UDESC – Joinville
Departamento de Ciência da Computação

Ponteiros

Os **int**s guardam inteiros.
Os **float**s guardam números de ponto flutuante.
Os **char**s guardam caracteres.
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. Exemplo: Um ponteiro **int** aponta para um inteiro, isto é, guarda o endereço de um inteiro.

1. Declarando 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 e sim um endereço para aquele tipo especificado.

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

O primeiro exemplo declara um ponteiro para um inteiro.
O segundo declara dois ponteiros para caracteres.

Eles ainda não foram inicializados (como toda variável do C que é apenas declarada). Isto significa que eles apontam para um lugar indefinido. Este lugar pode estar, por exemplo, na porção da memória reservada ao sistema operacional do computador. Usar o ponteiro nestas circunstâncias pode levar a um travamento do micro, ou a algo pior.

O ponteiro deve ser inicializado (apontado para algum lugar conhecido) antes de ser usado!

2. Inicializando Ponteiros

Para atribuir um valor a um ponteiro recém-criado poderíamos igualá-lo a um valor de memória. Para saber o endereço de uma variável basta usar o operador **&**.

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

Criamos um inteiro **count** com o valor 10 e um apontador para um inteiro **pt**. A expressão **&count** nos dá o endereço de count, o qual armazenamos em **pt**.

3. Utilizando Ponteiros

Como nós colocamos um endereço em **pt**, ele está agora "liberado" para ser usado. Podemos, por exemplo, alterar o valor de **count** usando **pt**. Para tanto vamos usar o operador "inverso" do operador **&**. É o operador ***. No exemplo acima, 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**.

```
printf ("%d",count);
printf ("%d",*pt);
printf ("%p",pt);    //endereço de memória
```

Exemplos de usos de ponteiros:

```
#include <stdio.h>
int 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);
       return(0);
}
```

```
#include <stdio.h>
int main ()
{
       int num,*p;
       num=55;
       p=&num;     /* Pega o endereco de num */
       printf ("\nValor inicial: %d\n",num);
       *p=100; // Muda o valor de num de uma maneira indireta
       printf ("\nValor final: %d\n",num);
       return(0);
}
```

No primeiro exemplo, o código **%p** usado na função **printf()** indica à função que ela deve imprimir um endereço.

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

```
main()
{
  int var = 1;
  int *ptr;
  /* inicializa ptr para apontar para var */
  ptr = &var;
  printf(“\nAcesso direto, var = %d”, var);
  printf(“\nAcesso indireto, var = %d”, *ptr);
  /* Exibe o endereço da variável de duas maneiras*/
  printf(“\n\nEndereço de var = %d”, &var);
  printf(“\nEndereço de var = %d”, ptr);
}
```

4. Aritmética de Ponteiros

Podemos fazer algumas operações aritméticas com ponteiros.

A primeira, e mais simples, é igualar dois ponteiros. Se temos dois ponteiros **p1** e **p2** podemos igualá-los fazendo **p1=p2**. Repare que estamos fazendo com que **p1** aponte para o mesmo lugar que **p2**. Se quisermos que a variável apontada por **p1** tenha o mesmo conteúdo da variável apontada por **p2** devemos fazer ***p1=*p2**. Basicamente, depois que se aprende a usar os dois operadores (**&** e ***) fica fácil entender operações com ponteiros.

As próximas operações, também muito usadas, são o **incremento** e o **decremento**. Quando incrementamos um ponteiro ele 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. Esta é mais uma razão pela qual o compilador precisa saber o tipo de um ponteiro: se você incrementa um ponteiro **char*** ele anda 1 byte na memória e se você incrementa um ponteiro **double*** ele anda 8 bytes na memória. O decremento funciona semelhantemente. Supondo que **p** é um ponteiro, as operações são escritas como:

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

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)++;
```

Outras operações aritméticas úteis são a soma e subtração de inteiros com ponteiros. Vamos supor que você queira incrementar um ponteiro de 15. Basta fazer:

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

Para usar o conteúdo do ponteiro 15 posições adiante:

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

A subtração funciona da mesma maneira.

Uma outra operação, às vezes útil, é a comparação entre dois ponteiros. Em primeiro lugar, 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. Então uma comparação entre ponteiros pode nos dizer qual dos dois está "mais adiante" na memória. A comparação entre dois ponteiros se escreve como a comparação entre outras duas variáveis quaisquer:

```
p1>p2
```

No código a seguir, qual o valor de y no final do programa?

```
int main()
{
    int y, *p, x;
    y = 0;
    p = &y;
    x = *p;
    x = 4;
    (*p)++;
    x--;
    (*p) += x;
    printf ("y = %d\n", y);
    return(0);
}
```

5. Ponteiros e Vetores

5.1. Vetores como ponteiros

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: tam1 x tam2 x tam3 x ... x tamN x tamanho_do_tipo.

O compilador então aloca este número de bytes em um espaço livre de memória. O nome da variável que você declarou é na verdade um ponteiro para o tipo da variável da matriz. Tendo alocado na memória o espaço para a matriz, ele toma o nome da variável (que é um ponteiro) e aponta para o primeiro elemento da matriz.

```
int dados[10];
```

`dados ou &dados[0]` : aponta para o 1º elemento do vetor

Como podemos usar a seguinte notação? nome_da_variável[índice]

A notação acima é absolutamente equivalente a se fazer: *(nome_da_variável+índice)

No C, a indexação começa com zero porque, ao pegarmos o valor do primeiro elemento de um vetor, queremos, de fato, ***nome_da_variável** e então devemos ter um índice igual a zero. Então sabemos que:

*nome_da_variável é equivalente a nome_da_variável[0]

Vamos ver agora a varredura sequencial de um vetor. Quando temos que varrer todos os elementos de um vetor de uma forma sequencial, podemos

usar um ponteiro, o qual vamos incrementando. Considere o seguinte programa para zerar um vetor:

```
int main ()
{
	int vetor[50],i;
	for (i=0;i<50;i++)
		vetor[i]=0;
	return(0);
}
```

Utilizando ponteiros:

```
int main ()
{
	int vetor[50],*p,i;
	p = vetor;  // ou p = &vetor[0];
	for (i=0;i<50;i++)
	{
		*p=0;
		p++;
	}
	return(0);
}
```

No primeiro programa, cada vez que se faz **vetor[i]** o programa tem que calcular o deslocamento para dar ao ponteiro. Ou seja, o programa tem que calcular 50 deslocamentos. No segundo programa o único cálculo que deve ser feito é o de um incremento de ponteiro. Fazer 50 incrementos em um ponteiro é muito mais rápido que calcular 50 deslocamentos completos.

5.2. Ponteiros como vetores

O nome de um vetor é um ponteiro constante e podemos indexar o nome de um vetor. Como conseqüência podemos também indexar um ponteiro qualquer.

```
#include <stdio.h>
int 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]);
	return(0);
}
```

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

11. Bibliografias principais:

- Apostila de Linguagem C da UFMG disponível na internet em http://ead1.eee.ufmg.br/cursos/C/
- SCHILDT, H. **C Completo e Total.** 3ª ed. São Paulo: Makron Books Ltda, 1996. 827 p.
- DEITEL, H.M. & DEITEL, P.J. **Como Programar em C**. Rio de Janeiro: LTC – Livros Técnicos e Cie