Ponteiros

Manipulando um ponteiro

Já vimos o que significa a declaração de um ponteiro: é a alocação de um espaço de memória correspondente ao tamanho de um endereço (4 bytes no caso de máquinas de 32 bits). Vamos agora ver como podemos manipular um ponteiro.

Suponha que eu tenha o seguinte trecho de código:

```
int x = 2;
int *pf;
pf = &x;
```

Se quiséssemos modificar o valor armazenado na variável **x**, bastaria nós escrevermos uma linha tipo "**x = 5;**", que o valor armazenado em **x** passaria a ser o valor 5. Esquematicamente, teríamos o seguinte:

Endereço	RAM
1000	~~2~~ 5
1004	1000

Supondo que **x** fosse alocada no endereço 1000 e **pf** fosse alocada no endereço 1004.

Mas e se quiséssemos alterar o valor da variável **x** por meio do ponteiro **pf?** Pois, afinal, **pf** contém o endereço de **x** (ou como é mais normal de se dizer: "**pf** aponta para **x"**).

Para fazer isso, usaremos o operador unário "*" do seguinte modo:

***pf = -1**

Nessa linha, estamos pedindo para que o compilador C gere um código capaz de armazenar o valor "-1" no endereço de memória armazenado na variável **pf** (que no caso, corresponde ao endereço de memória da variável **x**). Esquematicamente, está sendo feito o seguinte:

Endereço	RAM
1000	~~2~~ ~~5~~ -1
1004	1000

Exercícios:

1. Para cada uma das linhas de código seguintes, diga o que representa o valor impresso na tela:
   a. int i = 2; printf("%i", i);
      Ex.: representa o valor inteiro armazenado na posição de memória alocada (4 bytes) para a variável i.
   b. char c = 'B'; printf("%i", c);
      Ex.: representa o valor inteiro armazenado na posição de memória alocada (4 bytes) para a variável c.
   c. char c = 'A', *pc; pc = &c; *pc = 66; printf("%c", *pc);
      Ex.: representa o caractere correspondente na tabela ASCII, ao valor do byte armazenado no endereço de memória relativo à variável c.
   d. char c = 'A', *pc; pc = &c; *pc = 66; printf("%c", c);
      Ex.: representa o caractere correspondente na tabela ASCII, ao valor do byte armazenado na variável c.
   e. char c = 'A', *pc; pc = &c; *pc = 66; printf("%p, %p", pc, &c);
      Ex.: representa o valor armazenado na posição de memória relativa à variável pc e ao endereço de memória da variável c.

2. Para cada uma das linhas de código seguintes, diga se ela é potencialmente problemática e, caso seja, diga o porquê:
    a. `int i = 2, *pi; pi = i;`
    b. `int i = 2, *pi; pi = (int *)i;`
    c. `int i = 2, *pi; *pi = i;`
    d. `int i = 2, *pi; *pi = &i;`
    e. `int i = 2, *pi; *pi = (int) &i;`