Ponteiros

Vetores de Ponteiros

Até o momento declaramos vetores de inteiros e de caracteres, mas também poderíamos ter declarado vetores de floats, double, etc. E também poderíamos ter escrito a seguinte declaração:

int *v[3];

O que essa linha significa? Vamos tentar deduzir com base no que já estudamos até o momento.

Primeiro será alocado na memória um espaço para 3 elementos. Mas elementos de que tipo? Conforme a declaração serão elementos do tipo "int *". Ou seja, será alocado um espaço de memória para 3 elementos do tipo ponteiro para inteiro!

Conclusão, a linha anterior declara um nome **v** que é um vetor de ponteiros!

Esquematicamente, a linha anterior compreende os seguintes passos:

1. Alocará na memória RAM um espaço de memória correspondente a 3 elementos do tipo "int *". No caso, 3 x 4 = 12 bytes.
2. Fará com que a palavra "v" seja o nome pelo qual se pode modificar e alterar os valores armazenados nesse espaço de memória de 12 bytes.

Vejamos como fica na memória essa variável **v**, supondo que esse espaço de memória alocado para "v" seja o endereço 1000:

Endereço	RAM
1000	Lixo
1004	Lixo
1008	Lixo

v é como se fosse sinônimo do número 1000.

A declaração de um vetor em C, essencialmente, diz que o nome "v" é o endereço de memória da 1ª posição do vetor na memória, ou equivalentemente, é o endereço do 1° elemento do vetor.

Apesar da semelhança com a nossa já conhecida declaração "**int v[3];**", inclusive no tamanho do espaço total alocado, a declaração "**int *v[3];**" aloca na memória espaço para 3 *ponteiros para um tipo inteiro*. Ou seja, a seguinte linha seguinte geraria um *warning* pelo C (a não ser que forçássemos via *casting*):

v[0] = 3;

Pois, o compilador está esperando que nas posições de v sejam armazenados endereços de variáveis inteiras. Como no exemplo a seguir:

```
int *v[3];
int a = 1, b = 2, c = 3;
*v = &a; *(v+1) = &b; v[2] = &c;
```

Vejamos como fica a memória ao executarmos esse trecho de código anterior (supondo que **a**, **b** e **c** foram alocadas nos endereços 1012, 1016 e 1020, respectivamente. E **v** foi alocado no endereço 1000):

Endereço	RAM
1000	1012
1004	1016
1008	1020
1012	1
1016	2
1020	3

Entendido isso, tudo o que valia para vetor de inteiros, continua valendo agora para um vetor de ponteiros.

E isso é extensível indefinidamente, como por exemplo, poderia ser declarado um vetor de ponteiros de ponteiros.

```
int **ppv[3];
ppv[0] = v; ppv[1] = &v[1]; ppv[2] = v + 2;
```

Vejamos como fica a memória ao executarmos esse trecho de código anterior (supondo que **ppv** foi alocado no endereço 1024):

Endereço	RAM
1000	1012
1004	1016
1008	1020
1012	1
1016	2
1020	3
1024	1000
1028	1004
1032	1008

1. Analise cada uma das linhas de código seguintes e diga o que é impresso na tela com base nas declarações e atribuições de variáveis deste documento:
    a. for (int i=0; i < 3; i++) printf("%d", **ppv[i]);
    b. printf("%p", ppv);
    c. printf("%p", &ppv);
    d. printf("%p", &ppv[0]);
    e. printf("%p", ppv[0]);
    f. printf("%p", *ppv[0]);
    g. printf("%p", **ppv);

h. printf("%p", **(ppv+1));
i. printf("%p", ***ppv); /* execute com e sem a opção "-Wall" */
j. printf("%p", ***(ppv+1)); /* execute com e sem a opção "-Wall" */

2. Mostre dois modos de reescrever a linha anterior de modo a utilizar a opção "-Wall" do gcc sem receber nenhuma mensagem de *warning* (**dicas**: *casting* e parâmetro formatação do printf):

printf("%p", ***ppv);

Observação: deve continuar a ser impresso na tela exatamente o mesmo valor.

3. Reescreva o laço seguinte de modo a tornar a sua execução mais rápida:

for (register int i=0; i < 3; i++) printf("%d", **ppv[i]);