Lista de Exercícios 3 de Introdução à Programação
Professor: Marcos Costa
Monitores: Felipe, Márcio

Orientações a serem seguidas:

a) a lista é individual e deverá ser entregue e apresentada em data a ser informada posteriormente;
b) a entrada dos dados e a saída dos programas será feita utilizando arquivos – a questão exemplo mostra como fazer;
c) tire suas as dúvidas da lista preferencialmente com os monitores;
d) faça a lista progressivamente, dia após dia (não deixe para a "última hora");
e) as entradas e as saídas são padronizadas – observe atentamente a padronização nas questões;
f) não valide as entradas, exceto quando for solicitado na questão;
g) os arquivos de entrada mostrados nas questões são apenas exemplos – de maneira que, no momento de realizar testes, poderão ser utilizados outros arquivos, seguindo sempre a formatação indicada;
h) a entrega de questões fora dos padrões implica na perda parcial ou total da mesma;
i) deverão ser entregues (em CD ou pen-drive) apenas os fontes (arquivos com extensão .c) utilizando o seguinte padrão:
– será criada uma pasta para cada questão (q1, q2, q3, etc.) dentro de uma pasta raiz chamada L2C;
– dentro de cada pasta deverá estar o arquivo correspondente à resposta da questão (q1.c, q2.c, q3.c, etc.).
j) Os arquivos de entrada e saída de cada questão, bem como os executáveis não devem ser enviados;
k) na primeira linha de cada código fonte deve necessariamente haver um comentário de linha contendo o nome do autor da questão;
l) os códigos fonte não devem ter system("PAUSE").
m) A utilização de entrega fora das orientações estabelecidos nos itens acima implicarão em perda de pontos ou ainda da questão inteira, dependendo do caso.

Introdução à Programação: Lista 2

Questão exemplo: Escreva um programa que lê um número inteiro e imprime o número lido multiplicado por dois.

7

Exemplo de arquivo texto de entrada da questão (de nome e.txt), contendo apenas o número a ser lido – deve ser colocado, no projeto, no mesmo diretório do arquivo fonte .c

```
#include <stdio.h>
int main(void) {
	int a;
	/**
	* A função freopen redireciona a entrada padrão (por default, o teclado), para o arquivo "e.txt".
	* O "r" do segundo parâmetro significa que o arquivo "e.txt" será utilizado para leitura.
	*/
	freopen("e.txt", "r", stdin);
	/**
	* A função freopen redireciona a saída padrão (por default, o monitor), para o arquivo "s.txt".
	* O "w" do segundo parâmetro significa que o arquivo "s.txt" será utilizados para a escrita.
	*/
	freopen("s.txt", "w", stdout);
	//observe que, o scanf e o printf são utilizados normalmente.
	scanf("%d", &a);
	printf("%d", 2*a);
	/**
	* A função fclose é utilizada para fechar os arquivos abertos, que não
	* vão mais ser utilizados.
	*/
	fclose(stdin);
	fclose(stdout);
	return 0;
}
```

10

Exemplo de arquivo de saída *"s.txt"*. Procure observar atentamente como a saída é descrita e os exemplos são mostrados.

Erros comuns de saída:

- adicionar mensagens não solicitadas (ex. Dobro = 10);
- colocar linha em branco no começo do arquivo quando não solicitado (a solicitação foi de apenas escrever um número);
- colocar tabulação quando não solicitado, etc.

Introdução à Programação: Lista 2

1. Escreva um programa que lê duas matrizes 3x3 de inteiros e uma invoca uma função *void somarMatrizes(int a[3][3], int b[3][3])* que calcula e imprime o valor da soma das duas matrizes.

0 1 2 3 4 5 6 7 8 2 2 2 1 1 1 4 5 6 *e1.txt*
2 3 4 4 5 6 10 12 14 *s1.txt*

2. Os incas ficaram conhecidos pela grande civilização que reinou na região dos Andes durante vários séculos. O que pouca gente sabe é que os incas construíram pirâmides de base quadrada em que a única forma de se atingir o topo era seguir em espiral pela borda, que acabava formando uma escada em espiral. Estas pirâmides ainda se encontram escondidas na floresta amazônica e sua descoberta trará uma aplicação para este exercício.

   Neste problema você deverá fazer um programa que recebe uma matriz quadrada $A_{4 \times 4}$ de números inteiros e verifica se a matriz é inca, ou seja, se partindo do canto superior esquerdo da matriz, no sentido horário, em espiral, a posição seguinte na ordem é o inteiro consecutivo da posição anterior.

