Lista de Exercícios

19) Argumentos

a) Escreva um programa que receba, pela linha de comando, dois inteiros positivos *n* e *s* e imprima os números produzidos por *n* invocações da função rand com semente *s*.

b) Escreva um programa que receba, pela linha de comando, os inteiros positivos *n*, *l*, *h* e *s* e imprima os números produzidos por *n* invocações da função randomInteger com argumentos *l* e *h* e semente *s*.

```
// A função randomInteger devolve um inteiro
// aleatório entre low e high inclusive,
// ou seja, no intervalo fechado low..high.
// Vamos supor que low <= high e que
// high - low <= RAND_MAX. (O código foi copiado
// da biblioteca random de Eric Roberts.)
int randomInteger (int low, int high)
{
   double d;
   d = (double) rand () / ((double) RAND_MAX + 1);
   int k = d * (high - low + 1);
   return low + k;
}
```

25) Funções

a) Construa uma função *encaixa* que dados dois inteiros positivos *a* e *b* verifica se *b* corresponde aos últimos dígitos de *a*.

Ex.:

a	***B***	**Resultado**
567890	890	=> encaixa
1243	1243	=> encaixa
2457	245	=> não encaixa
457	2457	=> não encaixa

b) Escreva uma função que recebe um inteiro positivo *m* e devolve 1 se *m* é primo, 0 em caso contrário.

c) Escreva um programa que leia um inteiro não-negativo *n* e imprima a soma dos *n* primeiros números primos.

d) Escreva um programa que leia um inteiro positivo *n* e uma seqüência de *n* inteiros não-negativos e imprime o *mdc* de todos os números da seqüência.

e) Escreva uma função que recebe como parâmetro um inteiro positivo *ano* e devolve 1 se *ano* for bissexto, 0 em caso contrário. (Um ano é bissexto se (ano % 4 == 0 && (ano % 100 != 0 || ano % 400 == 0)).)

f) Escreva uma função que lê, linha a linha, uma matriz real A_{mxn}

g) Escreva uma função que imprime uma matriz real Amxn

h) Escreva uma função que calcula a soma dos elementos da linha *i* de uma matriz real A_{mxn}.

i) Escreva uma função que calcula o produto dos elementos da coluna *j* de uma matriz real A_{mxn}.

j) Escreva uma função que recebe uma matriz de caracteres 8x8 representando um tabuleiro de xadrez e calcula o valor total das peças do jogo. Espaços vazios do tabuleiro são codificados como casas com ` '(branco) e têm valor 0 (zero). O valor das demais peças é dado de acordo com a tabela:

Peça	Valor
peão	1
cavalo	3
bispo	3
torre	5
rainha	10
rei	50

44) Bibliotecas

a) Faça um programa que teste todas as principais funções da bilbioteca math.h

Esta biblioteca contém várias constantes e funções matemáticas. Para usar a biblioteca você precisa compilar o seu programa com a opção -lm:
gcc meu_programa.c -lm (Esse “lm” é um “LM” em letras minúsculas.)

```
// Arquivo math.h.
// Interface da biblioteca math.
//////////////////////////////////////////////////////
//////////////////////////////////////////////////////
// Seção 1 -- Funções trigonométricas
//////////////////////////////////////////////////////
double sin (double);
double cos (double);
double tan (double);
//////////////////////////////////////////////////////
// Seção 2 -- Exponenciais e logaritmos
//////////////////////////////////////////////////////
// Devolve e^x, ou seja, o número e elevado à potência x.
// Uso típico: y = exp (x);
double exp (double);
// Devolve o logaritmo de x na base e. Não use com x
// negativo (ou nulo). Uso típico: y = log (x);
double log (double);
// Devolve o logaritmo de x na base 10. Não use com x
// negativo (ou nulo). Uso típico: y = log10 (x);
double log10 (double);
//////////////////////////////////////////////////////
// Seção 3 -- Raiz e potência
//////////////////////////////////////////////////////
// Devolve a raiz quadrada de x. Não use com x < 0.
// Uso típico: y = sqrt (x);
double sqrt (double);
// Devolve x^y, ou seja, x elevado à potência y. Não use
```

```
// com x = 0.0 e y < 0.0. Não use com x < 0.0 e y não
// inteiro. Caso especial: pow (0.0, 0.0) == 1.0. Que
// acontece se x^y não couber em double? Veja man pages.
// Uso típico: p = pow (x, y);
double pow (double, double);
////////////////////////////////////////////////////////
// Seção 4 -- Arredondamentos
////////////////////////////////////////////////////////
// A função devolve o maior inteiro que é menor que ou
// igual a x, isto é, o único inteiro i que satisfaz
// i <= x < i+1. Uso típico: i = floor (x);
double floor (double);
// A função devolve o menor inteiro que é maior que ou
// igual a x, isto é, o único inteiro j que satisfaz
// j-1 < x <= j. Uso típico: j = ceil (x);
double ceil (double);
```

