemplo 33ff44.

Então, quando fazemos uso da instrução:

```
scanf("%d", &x);
```

Estamos expressando algo como*: "capture do teclado um inteiro e armazene-o no endereço de x"*. Isso é diferente para uma atribuição, quando utilizamos, por exemplo:

```
x = 10;
```

Para o Sistema Operacional, ele já conhece o endereço de x, porque declaramos lá em cima, então interpreta algo como *"no endereço da variável x o valor armazenado será atualizado para 10"*.

Observe, o que conversamos até agora sobre declaração, uso e atualização de uma variável, está associado ao **endereço**.

Bom, agora sim, podemos desmistificar os ponteiros:

É um tipo de variável que armazena endereços de outras variáveis.

Declarando um Ponteiro

E como declaramos uma variável ponteiro? Bom, como falamos, a variável *ponteiro guarda endereço de memória de outras variáveis*, e o espaço ocupado por uma variável do tipo inteiro na memória é diferente do espaço ocupado por uma variável do tipo char, um inteiro ocupa 4 bits e um char ocupa 1 bit. Você pode pensar: "xiii vou ter que decorar isso?" Não! Decorar não. Vocês devem só saber que como estamos tratando de endereço, *e para a manipulação dos ponteiros é importante saber onde começar e até onde vai cada espaço de memória reservado para uma variável*. E que, um ponteiro para uma variável inteira é diferente para uma variável char.

Então, quando declaramos uma variável ponteiro, temos que indicar o tipo de endereço que vamos armazenar. Olha só:

```
int *p;
char *t;
float *w;
```

No exemplo acima declaramos três ponteiros para tipos diferentes de dados. E porque o "*"? Há! Utiliza-los no ato da declaração, e indica para o compilador C que estamos declarando um ponteiro e não uma variável simples. Então, fazer uso do "*" no ato da declaração é uma regra sintática. Uma regra de escrita. Então, sintaticamente, a regra para declaração de um ponteiro é:

```
<tipo> *nome_do_ponteiro;
```

E se tivermos mais de um ponteiro do mesmo tipo, podemos declarar na mesma linha? Sim. Olha como fica a regra:

```
<tipo> *nome_do_ponteiro1, *nome_do_ponteiro2, ... , *nome_do_ponteiroN ;
```

Um exemplo:

```
int *p, *q
```

Em que p e q, são variáveis ponteiro para inteiro.

Tudo tranquilo até aqui? Espero que sim!

Vamos seguindo. Vamos atribuir ao ponteiro o endereço de uma variável? Bora!

Atribuindo um Endereço ao Ponteiro

Imagine declarar uma variável inteira e um ponteiro para inteiro:

```
int *p;
int x;
```

Show! Sabemos que neste ponto do programa o compilador já reservou um espaço de memória para guardar a variável x e a variável p. Então, pergunto: você lembra como obtemos o endereço de x? Você pode dizer que nunca realizou essa operação, certo? Mas, lembra que começamos esse material dizendo que conhecemos ponteiros? Olha só:

```
scanf("%d", &x);
```

Lembra o que isso significa? *"capture do teclado um inteiro e armazene-o no endereço de x"*

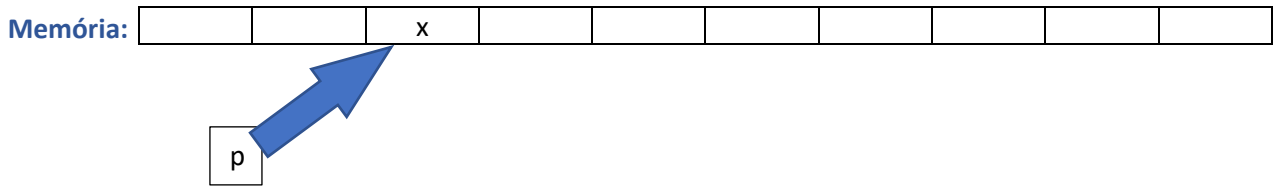
SHOW! ENDEREÇO! O operador em C que obtemos o valor do endereço de uma variável é **&**. Olha aí, já conhecíamos!