   A primeira linha do programa informa um número *n*, que representa o número de matrizes deverão ser lidas. Em seguida, haverá *n* grupos de 5 linhas. A primeira linha de cada grupo representa um nome para a matriz (apenas 2 caracteres), as 4 linhas seguintes representam os elementos da matriz.

   Exemplo: A matriz abaixo é inca

$$\begin{pmatrix} 1 & 2 & 3 & 4 \\ 12 & 13 & 14 & 5 \\ 11 & 16 & 15 & 6 \\ 10 & 9 & 8 & 7 \end{pmatrix}$$

2 M1 1 2 3 4 12 13 14 5 11 16 15 6 10 9 8 7 M2 1 5 3 4 12 0 14 5 2 16 15 6 10 0 8 1 *e2.txt*
M1: inca M2: nao inca *s2.txt*

3. Escreva um programa que lê várias matrizes quadradas (de inteiros) de tamanho 3, leia os elementos de cada uma das matrizes, calcule e imprima o seu determinante. A primeira linha contém o número de *n* matrizes. As *4xn* linhas seguintes serão dispostas da seguinte forma: uma linha em branco seguida por 3 linhas contendo os elementos de cada matriz.

2 1 3 10 -1 1 10 0 2 10 2 0 0 67 5 0 44 21 9 *e3.txt*
0 90 *s3.txt*

4. Considere uma matriz de distância entre cidades 6 x 6, como no exemplo abaixo:

	0.(Cáceres)	1.(BBugres)	2.(Cuiabá)	3.(VGrande)	4.(Tangará)	5.(PLacerda)
0.(Cáceres)		63	210	190		190
1.(BBugres)	63		160	150	95	
2.(Cuiabá)	210	160		10		
3.(VGrande)	190	150	10			
4.(Tangará)		95				80
5.(PLacerda)	190				80	

Considere também (como exemplo) um vetor de viagem indo de Cuiabá até Cáceres pela seguinte rota:

Índice	0	1	2	3	4	5
Cidade	2	3	1	4	5	0

Escreva um programa que leia a matriz, o vetor e calcule a distância percorrida durante a viagem. A entrada contém a matriz (as rotas 0 são inalcansáveis diretamente ou representam cidades de origem e destino iguais), bem como a rota seguida na viagem.

matriz 0 63 210 190 0 190 63 0 160 150 95 0 210 160 0 10 0 0 190 150 10 0 0 0 0 95 0 0 0 80 190 0 0 0 80 0 rota 2 3 1 4 5 0 *e4.txt*
525 *s4.txt*

5. A potencialização (Exponenciação) significa multiplicar um número real (base) por ele mesmo *x* vezes, onde *x* é a potência (número natural). Por exemplo, $3^3 = 3 . 3 . 3 = 3 . 9 = 27$.
   Multiplicar um número X por outro Y significa somar o número X com ele mesmo Y vezes. Por exemplo, 3 x 4 = 3 + 3 + 3 + 3 = 12

   Faça um programa que receba inicialmente um número *n* que representará a quantidade de vezes que ele fará os cálculos. Depois receberá 2 números inteiros positivos para cada uma das *n* vezes calculando a multiplicação e a potencialização dos dois(levando em consideração que o primeiro número deve ser a base).
   Obs1.: Para realizar os cálculos, escreva 2 funções:

   - *void multiplica(int x, int y, int *resultado);*
     - Calcula a multiplicação utilizando a operação da soma. Armazena o resultado da multiplicação no inteiro apontado pelo ponteiro resultado.
   - *void calculaExponenciacao(int base, int expoente, int *resultado);*
     - Utiliza a função multiplica nos seus cálculos. Armazena o resultado da exponenciação no inteiro apontado pelo ponteiro resultado.

3 3 3 4 0 5 8 *e5.txt*
27 1 390625 *s5.txt*

6. Escreva um programa que lê 5 vetores de inteiros de tamanho 5 e coloque-os em uma matriz de inteiros 5x5, ordenando-os, sendo que o primeiro vetor lido fica na primeira linha, o segundo na segunda, e assim sucessivamente. Para resolver a questão, escreva as funções abaixo descritas, nas quais L é uma constante de valor 5:
   - *void ler(int v[L]) //lê um vetor de tamanho L*
   - *void ordenar(int v[L]) //ordena um vetor de tamanho L*
   - *void popularMatriz(int m[L][L], int v[L], int i) //popula a linha i da matriz m, com os elementos de v, já ordenados.*
   - *void imprimirMatriz(int m[L][L]) //imprime os elementos da matriz m*

