MC102 – ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES

Lista 6

OBS1:*Nos exercícios abaixo, quando falarmos de passagem por referência assuma que estamos interessados em modificar a variável passada por parâmetro e portanto você deve passar o endereço desta como argumento.*
OBS2: *Lembre-se que vetores em C são sempre passados por "referência".*

1. Suponha que criamos uma estrutura para armazenar produtos de um supermercado:

```
struct Produto{
   char nome[80];
   double preco;
   int quantidade;
};
```

   Implemente duas funções, uma que devolve o vetor ordenado por preços e outra que devolve o vetor ordenado pela quantidade de itens no estoque. Funções **void ordenaPreco(struct Produto vet[], int n)** e **void ordenaQuant(struct Produto vet[], int n)**.

2. Suponha que criamos uma estrutura para armazenar Datas:

```
struct Data{
   int dia;
   int mes;
   int ano;
};
```

   Implemente um algoritmo que receba um vetor de Datas como parâmetro e que retorne as datas em ordem cronológica (crie uma função com cabeçalho: **void ordena(struct Data vet[], int tam)**.
   Dica: Ordene o vetor separadamente por cada um dos campos.

3. Suponha que criamos uma estrutura para armazenar dados de pessoas e um vetor para armazenar dados de várias pessoas:

```
struct Pessoa{
   int rg;
   int cpf;
   char nome[80];
};

struct Pessoa cadastro[100];
```

Suponha que o vetor esteja ordenado em ordem crescente por valor de RG. Implemente uma função de busca por RG, que opera como a busca binária, e que caso exista uma pessoa no cadastro com o RG a ser buscado, devolve o índice deste cadastro e caso não exista o RG a ser buscado, devolve -1.

4. Refaça as funções de busca sequencial e busca binária vistas em aula assumindo que o vetor possui chaves que podem aparecer repetidas. Neste caso, você deve retornar em um outro vetor todas as posições onde a chave foi encontrada.
   Protótipo da função: **void busca(int vet[], int tam, int chave, int posicoes[], int *n)**

   - Você deve devolver em **posicoes[]** as posições de **vet** que possuem a **chave**, e devolver em ***n** o número de ocorrências da chave.
     - **OBS:** Na chamada desta função, o vetor **posições** deve ter espaço suficiente (por exemplo, tam) para guardar todas as possíveis ocorrências da chave.

5. Determine o valor especificado em cada item abaixo considerando que foi executado as seguintes instruções (assuma que o endereço de x é 1000 e de y é 1004):

```
int x = 10, y=20;
int* p1;
int* p2;
p1 = &x;
p2 = &y;
(*p1)++;
```

   (a) x
   (b) y
   (c) &x
   (d) &y
   (e) p1
   (f) p2
   (g) *p1 + *p2
   (h) *(&x)
   (i) &(*p2)

6. O que será impresso pelo programa abaixo:

```
#include <stdio.h>

struct T{
  int x;
  int y;
};
typedef struct T  T;
```

```
void f1(T *a);
void f2(int *b);

int main(){
  T a, b, *c, *d;
  c = &a;
  a.x = 2;
  a.y = 4;
  b.x = 2;
  b.y = 2;
  d = c;
  f1(d);
  b = *d;
  printf("x: %d --- y: %d\n",b.x,b.y);
}


void f1(T *a){
  f2(&(a->x));
  f2(&(a->y));
}

void f2(int *b){
  *b = 2*(*b);
}
```

7. Escreva uma função chamada **teste** que recebe um valor n passado por valor e dois inteiros b e k passados por "referência". Sua função deve retornar em b e k valores tal que $b^k = n$ e b seja o menor possível.

8. Faça uma função chamada **primo** que recebe como parâmetro um inteiro m e dois outros inteiros $p1$ e $p2$ passados por referência. A função deve retornar em $p1$ o maior número primo que é menor do que m e deve retornar em $p2$ o menor número primo que é maior do que m.

9. Faça uma função chamada **media** que recebe um vetor de *double*, um inteiro n que indica o tamanho do vetor, e um inteiro i passado por referência. A função deve retornar a média dos n elementos no vetor e no inteiro i, passado por referência, deve retornar a posição do elemento que tem o valor mais próximo da média.
   Cabeçalho: double media(double vet[], int *i);.