Algoritmos e Programação I: Lista de Exercícios 06 - Orientações para submissão no Moodle e BOCA. *

Faculdade de Computação
Universidade Federal de Mato Grosso do Sul
79070-900 Campo Grande, MS
`http://moodle.facom.ufms.br`

Após submeter no BOCA, não se esqueça de enviar o programa para o Moodle, usando o seguinte nome: pxxloginname.c, onde xx é o número do problema e o loginname é o seu login.

Compile o seu programa usando:

```
gcc -Wall -pedantic -std=c99 -o programa programa.c [-lm]
```

A flag `-lm` deve ser usada quando o programa incluir a biblioteca `math.h`.

Uma forma eficiente de testar se o seu programa está correto é gerando arquivos de entrada e saída e verificar a diferença entre eles. Isso pode ser feito da seguinte forma:

```
./programa < programa.in > programa.out
```

O conjunto de entrada deve ser digitado e salvo num arquivo `programa.in`. O modelo da saída deve ser digitado e salvo num arquivo `programa.sol`.

Verifique se a saída do seu programa `programa.out` é EXATAMENTE igual ao modelo de saída `programa.sol`. Isso pode ser feito com a utilização do comando `diff`, que verifica a diferença entre dois arquivos.

```
diff programa.out programa.sol
```

O seu programa só está correto se o resultado de `diff` for vazio (e se você fez todos os passos como indicado).

A solução dos exercícios deve seguir a metodologia descrita em sala:

a. Diálogo

b. Saída/Entrada (pós e pré)

c. Subdivisão

d. Abstrações

*Este material é para o uso exclusivo da disciplina de Algoritmos e Programação I da FACOM/UFMS e utiliza as referências bibliográficas da disciplina (B. Forouzan e R. Gilbert, A. B. Tucker et al. e S. Leestma e L. Nyhoff, Lambert et al., H. Farrer et al., K. B. Bruce et al., e Material Didático dos Profs. Edson Takashi Matsubara e Fábio Viduani Martinez.

e. Implementação

f. Teste

Na submissão ao BOCA não se esqueça de comentar todos os `printf`'s da entrada e que a saída não pode conter acentuação ou *ç*.

Exercícios

1. Assuma que as seguintes declarações tenham sido feitas:

```
// Declaração de tipos
typedef enum {vermelho, amarelo, azul, verde, branco, preto} Cores;
typedef enum {A, B, C, D, E, F} Indices;
typedef float VetorCores[6];
typedef int Vetor[10];
typedef char VetorContaLetra[6];

// Declaração das variáveis
VetorCores        Preco;
Vetor             Numero;
VetorContaLetra   ContaLetra;
unsigned int      i;
char              Ch;
Cores             Cor;
```

Para cada um dos seguintes itens, diga qual o valor (se algum) será atribuído para cada elemento do vetor ou explique porque ocorre erro.

a.
```
for (i = 0; i <= 9; i++) {
    Numero[i] = i / 2;
} // fim for
```

b.
```
for (i = 0; i <= 5; i++)
   Numero[i] = i * i;
for (i = 6; i <= 9; i++)
   Numero[i] = Numero[i-5];
```

c.
```
i = 1;
while (i != 9) {
   if ((i % 3) == 0)
      Numero[i] = 0;
   else
      Numero[i] = i;
   i++;
} // fim while
```

d.
```
Numero[0] = 1;
i = 1;
do {
   Numero[i] = 2 * Numero[i-1];
   i++;
} while (i != 9);
```

e.
```
for (Cor = amarelo; Cor <= branco; Cor++) {
   Preco[Cor] = 13.95;
} // fim for
```

f.
```
for (Cor = vermelho; Cor <= preto; Cor++) {
   switch(Cor) {
      case vermelho:
      case azul:
         Preco[Cor] = 19.95;
         break;
      case amarelo:
         Preco[Cor] = 12.75;
         break;
      case verde:
      case preto:
         Preco[Cor] = 14.50;
         break;
      default:
         Preco[Cor] = 13.95;
   } // fim switch
} // fim for
```

2. Para cada um dos seguintes itens, escreva instruções e declarações apropriadas para criar o array especificado:

   a. Um array com 5 inteiros, cada elemento do array tem o mesmo valor do índice.

   b. Um array com 10 inteiros, cada elemento do array tem o valor em ordem inversa a dos índices.

   c. Um array com 20 valores `bool`, cada elemento do array tem o valor `true` se o índice correspondente é par, e `false` caso contrário.

   d. Um array cujos índices são as cinco disciplinas algoritmos, estatística, geometria, cálculo e álgebra, e os elementos do array são as notas recebidas nas disciplinas: 9.0 em algoritmos e geometria, 7.5 em estatística, 4.0 em cálculo, e 5.5 em álgebra.

