Aula 08:
- Exercícios de Ponteiros

Prof. João Henrique Kleinschmidt
Material elaborado pelo prof. Jesús P. Mena-Chalco

3Q-20108

Exemplo 1

```
1  #include <stdio.h>
2  int main() {
3      int x1 = 2006;              // inteiro x1 recebe 2006
4      int *p1 = &x1;              // ponteiro p1 recebe endereço de x1
5      int x2 = 2017;              // inteiro x2 recebe 2017
6      int *p2 = &x2;              // ponteiro p2 recebe endereço de x2
7      char c = 'u';               // caractere c recebe 'u'
8      printf("%p\n", &c);         // imprime um endereço (algo como 0x7ffffe0c697f)
9      printf("%p\n", &x1);        // imprime um endereço (algo como 0x7ffffe0c6980)
10     printf("%p\n", &x2);        // imprime um endereço (algo como 0x7ffffe0c6984)
11     printf("%p\n", p1);
12     printf("%p\n", p2);
13     printf("%d\n", *p1 + *p2)
14     return 0;
15 }
```

Lembre-se de que na função printf(), %p é usado para imprimir endereços. Observe que o endereço de x1 é acessado por meio de p1 (um ponteiro) e também por &x1 (linhas 11 e 9, respectivamente). Outro ponto importante é que na declaração de ponteiros (linhas 4 e 6), usa-se * seguido do nome da variável. Caso não fosse atribuído um endereço, nesses dois casos, o recomendado seria atribuir o valor NULL, como segue: int *p1 = NULL. Por outro lado, o uso de * fora de atribuições é feito para acessar o conteúdo para o qual um determinado ponteiro aponta (linha 13).

Pergunta: O que as linhas 11, 12 e 13 do Exemplo 1 imprimem?

Exemplo 2

```
1  #include <stdio.h>
2  int main() {
3      float *p3 = NULL;
4      float x3 = 100.0;
5      p3 = &x3;
6      printf("%f\n", *p3);        // imprime o conteúdo apontado por p3
7      *p3 = 10.5;
8      printf("%f\n", *p3);        // imprime o conteúdo apontado por p3
9      printf("%f\n", x3);         // imprime o valor de x3
10     printf("%p\n", &p3);        // imprime o endereço de p3
11     printf("%p\n", p3);         // imprime o endereço p3
12     printf("%p\n", &x3);        // imprime o endereço de x3
13     return 0;
14 }
```

As linhas 10, 11 e 12 do Exemplo 2 imprimem endereços. Observe a diferença sutil entre as linhas 10 e 11. A primeira imprime o endereço do ponteiro p3 (pois ponteiros também são variáveis, e por isso, também possuem endereço) e a segunda imprime o endereço que p3 armazena (o conteúdo de p3 é o endereço de x3, ou seja, &x3).

Pergunta: (a) Os endereços impressos nas linhas 10, 11 e 12 são iguais? (b) Qual é o valor final de x3?

Exemplo 3

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main() {
4      int n;
5      printf("n = ");
6      scanf("%d", &n);
7      int *v = malloc(n * sizeof(int));
8      for(int i = 0; i < n; i++)
9          v[i] = i;
10     printf("n[0]: %d\n", *(v));
11     printf("n[0]: %d\n", v[0]);
12     printf("n[n-1]: %d\n", *(v + n - 1));
13     printf("n[n-1]: %d\n", v[n - 1]);
14     free(v);
15     v = NULL;
16     return 0;
17 }
```

Na linha 7 malloc aloca n vezes o tamanho do tipo int e retorna o endereço desse bloco. A variável v e um ponteiro para int que recebe esse endereço, que e na prática, o endereço do primeiro elemento do vetor de int. Na versão estática, para n = 100, seria o mesmo que fazer int v[100], sendo o primeiro elemento representado por v[0]. Cabe lembrar que usando a notação de ponteiros, *(v) aponta para o primeiro elemento do bloco alocado. Por exemplo, para acessar o conteúdo da quinta posição pode-se fazer *(v + 5). Para acessar o conteúdo da última posição, pode-se fazer *(v + n - 1). Seria o mesmo que fazer v[5] ou v[n - 1].

Exemplo 4

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  int main() {
4      int *v = malloc(10 * sizeof(int));  // início de alocação de vetor
5      for (int i = 0; i < 10; i++){
6          *(v + i) = i * 5;
7          printf("v[%d] : %d (%p)\n", i, *(v + i), v + i);
8      }
9      int **m;                                // início de alocação de matriz
10     m = malloc (4 * sizeof (int *));        // o endereço para o bloco alocado
11     for (int i = 0; i < 4; i++)
12         *(m + i) = malloc (5 * sizeof (int));
13     for (int i = 0; i < 4; i++){            // percorrendo matriz
14         for (int j = 0; j < 5; j++){
15             m[i][j] = (i * 5) + j;          // atribui um valor qualquer a m[i][j]
16             printf("m[%d][%d] = %d (%p)\n", i, j, m[i][j], &m[i][j]);
17         }
18     }
19     return 0;
20 }
```

Neste exemplo é alocada memória suficiente para armazenar 10 elementos do tipo int em um vetor (linhas 4 a 8), e também 20 elementos do tipo int em uma matriz com 4 linhas e 5 colunas (linhas 9 a 12). A linha 10 aloca um bloco com que ocupa 4 vezes o tamanho de um ponteiro para int. Observe os diferentes usos do operador *: (i) pode indicar a declaração de ponteiro, (ii) o acesso ao conteúdo apontado por um ponteiro, e (iii) uma simples multiplicação.