INSTITUTO FEDERAL DE GOIÁS
CÂMPUS GOIÂNIA
BACHARELADO EM SISTEMAS DE INFORMAÇÃO
TEORIA GERAL DA ADMINISTRAÇÃO

Nome do Aluno: Luiz Antônio Rodrigues dos Santos Data: 07/11/2017
Prof. Renan Rodrigues

Passagens de Parâmetros, Endereços e Ponteiros em C

1. Responda as seguintes questões:

a) Qual a importância da modularização de algoritmos?

R: Melhor entendimento e legibilidade do código, fluxo de execução mais claro, evita redundância no código, também se torma mais fácil encontrar erros, documentar e reutilizar o código e facilita dar manutenção nos programas.

b) Qual a diferença entre funções e procedimentos?

R: A função obrigatoriamente retorna um valor, o procedimento não retorna nenhum valor só executa uma ação.

c) Caracterize os métodos de passagem de parâmetros por valor e por referência.

R: Na passagem de parâmetro por valor, uma cópia do valor do argumento é passado para a função, as alterações nos parâmetros não tem efeito nas variáveis usadas para chamá-la, enquanto na passagem de parâmetro por referência, é passado um ponteiro que receberá o endereço de memória como parâmetro e a função poderá alterar o conteúdo das variáveis usadas para chamá-la.

d) O que são ponteiros? Cite exemplo de uso.

R: Ponteiro é uma variáveis que contém endereço de memória. Pode ser usado para fornecer meios pelos quais funções podem modificar seus argumentos, suportar rotinas de alocação dinâmica em C e pode também aumentar a eficiência de certas rotinas.

2. Suponha que os elementos de um vetor v são do tipo int e cada int ocupa 4 bytes no seu computador. Se o endereço de v[0] é 55000, qual o valor da expressão v+3?

R: 55012

3. Suponha que v é um ponteiro para um vetor. Descreva a diferença conceitual entre as expressões v[3] e v+3.

R: v[3] contém o valor do vetor na posição 4, e v+3 contém e o endereço de memória do vetor na posição 4.

4. Seja p um ponteiro para um int. Explique a diferença entre p++; (*p)++; *(p++).

R: p++ vai incrementar o ponteiro, após essa instrução o ponteiro apontará pra o próxima posição superior; (*p)++ vai incrementar o valor do conteúdo da variável que p está apontando; *(p++) incrementa p e acessa o valor encontrado na nova posição.

5. Assumindo que v é um ponteiro para um vetor do tipo int, explique o significado de cada uma das seguintes expressões: *(v+2); *(v+4); v+4; v+2.

R: *(v+2) contém o valor do conteúdo vetor na posição 3; *(v+4) contém o valor do conteúdo vetor na posição 5; v+4 contém o endereço de memória do vetor na posição 5; v+2 contém o endereço de memória do vetor na posição 3.

6. Qual é o valor de count após a execução do seguinte código:

```
#include <stdio.h>
int main(int argc, char *argv[]) {
     int x, *y, count;
     x=100;
     count=999;
     y= &x;
     count= *y;
     return 0;
}
```

R: 100

7. Escreva um procedimento em C com protótipo

void operacoes(int a, int b, int *soma, int *subtracao); que receba dois números inteiros e devolva a soma dos dois números em *soma e a subtração em *subtracao. Crie o programa principal para testar a implementação.

R:

```
void operacoes(int a, int b, int *soma, int *subtracao){
    *soma = a + b;
    *subtracao = a - b;
}
int main(int argc, char *argv[]) {
    int soma, subtracao;
    int a = 100, b = 50;

    operacoes(a, b, &soma, &subtracao);

    printf("Soma: %d\n", soma);
    printf("Subtracao: %d\n\n", subtracao);
}
```

8.Qual é o valor de *p e *q após a execução do seguinte código:

#include <stdio.h>

```
int main(int argc, char *argv[]) {
     int a=3, b=2, *p = NULL, *q = NULL;
     p = &a;
     q = p;
     *q = *q +1;
     q = &b;
     b = b + 1;
     return 0;
}
```

R: *p vale 4 e *q vale 3

9. Verifique o programa abaixo. Encontre o seu erro e corrija-o para que escreva o número 10 na tela.

```
#include <stdio.h>
int main(int argc, char *argv[]) {
     int x, *p, **q;
     p = &x;
     q = &p;
     x = 10;
     printf("\n%d\n", &q);
     return 0;
}
```

R:

```
int main(int argc, char *argv[]) {
     int x, *p, **q;
     p = &x;
     q = p;
     x = 10;
     printf("\n%d\n", *q);
     return 0;
}
```

10. Reescreva o programa abaixo usando ponteiros, ajustando a declaração do vetor através de alocação dinâmica e os acessos ao vetor através da aritmética de ponteiros.

```
#include <stdio.h>
#include<stdlib.h>
#define MAXN 10
int main(int argc, char *argv[]) {
     int m[MAXN][MAXN];
     for(int i=0; i<MAXN; i++) {
          for(int j=0; j<MAXN; j++) {
          m[i][j] = 0;
          }
     }
     for(int i=0; i<MAXN; i++) {
          for(int j=0; j<MAXN; j++) {
          printf("%d ", m[i][j]);
          }
          printf("\n");
     }
     return 0;
}
```

R:

```
int main(int argc, char *argv[]) {
    int *m;

    m = calloc(MAXN*MAXN, sizeof(int));

    if (!m){
        printf("Calloc não foi definido.\n");
    }

    for(int i = 0; i < MAXN; i++){
        for (int j = 0; j < MAXN; j++){
            *(m+(i*MAXN+j)) = 0;
        }
     }

     for(int i = 0; i < MAXN; i++){
        for (int j = 0; j < MAXN; j++){
            printf("%d ", *(m+(i*MAXN+j)) );
        }
        printf("\n");
     }
}
```