Ponteiros

- Ponteiros guardam endereços de memória.
- Um ponteiro também tem tipo. No C quando declaramos ponteiros nós informamos ao compilador para que tipo de variável vamos apontá-lo. Por exemplo, Um ponteiro **int** aponta para um inteiro, isto é, guarda o endereço de um inteiro.

Declarando e Utilizando Ponteiros

- Para declarar um ponteiro temos a seguinte forma geral:

  *tipo_do_ponteiro *nome_da_variável;*

- É o asterisco (*) que faz o compilador saber que aquela variável não vai guardar um valor mas sim um endereço para aquele tipo especificado.

Exemplos:

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

Declarando e Utilizando Ponteiros

- Ponteiros não inicializados apontam para um lugar indefinido.
- Os ponteiros devem ser inicializados (apontado para algum lugar conhecido) antes de ser usados

Declarando e Utilizando Ponteiros

- Para atribuir um valor a um ponteiro recém-criado poderíamos igualá-lo a um valor de memória.
- Mas, como saber a posição na memória de uma variável do nosso programa?
- Podemos então deixar que o compilador faça este trabalho por nós. Para saber o endereço de uma variável basta usar o operador &. Veja o exemplo:

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

Declarando e Utilizando Ponteiros

- Para alterar o valor de uma variável apontado por um ponteiro, basta usar o operador *.

```
int count=10;
int *pt;
pt = &count;
*pt = 12;
```

- Resumindo,

*pt → o conteúdo da posição de memória apontado por *pt*;
&count → o endereço onde armazena a variável *count*.

Declarando e Utilizando Ponteiros

- Uma observação importante: apesar do símbolo ser o mesmo, o operador * (multiplicação) não é o mesmo operador que o * (referência de ponteiros). Para começar o primeiro é binário, e o segundo é unário pré-fixado.

Exemplo 1

```
#include <stdio.h>
void main ()
{
      int num, valor;
      int *p;
      num = 55;
      p = &num;     /* Pega o endereco de num */
      valor = *p;
           /* Valor e igualado a num de uma maneira indireta */
      printf ("\n\n%d\n", valor);
      printf ("Endereco para onde o ponteiro aponta: %p\n", p);
      printf ("Valor da variavel apontada: %d\n", *p);
}
```

%p indica à função que ela deve imprimir um endereço.

Exemplo 2

```
#include <stdio.h>
void main ()
{
      int num, *p;
      num = 55;
      p = &num;     /* Pega o endereco de num */
      printf ("\n Valor inicial: %d\n", num);
      *p = 100; /* Muda o valor de num de uma maneira indireta */
      printf ("\n Valor final: %d\n", num);
}
```

Operações aritméticas com ponteiros

- p1 = p2; p1 aponte para o mesmo lugar que p2;
- *p1 = *p2; a variável apontada por p1 tenha o mesmo conteúdo da variável apontada por p2;
- p++; passa a apontar para o próximo valor do mesmo tipo para o qual o ponteiro aponta. Isto é, se temos um ponteiro para um inteiro e o incrementamos ele passa a apontar para o próximo inteiro
- p--; funciona semelhantemente;

Operações aritméticas com ponteiros

- (*p)++; incrementar o conteúdo da variável apontada pelo ponteiro p;
- p = p+15; ou p+=15; incrementar um ponteiro de 15;
- *(p +15); usar o conteúdo do ponteiro 15 posições adiante;

Operações Relacionais com ponteiros

- == e != para saber se dois ponteiros são iguais ou diferentes;
- >, <, >= e <= estamos comparando qual ponteiro aponta para uma posição mais alta na memória. Então uma comparação entre ponteiros pode nos dizer qual dos dois está "mais adiante" na memória.

Operações Ilegais com ponteiros

- Há entretanto operações que você não pode efetuar num ponteiro. Você não pode dividir ou multiplicar ponteiros, adicionar dois ponteiros, adicionar ou subtrair **floats** ou **doubles** de ponteiros.