Então, se queremos atribuir ao ponteiro p o endereço de x, basta fazermos:

```
p = &x;
```

Pronto, a partir de agora através do ponteiro p conseguimos manipular a variável x, poque sabemos seu endereço, porque sabemos sua localização na memória. E se sabemos seu endereço podemos alterar e consultar seu conteúdo e podemos até apagar essa variável da memória.

Ilustrando esse processo poderíamos ter:



Conseguimos um ultra, mega, hiper, acesso a uma variável sem precisar saber seu nome.

Alterar o valor de uma variável através de um ponteiro

Então, já que temos acesso ao endereço de x, vamos conhecer como podemos realizar todas essas operações?

Alterar o valor da variável x através do ponteiro p:

```
int *p;
int x;
p= &x;
*p = 10;
```

Opa! Então o "*" é utilizado para *declarar uma variável ponteiro e para alterar o conteúdo da variável apontada* pelo ponteiro?

Sim, isso mesmo!

A operação **p = 10*, significa dizer "altere o conteúdo da variável apontada por p pelo valor 10".

Imprimir o conteúdo de uma variável através de um ponteiro

Com o "*" podemos também acessar o conteúdo da variável "apontada" para imprimir ou para realizar operações, como:

Imprimir o conteúdo de x através do ponteiro p:

```
int *p;
int x;
p= &x;
*p = 10;
printf(" valor de x = %d", *p);
```

Realizar operações aritméticas com uso de ponteiros

Fazer uso do conteúdo de x através do ponteiro p em operações aritméticas:

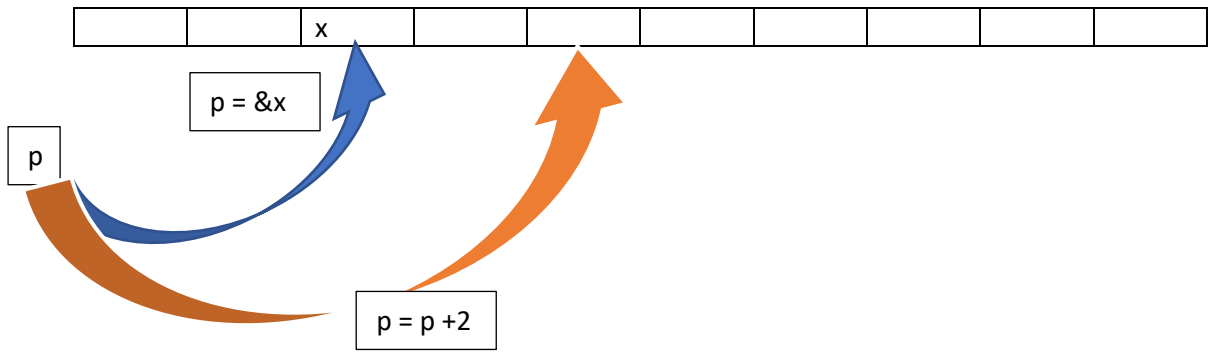
```
int *p;
int x, y;
p= &x;
*p = 10;
y = *p+3;
```

Acessar outros espaços de memórias

Com uma variável ponteiro podemos acessar outros espaços de memória

```
int *p;
int x;
p= &x;
p = p + 2;
```

Como o p estava apontando x, ao realizarmos p + 2, o ponteiro “anda” duas “casas” de memória e passa a apontar um novo endereço:



A necessidade do uso de ponteiros

E por que essa necessidade surgiria? Bom, vamos avançar no estudo da linguagem C, e em alguns momentos vamos ter alguns blocos de comandos que não estarão dentro do nosso bloco principal (main), e esse será o tema do nosso próximo conteúdo: *funções!* Nelas poderemos fazer uso de ponteiros. Assim como quando tivermos conjuntos de variáveis, os *vetores*, com eles também estaremos trabalhando com ponteiros.

Então, vamos aproveitar o momento para praticar e entender mais sobre ponteiros!