3. Assumindo que os valores do tipo `char` são armazenados em 1 byte, `unsigned int` e `int` em dois bytes, e `float` em 4 bytes, calcule quantos bytes serão necessários para armazenar cada um dos arrays abaixo.

   a. `int A[5];`

   b. `float A[5];`

   c. `char A[10];`

   d. `unsigned int A[26];`

4. (`ordenar.c`) [etm] Dado um vetor de tamanho m de números naturais (ou seja, apenas números inteiros positivos, incluindo o 0), imprimir o vetor ordenado.

   Entrada: A primeira linha contém um inteiro m equivalente ao tamanho do vetor, seguido na próxima linha do vetor a ser ordenado.

```
ordenar.in
10
5 2 3 9 7 7 15 20 19 0
```

Saída: A saída consiste em escrever o vetor ordenado de forma crescente. Obs: Não existe quebra de linha após o resultado da saída.

```
ordenar.sol
0 2 3 5 7 7 9 15 19 20
```

5. (`paralelo.c`) [etm] Um vetor é paralelo a outro vetor caso exista um número real k, onde:

$$k * a[i] = b[i], \text{ para todo } i \text{ inteiro.}$$

Exemplo:

$A = [1, 2, 3]$ é paralelo ao vetor $B = [2, 4, 6]$, pois se $k = 2$, então temos:

2*1=2
2*2=4
2*3=6

logo A e B são paralelos.

Dado n pares de vetores de tamanho 5 cada, verificar se os vetores são paralelos ou não.

Entrada: A primeira linha contém um inteiro m equivalente a quantidades de pares de vetores de tamanho 5, seguido dos pares de vetores, cada par numa linha.

```
paralelo.in
2
1 2 3 4 5 2 4 6 8 10
1 1 1 1 1 2 2 2 2 3
```

Saída: A saída consiste em imprimir 1 caso o par de vetores seja paralelo e 0 ao contrário.

```
paralelo.sol
1
0
```

6. (`semrepeticao.c`) [etm] Dado um vetor de tamanho n, imprima todos os números do vetor sem repetição:

Entrada: A primeira linha contém um inteiro n indicando o tamanho do vetor, seguido dos números que consistem o vetor.

```
semrepeticao.in
5
1 2 2 3 1
```

Saída: Imprimir todos os elementos do vetor sem repetição na ordem dada na entrada. Obs: Não existe quebra de linha após o resultado da saída.

```
semrepeticao.sol
1 2 3
```

7. (`corrida.c`) [etm] Durante uma corrida de automóveis com $n \leq 20$ voltas de duração foram anotados para um piloto, na ordem, os tempos registrados em cada volta. Fazer um programa em `C` para ler os tempos das n voltas, calcular e imprimir:

   - a volta em que o melhor tempo ocorreu;
   - tempo médio das n voltas.

   Entrada: Um inteiro $n \leq 20$ indicando o números de voltas seguido de um vetor com n números reais, indicando o tempo em minutos de cada uma das voltas.

```
corrida.in
3
1.5 1.7 1.25
```

   Saída: A melhor volta e o tempo médio das n voltas. Ambos com precisão de duas casas decimais após a vírgula. Obs: Não existe quebra de linha após o resultado da saída.

```
corrida.sol
1.25 1.48
```

8. (`megasena.c`) [etm] Faça um programa que leia 6 números inteiros positivos num vetor `resultado[6]`, que é o resultado de um sorteio da mega-sena. Em seguida, leia outros 6 números inteiros positivos num vetor `aposta[6]`, que é a aposta de um jogador. O seu programa deve comparar os valores do resultado e da aposta e imprimir a aposta, o resultado do sorteio e o número de acertos do apostador.

