**LISTA 19**

1. Desde que um ponteiro contenha o endereço de uma variável, é possível acessar esta variável **indiretamente** através do ponteiro. Suponha que **x** é uma **variável do tipo int** e que **px** é um **ponteiro**.

int x; int \*px;

O operador unário **&** fornece o endereço de um objeto, de forma que o comando:

px = &x;

atribui o endereço de **x** à variável **px**. Diz-se então que **px** “aponta” para **x**. O operador

**&** pode ser aplicado somente a variáveis e elementos de vetores.

O operador unário **\*** trata seu operando como um endereço e acessa este endereço para buscar o conteúdo da variável alvo. Então, se **y** também é um **int**,

int y; y = \*px;

atribui a **y** o conteúdo do objeto para o qual **px** aponta. Assim, a sequência

px = &x; y = \*px;

atribui a **y** o mesmo valor atribuído no comando

y = x;

Baseando-se nestas informações, indique o que acontece quando:

* 1. y = \*px + 1;

**y = x + 1;**

* 1. printf(“%d\n”, px);

**O endereço a qual px aponta será imprimido na tela em valor decimal;**

* 1. py = px; // py é ponteiro int

**py estara apontando para o mesmo local que px aponta;**

* 1. \*px += 1;

**x = x + 1;**

* 1. (\*px)++;

**x = x + 1;**

* 1. px++;

**O ponteiro ira aponter para o proximo endereço. Neste caso como é do tipo int o incremento sera de 2, pois int ocupa dois bytes.**

1. Em C o relacionamento entre ponteiros e vetores é tão estreito que ponteiros e vetores deveriam ser realmente tratados juntos. Qualquer operação que possa ser feita com índices de um vetor, pode ser feita com ponteiros. A versão com ponteiro será, em geral, mais rápida, mas, pelo menos para os iniciantes, mais difícil de compreender imediatamente.

A declaração

int a[10];

define um vetor **a** de tamanho 10, isto é, um bloco de 10 posições consecutivos

chamados de a[0], a[1], a[2],..., a[9]. A notação **a[i]** significa o elemento da i-ésima posição do vetor a partir do início do mesmo. Se **pa** é um ponteiro para um inteiro, declarado como segue:

int \*pa;

então a atribuição

pa = &a[0];

faz com que **pa** aponte para o zero-ésimo elemento de **a**, ou seja, **pa** contém o

endereço de **a[0]**. Já a atribuição

x = \*pa;

copiará o conteúdo de **a[0]** em **x**.

Baseando-se nestas informações, indique o que acontece quando:

1. \*(pa + 1);

**Será pego o conteudo do proximo vetor, neste caso a[1];**

1. pa + 1;

**Ao incrementar 1 neste ponteiro ele passara a apontar parar o proximo endereço de memoria, serão incrementados 2 bytes pois esse é o valor necessário para que se avance um int, pois um int tem 2 bytes. Logo pa estara apontando para a[1];**

1. pa = a;

**O mesmo que pa = &a[0], ou seja, pa vai armazenar o endereço do vetor 0 de a;**

1. pa++;

**Ira apontar para o proximo endereço de memoria, neste caso a proxima posição do vetor;**

1. printf(“%d”, \*(a + 2));

**O a sozinho indicaria &a[0] o mais dois faz avançar duas casas logo vai pra &a[2]. O asterisco tranforma o ponteiro numa variavel, mostrando então o que esta ali contido.**

**Sugestão**: Escreva um programa que execute tais instruções para facilitar seu entendimento.