Departamento de Ciência da COMPUTAÇÃO

UDESC Joinville

Linguagem C

ponteiros

André Tavares da Silva

andre.silva@udesc.br

Ponteiros

- Definição:
  - Em programação, um ponteiro ou apontador é um tipo de dado de uma linguagem de programação cujo valor se refere diretamente a um outro valor alocado em outra área da memória, através de seu endereço. Um ponteiro é uma simples implementação do tipo referência da Ciência da computação.
- Conceito similar ao trabalhar com vetores:

```
int i;
float vetor[10];
for(i=0; i<10; i++)
   vetor[i] = 0.0;
```

Ponteiros

- Um ponteiro é uma variável que armazena o endereço de outra variável.
- Iniciantes em programação normalmente têm dificuldade para entender o conceito de ponteiros.
- Imaginando a memória como sendo um grande vetor, um ponteiro nada mais é que uma variável semelhante àquelas que armazenam o índice de um vetor.

Ponteiros – Conceitos importantes

- Quando declaramos uma variável, informamos ao compilador duas coisas, o nome da variável e o tipo da mesma. Exemplo:

```
int n;
```

- Um bloco específico da memória do computador é alocado para guardar o valor da variável.
- O tamanho do bloco depende do intervalo de valores permitido à variável. Por exemplo, em um SO de 32 bits, o tamanho de uma variável do tipo int é de 4 bytes.

Ponteiros

- Um ponteiro, como qualquer variável, deve ter um tipo, que é *o tipo da variável para a qual ele aponta*. Para declarar um ponteiro, especificamos o tipo da variável para a qual ele aponta e *seu nome precedido por asterisco*:

```
int ponteiro;
int *ponteiro;
```

Ponteiros

- Para acessar o endereço de uma variável, utilizamos o operador & (e comercial), chamado "operador de referência" ou "operador de endereço". Como o nome sugere, ele retorna o endereço na memória de seu operando.

```
int  a;
int *ptr;
ptr = &a;
```

Ponteiros

- Para acessar o endereço de uma variável, utilizamos o operador & (e comercial), chamado "operador de referência" ou "operador de endereço". Como o nome sugere, ele retorna o endereço na memória de seu operando.

```
int  a;
int *ptr = &a;
```

Ponteiros

- Como o ponteiro contém um endereço, podemos também atribuir um valor à variável guardada nesse endereço, ou seja, à variável apontada pelo ponteiro. Para isso, usamos o operador * (asterisco), que basicamente significa "o valor apontado por":

```
int  a;
int *p = &a;
*p = 20;
printf (" a :%i\n", a);
```

Ponteiros

- Como o ponteiro contém um endereço, podemos também atribuir um valor à variável guardada nesse endereço, ou seja, à variável apontada pelo ponteiro. Para isso, usamos o operador * (asterisco), que basicamente significa "o valor apontado por":

```
int  a;
int *p = &a;
a = 20;
printf (" *p :%i\n", *p);
```

Operação com ponteiros

```
int *p1, *p2;

p1 = p2;
```

Operação com ponteiros

```
int *p1, *p2;

p1 = p2;
```

- Esse exemplo fará com que **p1** aponte para o mesmo lugar que **p2**. Ou seja, usar **p1** será equivalente a usar **p2** após essa atribuição.

Operação com ponteiros

```
*p1 = *p2;
```

Operação com ponteiros

```
*p1 = *p2;
```

- Nesse caso, estamos igualando os valores apontados pelos dois ponteiros: alteraremos o valor apontado por **p1** para o valor apontado por **p2**.

Operação com ponteiros

```
p++;
```

Operação com ponteiros

```
p++;
```

- Incremento do ponteiro. Quando incrementamos um ponteiro ele passa a apontar para o próximo valor do mesmo tipo em relação ao valor 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. *Note que o incremento não ocorre byte-a-byte*!

Operação com ponteiros

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

Operação com ponteiros

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

- Colocamos ***p** entre parênteses para especificar que queremos alterar o valor apontado por **p**. Ou seja, incrementar o **conteúdo** da variável apontada pelo ponteiro **p**.

Operação com ponteiros

```
x = *p++;
```

Operação com ponteiros

```
x = *p++;
```

- A precedência do operador ++ sobre o operador * faz com que a expressão seja equivalente a *(p++).
- A variável **x** recebe o valor atual do ponteiro **p** que em seguida aponta para o próximo valor.

Operação com ponteiros

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

Operação com ponteiros

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

- Esta linha atribui a uma variável **x** o conteúdo do décimo-quinto inteiro adiante daquele apontado por **p**.
- Por exemplo, suponhamos que tivéssemos um vetor **v** cujo primeiro elemento é apontado por **p**, seria o mesmo que escrever **`x = v[15]`**.

Ponteiros como parâmetros de funções

- Comecemos por uma situação-problema:
  - eu tenho 2 variáveis e quero trocar o valor delas.

```
void troca(int i, int j)
{
    int aux;
    aux = i;
    i = j;
    j = aux;
}
```

Ponteiros como parâmetros de funções

- Na função principal (*main*):

```
int main ()
{
   int a, b;
   a = 5;
   b = 10;
   printf ("\n\nEles valem %d, %d\n", a, b);
   troca (a, b);
   printf ("Eles agora valem %d, %d\n", a, b);
   return 0;
}
```

Ponteiros como parâmetros de funções

- Comecemos por uma situação-problema:
  - eu tenho 2 variáveis e quero trocar o valor delas.

```
void troca(int *i, int *j)
{
    int aux;
    aux = *i;
    *i = *j;
    *j = aux;
}
```

Ponteiros como parâmetros de funções

- Na função principal (*main*):

```
int main ()
{
   int a, b;
   a = 5;
   b = 10;
   printf ("\n\nEles valem %d, %d\n", a, b);
   troca (&a, &b);
   printf ("Eles agora valem %d, %d\n", a, b);
   return 0;
}
```

Ponteiros como parâmetros de funções

- Em C não existe explicitamente passagem de parâmetro por referência, apenas passagem de parâmetro por valor.
- Passagem de parâmetros por referência é realizada utilizando ponteiros.
- *scanf* é um exemplo de função que utiliza passagem de parâmetro "por referência". Ela envia o endereço da variável que receberá o valor informado pelo usuário.

sizeof()

- Ponteiros armazenam um valor para um determinado byte na memória. Quando usamos os operadores aritméticos com ponteiros, ele avança e retrocede automaticamente de acordo com o tamanho da variável utilizada (declaração do ponteiro).
- Algumas vezes é necessário endereçar os ponteiros através de equações ou operações complexas, sendo necessário conhecer o tamanho utilizado pelo tipo de dado endereçado.
- O operador sizeof() calcula e retorna o número de bytes ocupados em memória pelo tipo que foi passado como parâmetro.

```
int a = sizeof(int);
int b = sizeof(a);
```