   Entrada:A primeira linha contém o resultado da mega-sena e a segunda linha contém a aposta do jogador.

```
megasena.in
55 23 01 05 06 29
01 03 21 55 23 11
```

   Saída: A saída consiste em escrever na primeira linha a aposta do jogador, na segunda linha o resultado da mega-sena e na terceira linha o número de acertos do jogador. Perceba que, após a saída da quantidade de acertos do jogador deve-se pular uma linha.

```
semrepeticao.sol
1 3 21 55 23 11
55 23 1 5 6 29
3
```

9. (`sequencia.c`) [etm] Dada m sequências de n números, imprimi-las na ordem inversa à da leitura.

   Entrada: A primeira linha contém o número de sequências (m), na segunda linha o tamanho das sequências (n) e nas próximas linhas, as m sequências de tamanho n, onde $0 < n \leq 100$.

   ```
   sequencia.in
   2
   4
   10 20 30 40
   5 2 1 9
   ```

   Saída: A saída consiste em escrever as m sequências na ordem inversa a entrada. Obs: Não existe quebra de linha após o resultado da saída.

   ```
   sequencia.sol
   40 30 20 10
   9 1 2 5
   ```

10. (`somanumero.c`) [etm] Dadas n sequências de dois vetores com k números inteiros entre 0 e 9, $0 \leq k \leq 100$, interpretadas como dois números inteiros de k algarismos, calcular a soma dos dois números inteiros.

    Exemplo:

    ```
    vetor 1:  1 2 3 4
    vetor 2:  2 3 4 6
    soma:  3 5 8 0
    ```

    Entrada: A primeira linha contém um inteiro n que indica o número de sequências de dois vetores. As n linhas seguintes, contém um número k, que equivale ao tamanho do vetor, seguido dos dois vetores.

    ```
    somanumero.in
    1
    8
    8 2 4 3 4 2 5 1
    3 3 7 5 2 3 3 7
    ```

    Saída: Escrever a soma dos dois números, representados pelos vetores das n sequências de dois números (vetores) dadas no formato abaixo.

    ```
    semrepeticao.sol
    1 1 6 1 8 6 5 8 8
    ```

11. Escreva uma função `ElementoMax` para encontrar o maior valor em um array de inteiros `Número`.

12. Escreva uma função para localizar e retornar o maior e o menor inteiro em um array `Número` e suas posições no array. A função deve também retornar a variação dos números, isto é a diferença entre o maior e o menor.

13. Escreva uma função para implementar o algoritmo de busca linear abaixo.

```
Algoritmo para efetuar uma busca linear em uma lista X[0],X[1],...,
X[NumItens] para um Item especificado. Se a busca teve sucesso,
true é atribuído à Encontrou e a posição do Item é atribuído à
Localização; caso contrário, false é atribuído à Encontrou.
Passo 1. Inicialize Posição igual a 0 e Encontrou igual a false
Passo 2. enquanto Localização <= NumItens && !Encontrou faça
            se Item == X[Localização]
               Atribua true para Encontrou
            caso contrário
               Incremente Localização por 1
```

14. Escreva uma função `Insere` para inserir um item em uma posição especificada de uma lista implementada como um array e uma função `Remove` para remover um item em uma posição especificada.

15. (primos.c)[fhvm] Primos entre si - Dois números inteiros são **primos entre si** quando não existe um divisor maior do que 1 que divida ambos. Isso significa que o máximo divisor comum de dois números que são primos entre si é igual a 1. Por exemplo, 4 e 9 são primos entre si já que o maior divisor comum entre eles é 1. Por outro lado, 5 e 20 não são primos entre si.

    a. Escreva uma função com a seguinte interface:

    ```
    int mdc(int a, int b)
    ```

    que receba dois números inteiros a e b e devolva o máximo divisor comum entre eles, usando o algoritmo de Euclides.

    b. Escreva um programa que receba diversas sequências de números inteiros e conte, para cada sequência, a quantidade de pares de números que são primos entre si. Cada sequência contém um primeiro número inteiro n, tal que $0 \leq n \leq 100$, e uma lista de n números inteiros. A última lista contém $n = 0$.

    [**Entrada:**]

    ```
    primos.in
    3 4 9 10
    8 2 3 5 7 11 13 17 19
    0
    ```

    [**Saída:**]

    ```
    primos.sol
    2
    28
    ```