4 6 7 0 3 10 11 8 9 1 1 2 3 3 0 0 9 1 1 2 5 1 0 6 2 *e6.txt*
0, 3, 4, 6, 7 1, 8, 9, 10, 11 0, 1, 2, 3, 3 0, 1, 1, 2, 9 0, 1, 2, 5, 6 *s6.txt*

7. Escreva um programa que lê *n* linhas (onde *n* é o número lido na primeira linha da entrada), cada uma contendo duas *string*s. O programa deve informar se, para cada linha lida, a primeira *string* é *substring* da segunda. Exemplo, o programa deve imprimir *"cif eh substring de Recife"*, quando lidas as strings *"cif"* e *"Recife"*. Outro exemplo, o programa deve imprimir *"abc eh substring de dcbabdabc"*, quando lidas as strings *"abc"* e *"dcbabdabc"*. Cada *string* lida pode ter no máximo até 20 caracteres. Não utilizar funções de bibliotecas que fazem processamento de *strings*, como *string.h*.

cif recife abc dcbabdabc verde azul *e7.txt*
cif eh substring de recife abc eh substring de dcbabdabc verde nao eh substring de azul *s7.txt*

8. Escreva um programa que lê um número indeterminado de inteiros positivos (*unsigned int*), armazena-os em um vetor, ordene-o e calcule a média de seus elementos (com uma casa decimal), o menor e o maior dos valores. O fim da entrada pode ser verificado com o número 0 (que não deve ser armazenado no vetor). O programa deve imprimir o vetor ordenado, a média, o maior e o menor valor, respectivamente. Para armazenar os elementos no vetor, crie-o inicialmente com 10 posições – utilizando a função *malloc*. Sempre que houver a necessidade de ler elementos além da capacidade do vetor, dobre-o, utilizando a função *realloc*. Observação: pode haver valores repetidos na entrada.

3 7 9 11 6 100 56 0 *e8.txt*
3 6 7 9 11 56 100 27.4 100 3 *s8.txt*

9. Escreva um programa que lê dois números inteiros positivos, *n* e *x*, respectivamente, retorna a soma os fatoriais dos *n* próximos números primos a partir de *x*. O programa termina quando forem lidos dois números iguais a 0.

   Crie, além da *main*, pelo menos essas duas funções:

   - *void ehPrimo(unsigned int a, bool *resultado);*
     - armazena, na variável apontada por *resultado*, *true* se *a* for primo ou *false*, se não for.
   - *void fatorial(unsigned int a, double *resultado);*
     - armazena, na variável apontada por *resultado*, o valor do fatorial do número *a*.

1 4 2 7 2 10 7 1 0 0 *e9.txt*

```
120
39921840
6266937600
355693695038768

s9.txt
```

Observações:

- Para trabalhar com tipo *booleanos*, utilize a biblioteca *stdbool.h*.
- A função que calcula o número fatorial deve retornar um *double* para que possam ser calculados números fatoriais maiores, já que uma função fatorial que retornasse um inteiro não poderia retorna corretamente um fatorial maior que 2147483647 (valor da constante INT_MAX definida em *limits.h*).

10. O programa abaixo cria uma matriz de inteiros através da função *criarMatriz* utilizando várias chamadas a *malloc*. A primeira chamada aloca o espaço necessário para armazenar os endereços das primeiras posições de cada linha. As demais chamadas alocam espaço para armazenar os inteiros de cada linha (cada linha possuirá tantos inteiros quanto o número de colunas). Procure entender bem a função *criarMatriz* e, baseado no exemplo, faça um programa que cria uma matriz de tamanho lido *mxn* (de inteiros) e lê os valores desta matriz. O programa deve, também, alocar espaço para a matriz transposta da matriz lida. A matriz transposta (*nxm*) deve ser calculada e ter seus valores impressos.

```
#include <stdio.h>
#include <stdlib.h>
//A função criarMatrizcria uma matriz de inteiros dinamicamente.
int** criarMatriz(int linha, int coluna) {
   int i, j, **m = (int**) malloc(linha * sizeof (int*));
   if (m == NULL) {
      printf("Nao foi possivel alocar memoria para a matriz");
      exit(EXIT_FAILURE); //a função exit termina a execução do programa.
   }

   for (i = 0; i < linha; i++) {
      m[i] = (int*) malloc(coluna * sizeof (int));
      if (m[i] == NULL) {
         printf("Nao foi possivel alocar memoriia para a matriz");
         //Liberando o espaço já alocado
         for (j = 0; j < i; j++) {
            free(m[j]);
         }
         free(m);
         exit(EXIT_FAILURE);
      }
   }
   return m;
}

int main(void) {
   int i, j, l, c, **m;
   scanf("%d %d", &l, &c);
   m = criarMatriz(l, c);
   for (i = 0; i < l; i++)
      for (j = 0; j < c; j++)
         m[i][j] = j;

   printf("{");
   for (i = 0; i < l; i++) {
      printf("{");
      for (j = 0; j < c; j++) {
         printf("%d", m[i][j]);
         if (j != c - 1) printf(", ");
      }
      printf("}");
   }
   printf("}\n");
   for (i = 0; i < l; i++) free(m[i]);
   free(m);
   return 0;
}
```