Ponteiros e Vetores

- Veremos nestas seções que ponteiros e vetores têm uma ligação muito forte.
- Quando você declara uma matriz da seguinte forma:

tipo_da_variável nome_da_variável [tam1][tam2] ... [tamN];

- O compilador C calcula o tamanho, em bytes, necessário para armazenar esta matriz. Este tamanho é:

$$tam_1 \times tam_2 \times tam_3 \times \ldots \times tam_N \times tamanho\_do\_tipo$$

- O nome da variável é um ponteiro que aponta para o primeiro elemento da matriz

Ponteiros e Vetores

- Então:

**nome_da_variável* é equivalente a *nome_da_variável[0]*

nome_da_variável[índice] é equivalente a **(nome_da_variável+índice)*

nome_da_variável é equivalente a *&nome_da_variável[0]*

&nome_da_variável[índice] é equivalente a *nome_da_variável + indice*

Ponteiros e Vetores

Exemplo: Varredura seqüencial de uma matriz

```
void main ()
{
        float matrx [50][50];
        int i, j;
        for (i = 0; i < 50; i++)
                for (j = 0; j < 50; j++)
                        matrx[i][j] = 0.0;
}
```

Ponteiros e Vetores

Podemos reescrevê-lo usando ponteiros:

```
void main ()
{
        float matrx [50][50];
        float *p;
        int count;
        p = matrx[0];
        for (count = 0; count < 2500; count++)
        {
                *p = 0.0;
                p++;
        }
}
```

Ponteiros e Vetores

- Há uma diferença entre o nome de um vetor e um ponteiro que deve ser frisada: um ponteiro é uma variável, mas o nome de um vetor não é uma variável. Isto significa, que não se consegue alterar o endereço que é apontado pelo "nome do vetor".

Ponteiros e Vetores

```
/* as operações abaixo são validas */
    int vetor[10];
    int *ponteiro, i;
    ponteiro = &i;
/* as operações a seguir são invalidas */
    vetor = vetor + 2;     /* ERRADO: vetor não e' variável */
    vetor++;               /* ERRADO: vetor não e' variável */
    vetor = ponteiro;      /* ERRADO: vetor não e' variável */
/* as operações abaixo são validas */
    ponteiro = vetor;      /* CERTO: ponteiro e' variável */
    ponteiro = vetor+2;    /* CERTO: ponteiro e' variável */
```

Ponteiros e Vetores

Exemplo

```
#include <stdio.h>
void main ()
{
        int matrx [10] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };
        int *p;
        p = matrx;
        printf ("O terceiro elemento do vetor e: %d",p[2]);
}
```

Podemos ver que p[2] equivale a *(p+2).

Ponteiros e Vetores

Escrevendo a nossa função StrCpy(), que funcionará de forma semelhante à função strcpy() da biblioteca:

```
#include <stdio.h>
void StrCpy (char *destino, char *origem)
{
        while (*origem)
        {
            *destino=*origem;
            origem++;
            destino++;
        }
       *destino='\0';
}
```

Ponteiros e Vetores

```
void main ()
{
        char str1[100], str2[100], str3[100];
        printf ("Entre com uma string: ");
        gets (str1);
        StrCpy (str2, str1);
        StrCpy (str3, "Voce digitou a string ");
        printf ("\n\n%s%s", str3, str2);
}
```

Vetores de ponteiros

- Podemos construir vetores de ponteiros como declaramos vetores de qualquer outro tipo. Uma declaração de um vetor de ponteiros inteiros poderia ser:

  ***int** *pmatrx [10];*

  No caso acima, *pmatrx* é um vetor que armazena 10 ponteiros para inteiros.

Ponteiros para Ponteiros

- Um ponteiro para um ponteiro é como se você anotasse o endereço de um papel que tem o endereço da casa do seu amigo.
- Podemos declarar um ponteiro para um ponteiro com a seguinte notação:

  tipo_da_variável ***nome_da_variável;*

- ***nome_da_variável* é o conteúdo final da variável apontada; **nome_da_variável* é o conteúdo do ponteiro intermediário.

Ponteiros para Ponteiros

Para acessar o valor desejado apontado por um ponteiro para ponteiro, o operador asterisco deve ser aplicado duas vezes, como mostrado no exemplo abaixo:

```
#include <stdio.h>
void main()
{
   float fpi = 3.1415, *pf, **ppf;
   pf = &fpi;                  /* pf armazena o endereco de fpi */
   ppf = &pf;                  /* ppf armazena o endereco de pf */
   printf("%f", **ppf);        /* Imprime o valor de fpi */
   printf("%f", *pf);          /* Tambem imprime o valor de fpi */
}
```

Cuidados a Serem Tomados

O principal cuidado ao se usar um ponteiro deve ser:

saiba sempre para onde o ponteiro está apontando.

```
void main () /* Errado - Nao Execute */
{
        int x, *p;
        x = 13;
        *p = x;
}
```