Ponteiros

1. Como Funcionam

Variáveis do tipo **int** guardam valores inteiros, as do tipo **float** guardam números de ponto flutuante, já as do tipo **char** guardam caracteres. **Ponteiros** guardam endereços de memória e ponteiro também tem tipo.

2. Declarando e Utilizando Ponteiros

Para declarar um ponteiro temos a seguinte forma geral:

*tipo_do_ponteiro *nome_da_variável;*

Exemplos de declarações:

```
int *pt;
char *temp,*pt2;
```

2. Declarando e Utilizando Ponteiros (continuação)

Um cuidado, muito importante, que deve ser tomado na manipulação de ponteiros, é o de inicializar um ponteiro antes de utilizá-lo, pois quando esses são declarados, apontam para um lugar indefinido.

Para atribuir um valor **válido** a um ponteiro recém-criado poderíamos igualá-lo a um endereço de memória de uma variável declarada.

Mas, como saber a posição na memória de uma variável do nosso programa?

2. Declarando e Utilizando Ponteiros (continuação)

Para saber o endereço de uma variável basta usar o operador **&**. Veja o exemplo:

```
...
int count=10;
int *pt;
pt=&count;
...
```

OBS.: Não confundir com o and bit a bit

2. Declarando e Utilizando Ponteiros (continuação)

Como nós colocamos um endereço válido em **pt**, ele agora pode ser utilizado. Podemos, por exemplo, alterar o valor de **count** usando **pt**. Para tanto vamos usar o operador "inverso" do operador **&**, que é o operador *****. No exemplo anterior, uma vez que fizemos **pt=&count** a expressão ***pt** é equivalente ao próprio **count**. Isto significa que, se quisermos mudar o valor de count para 12, basta fazer ***pt=12**.

Ponteiros

Exemplo 1:

```
#include <stdio.h>
main ()
{
    int num,valor;
    int *p;
    num=55;
    p=&num;
    valor=*p;
    printf ("\n\n%d\n",valor);
    printf ("Endereco para onde o ponteiro aponta:
    %p\n",p);
    printf ("Valor da variavel apontada: %d\n",*p);
}
```

Ponteiros

Exemplo 2:

```
#include <stdio.h>
main ()
{
    int num,*p;
    num=55;
    p=&num;
    printf ("\nValor inicial: %d\n",num);
    *p=100;
    printf ("\nValor final: %d\n",num);
}
```

2. Operações Aritméticas com Ponteiros

a) Atribuição → Se temos dois ponteiros, **p1** e **p2,** e quisermos que **p1** aponte para o mesmo lugar que **p2**, basta fazermos **p1=p2**. É interessante observar que se o objetivo for que a variável apontada por **p1** tenha o mesmo conteúdo da variável apontada por **p2** deve-se fazer ***p1=*p2**.

2. Operações Aritméticas com Ponteiros (continuação)

b) Incremento e Decremento → Quando incrementamos um ponteiro ele passa a apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta. Esta é uma razão pela qual o compilador precisa saber o tipo de um ponteiro.

2. Operações Aritméticas com Ponteiros (continuação)

b) Incremento e Decremento (continuação)

O decremento funciona de forma semelhante. Supondo que **p** é um ponteiro, as operações são escritas como:

```
p++;
p--;
```

2. Operações Aritméticas com Ponteiros (continuação)

b) Incremento e Decremento (continuação)

Estamos falando de operações com *ponteiros* e não de operações com o conteúdo das variáveis para as quais eles apontam. Por exemplo, para incrementar o conteúdo da variável apontada pelo ponteiro **p**, faz-se:

```
(*p)++;
```

Ponteiros

2. Operações Aritméticas com Ponteiros (continuação)

c) Soma e Subtração de Inteiros com Ponteiros → Vamos supor que você queira incrementar um ponteiro de 15 unidades. Basta fazer:

```
p=p+15;   ou   p+=15;
```

E se você quiser usar o conteúdo da memória apontada 15 posições adiante:

```
*(p+15);
```

A subtração funciona de forma similar.

2. Operações Aritméticas com Ponteiros (continuação)

d) Comparação entre dois Ponteiros → podemos saber se dois ponteiros são iguais ou diferentes (**==** e **!=**). No caso de operações do tipo **>**, **<**, **>=** e **<=** estamos comparando qual ponteiro aponta para uma posição mais alta *na memória*. A comparação entre dois ponteiros se escreve como a comparação entre outras duas variáveis quaisquer:

p1>p2

2. Operações Aritméticas com Ponteiros (continuação)

Há entretanto operações que ***não*** podemos efetuar sobre um ponteiro. Não se pode dividir ou multiplicar ponteiros, adicionar dois ponteiros, adicionar ou subtrair **float**s ou **double**s a ponteiros.

Exercícios:

a) Explique a diferença, caso exista, entre

p++; (*p)++; *(++p);

b) O que quer dizer *(p+10)?