2 3 7 2 3 8 0 4 *e10.txt*
7 8 2 0 3 4 *s10.txt*

11. Escreva um programa que gere uma tabuada de 1 até um valor $1 < n < 10$ na forma de uma matriz $n \times n$ tal que, na posição da linha i e coluna j da matriz, deve-se encontrar o valor $(i+1)*(j+1)$. Por exemplo, para $n=6$ o programa deve gerar a saída abaixo:

6 *e11.txt*
1 2 3 4 5 6 2 4 6 8 10 12 3 6 9 12 15 18 4 8 12 16 20 24 5 10 15 20 25 30 6 12 18 24 30 36 *s11.txt*

A tabuada deverá ser representada por uma matriz criada em tempo de execução, de acordo com o valor *n* lido. A memória alocada para a matriz deverá ser liberada ao final do programa. Utilize, para alocar memória, a função *calloc* e, para liberá-la, a função *realloc*.

12. É possível substituir um switch case por vetores de ponteiros para funções. Isso simplifica bastante o código fonte programa. O programa abaixo exemplifica bem isso, utilizando um caso típico candidato a *switch case*, no qual temos quatro funções e uma opção a ser lida que resultará na execução de uma dessas funções.

```
#include <stdio.h>
#include <stdlib.h>

void reopen(char * e, char * s) {
   freopen(e, "r", stdin); freopen(s, "w", stdout);
}

void close(void) {
   fclose(stdin);
   fclose(stdout);
}

void somar(int x, int y, int *r) { *r = x + y; }

void subtrair(int x, int y, int *r) { *r = x – y; }

void dividir(int x, int y, int *r) {
   if (y == 0)
      exit(EXIT_FAILURE); //Interrompe a execução do programa
   *r = x / y;
}

void multiplicar(int x, int y, int *r) { *r = x * y; }

int main(void) {
   int a, b, resultado, opcao;
   //Vetor de ponteiro para função
   void (*f[]) (int, int, int*) = {somar, subtrair, dividir, multiplicar};
   reopen("ex19.txt", "sx19.txt");

   do {
      /*
      * opcao 0 - soma
      * opcao 1 – subtração
      * opcao 2 – divisão
      * opcao 3 – multiplicação
      * opcao 4 - sair
      */
      scanf("%d", &opcao);
      if (opcao >= 0 && opcao < 4) {
         scanf("%d %d", &a, &b);
         f[opcao](a, b, &resultado);
```

```
            printf("%d\n", resultado);
        }

    } while (opcao != 4);

    close();
    return EXIT_SUCCESS;
}
```

0 4 5 1 10 3 2 7 2 3 8 5 4 *ex12.txt*
9 7 3 40 *sx12.txt*

Escreva um programa que lê um caractere *a* e, baseado nele, lê um número real (*double*) e invoca uma função de *math.h*, de acordo com a tabela abaixo. Na saída, o programa deve imprimir o resultado da invocação da função. O programa continua lendo até encontrar o caractere *'s'* – que indica o fim do processamento.

caractere	função
a	ceil
b	fabs
c	floor
d	cos
e	sin
f	tan
g	acos
h	asin
i	atan
j	exp
k	log
l	log10
m	sqrt

Observações:

✗ Para as funções *cos, sin* e *tan* o número lido é em graus e deverá ser convertido para radianos.
✗ O programa não deve utilizar *switch case* ou *if else* para invocar as funções matemáticas ou mesmo converter o ângulo em graus para radianos.
✗ Para evitar a lógica condicional citada no item anterior, utilize-se de vetores de ponteiros para funções. O índice 0 do vetor conterá um ponteiro para a função *ceil*, o índice 1 do vetor, um ponteiro para *fabs*, e assim sucessivamente.
✗ O programa deve ignorar linhas contendo caracteres diferentes dos especificados.

✗ Dica: converta os caracteres lidos para inteiro e subtraia deles o valor de *'a'* convertido para inteiro.
✗ Considere que apenas valores válidos são passados para as funções (exemplo, não serão passados ângulos inválidos para a função *tan*).
✗ Dica: sempre converta o número real lido utilizando também um vetor de ponteiros para funções.

