Ponteiros

Ponteiros de ponteiros

Ponteiro de ponteiro é uma variável ponteiro que armazena o endereço de uma outra variável ponteiro. Difícil? Vamos ver um trecho de código para clarear o conceito:

```
int x = 2; /* variável que é um inteiro */
int *pi;  /* variável que é um ponteiro para um inteiro */
int **ppi; /* variável que é um ponteiro de ponteiro para um inteiro */
```

Esquematicamente, na memória estaria acontecendo o seguinte:

Endereço	RAM
1000	2
1004	Lixo
1008	Lixo

Supondo que x foi alocada no endereço 1000, **pi** foi alocada no endereço 1004 e **ppi** foi alocada no endereço 1008.

Já sabemos como fazer **pi** conter o endereço de **x**, basta escrevermos a linha "pi = &x;". Esquematicamente ficaria o seguinte:

Endereço	RAM
1000	2
1004	1000
1008	Lixo

E **ppi**? O que faço com ela? Pela declaração, **ppi** é um ponteiro para um ponteiro de inteiro, logo, deve ser armazenado em **ppi** o endereço de uma variável que seja um "ponteiro para um inteiro". Em outras palavras, deve ser armazenada em **ppi**, o endereço de **pi**, pois **pi** é um ponteiro para um inteiro.

E como faço isso? Igual, ao que fez para armazenar o endereço de **x** em **pi**, basta escrever a seguinte linha: "**ppi = π**". Esquematicamente, ficaria:

Endereço	RAM
1000	2
1004	1000
1008	1004

Note, que a única diferença do que já fazia em relação a uma variável do tipo ponteiro é que, devemos armazenar em ppi o endereço de uma variável do tipo ponteiro (no caso, a variável **pi**) e não o endereço de uma variável que armazena um inteiro (no caso, a variável **x**).

E se quisesse alterar o valor da variável **x**, por meio de **ppi**? Simples, basta escrever a seguinte linha: "****ppi = -1**". O esquema da memória ficaria do seguinte modo:

Endereço	RAM
1000	~~2~~ -1
1004	1000
1008	1004

Veja se entendeu, diga o que vai ser impresso na tela para cada uma das seguintes linhas (assumindo o esquema de memória anterior):

- printf("%i", x);
- printf("%p", &x);
- printf("%i", *pi);
- printf("%p", pi);
- printf("%p", &pi);
- printf("%i", **ppi);
- printf("%p", &ppi);
- printf("%p", ppi);
- printf("%p", *ppi);

Exercícios:

1. Para cada uma das linhas de código seguintes, diga o que representa o valor impresso na tela:
   a. char c = 65; printf("%i", c);
   b. char c = 65; printf("%c", c);
   c. char c = 65, *pc = &c; printf("%c", *pc);
   d. char c = 65, *pc = &c; printf("%i", *pc);
   e. char c = 65, *pc = &c; printf("%p", pc);
   f. char c = 65, *pc = &c; printf("%p", &pc);
   g. char c = 65, *pc = &c; printf("%p", pc, &c);
   h. char c = 65, *pc = &c, **ppc = &pc; printf("%c", **ppc);
   i. char c = 65, *pc = &c, **ppc = &pc; printf("%c", *pc);
   j. char c = 65, *pc = &c, **ppc = &pc; printf("%p", &ppc);
   k. char c = 65, *pc = &c, **ppc = &pc; printf("%p", ppc, pc);
   l. char c = 65, *pc = &c, **ppc = &pc; printf("%p", ppc, &pc);
   m. char c = 65, *pc = &c, **ppc = &pc; printf("%p", *ppc, pc);
   n. char c = 65, *pc = &c, **ppc = &pc; printf("%p", *ppc, &c);
   o. char c = 65, *pc = &c, **ppc = &pc; printf("%p", ppc, pc);
   p. char a = 65, b = 2, *pc1 = &a, *pc2 = &b, **ppc; ppc = &pc1; **ppc = (*pc1 + *pc2); printf("%c", **ppc); printf("%c", a);

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, **ppi; ppi = π **ppi = -1;
   b. int i = 2, *pi, **ppi; **ppi = i;