- Esse vetor possuirá o mesmo número de índices do vetor que aponta para as funções matemáticas.
- Esse vetor possuirá em seus índices ponteiros para duas funções: *double fazNada(double n)* e *double converteGrausParaRadianos(double n)*. Os índices 3, 4 e 5 conterão ponteiros para a função converteGrausParaRadianos. Os demais índices terão ponteiros para a função *fazNada*.
- *converteGrausParaRadianos* converte o doublerecebido para radianos – essencial para as funções *cos*, *sin* e *tan*.
- *fazNada* recebe um double e retorna esse mesmo double inalterado. Essa função servirá para "converter"o valor de entrada para a demais funções.

```
a 10.65
c 56.4
e 30
x
l 1000
B
h 0.5
s
```

e12.txt

```
11.00
56.00
0.50
3.00
0.52
```

s12.txt

13. Considere a estrutura abaixo:

```
struct aritmeticaStr {
        int x, y, z;
        char op;
        bool erro;
};
typedef struct aritmeticaStr aritmetica;
```

A estrutura representa uma operação aritmética, seus operandos, seu resultado e, em algumas situações (no caso da operação de divisão, especificamente quando o denominador, z, for zero), um erro.
Por exemplo, *op* poderia ser a soma ('+'), os operandos x = 3, y = 4, e o resultado, z = 7. O flag de erro, erro = *false*. Como outro exemplo, *op* poderia ser a divisão ('/'), o operandos x = 4, y = 0, e o flag de erro, erro = true, pois o denominador de uma divisão (z) não pode ser zero (observe que, nesse caso, o resultado, z, não é relevante).
Escreva um programa que lê um número *n* e, em seguida, lê (para cada linha seguinte do arquivo de entrada) o tipo da operação (0 para soma; 1 para subtração, 2 para multiplicação e 3 para divisão) e os operandos *x* e *y* . A saída do programa deve ser o resultado das operações solicitadas na entrada, conforme mostrado no exemplo de saída, *s13.txt.*

```
6
0 3 -2
2 5 4
1 10 -
3 50 3
0 4 11
3 2 0
```

e13.txt

```
3 + -2 = 1
5 * 4 = 20
10 - -8 = 18
50 / 3 = 16
4 + 11 = 15
2 / 0 = erro
```

s13.txt

Observe o seguinte para implementar o programa:

a) Deverá ser criado um vetor de estrutura de tamanho n – para isso utilize a função malloc.

b) Cada elemento do vetor deverá armazenar a operação, os operandos, o resultado e o flag de erro.

- Exemplo: para a primeira linha de entrada do arquivo exemplo e7.txt, o primeiro elemento do vetor será um ponteiro para uma estrutura do tipo *aritmetica*, cujos valores são {3, -2, 1, '+', false}.

c) Crie quatro funções para realizar os cálculos:

```
/*
* recebe um ponteiro para a estrutura aritmética, realiza os cálculos, armazena o
* resultado em calc->z e a operação em calc->op (nesse caso, o valor de op será '+'
*e calc->erro = false).
*/
void soma(aritmetica * calc);
/*
* recebe um ponteiro para a estrutura aritmética, realiza os cálculos, armazena o
* resultado em calc->z e a operação em calc->op (nesse caso, o valor de op será '-'
* e calc->erro = false).
*/
void subtracao(aritmetica * calc);
/*
* recebe um ponteiro para a estrutura aritmética, realiza os cálculos, armazena o
* resultado em calc->z e a operação em calc->op (nesse caso, o valor de op será '*'
* e e calc->erro = false).
*/
void multiplicacao(aritmetica * calc);
/*
* recebe um ponteiro para a estrutura aritmética, realiza os cálculos, armazena o
* resultado em calc->z e a operação em calc->op (nesse caso, o valor de op será
* '/'). Caso calc->y = 0, calc->erro = true. Caso contrário, calc->erro = false.
*/
void divisao(aritmetica * calc);
```

d) Não utilize *switch case* ou *if* para invocar as funções de acordo com as operações aritméticas lidas (0, 1, 2, ou 3). Em vez disso, crie um vetor de ponteiro para função, de maneira que o primeiro elemento desse vetor (índice 0) seja um ponteiro para a função soma. O segundo (índice 1), um ponteiro para a função *subtracao*. O terceiro (índice 2), um ponteiro para a função *multiplicacao*. Finalmente, o quarto (índice 3), um ponteiro para a função *divisao*.