b) Faça um programa que teste todas as principais funções da bilbioteca ctype.h

A maioria das funções desta biblioteca serve para *classificar caracteres ASCII*, ou seja, para dizer se um dado byte representa uma letra, ou um dígito, ou um branco, etc. em código ASCII. O argumento de cada função não é um char mas sim um inteiro positivo ou um EOF. Usualmente, o argumento pertence ao intervalo –1..127.

```
// Arquivo ctype.h.
// Interface da biblioteca ctype.
////////////////////////////////////////////////////////
////////////////////////////////////////////////////////
// Seção 1: Funções booleanas
////////////////////////////////////////////////////////
// Todas as funções desta seção devolvem 0 para dizer "não"
// ou um inteiro diferente de 0 para dizer "sim".
// A função isspace decide se o argumento representa um
// branco (white-space), ou seja, um de \  \t \n \v \f \r.
// Uso típico: if (isspace (c)) ...
int isspace (int c);
// A função isdigit decide se o argumento é um dígito
// decimal (0 1 2 3 4 5 6 7 8 9).  Uso típico:
// if (isdigit (c)) ....
int isdigit (int c);
// A função islower decide se o argumento representa uma
// letra minúscula (a b c d e f g h i j k l m n o p q r s t
// u v w x y z). (É claro que letras com diacríticos, como
// á e ç por exemplo, não estão no conjunto.)  Uso típico:
// if (islower (c)) ....
int islower (int c);
// A função isupper decide se o argumento representa uma
// letra maiúscula (A B C D E F G H I J K L M N O P Q R S T
// U V W X Y Z). (É claro que letras com diacríticos não
// estão no conjunto.)  Uso típico:
// if (isupper (c)) ....
int isupper (int c);
// A função isalpha decide se o argumento representa uma
// letra (maiúscula ou minúscula).  Uso típico:
// if (isalpha (c)) ....
int isalpha (int c);
// A função isalnum decide se o argumento representa um
// caractere alfanumérico (letra ou dígito decimal).  Uso
// típico: if (isalnum (c)) ....
```

```
int isalnum (int c);
// A função ispunct decide se o argumento representa um
// caractere de pontuação, ou seja, um de ! " # $ % & ' ( )
// * + , - . / : ; < = > ? @ [ \ ] ^ _ ` { | } ~.  Uso
// típico: if (ispunct (c)) ....
int ispunct (int c);
//////////////////////////////////////////////////////////
// Seção 1: Funções de conversão
//////////////////////////////////////////////////////////
// A função toupper recebe uma letra c e devolve a
// correspondente letra maiúscula. Uso típico:
// C = toupper (c);
int toupper (int c);
// A função tolower recebe uma letra C e devolve a
// correspondente letra minúscula. Uso típico:
// c = tolower (C);
int tolower (int C);
```

d) Faça um programa que teste todas as principais funções da bilbioteca string.h

Esta biblioteca contém funções que manipulam strings. (Não confunda com a biblioteca obsoleta strings, que se desvia um pouco do padrão.) Em geral, as strings representam cadeias de caracteres em código ASCII, mas podem também representar cadeias em código UT-8.

```
// Arquivo string.h.
// Interface da biblioteca string.
//////////////////////////////////////////////////////////
#include <stddef.h>
// //////////////////////////////////////////////////////
// Manipulação de strings
//////////////////////////////////////////////////////////
// A função devolve o comprimento da string x. Em outras
// palavras, devolve o número de bytes de x (sem contar o
// \0 final).  O código da função tem o mesmo efeito que
//       for (i = 0; x[i] != 0; ++i) ;
//       return i;
// que por sua vez equivale a
//       y = x;
//       while (*y++) ;
//       return y-x-1;
// Uso típico: k = strlen (x);
unsigned int strlen (char *x);
// Copia a string x (inclusive o byte \0 final) no espaço
// alocado para a string y. Cabe ao usuário garantir que o
// espaço alocado a y tem pelo menos strlen(x) + 1 bytes.
// A função devolve y. Exemplo:
//       char y[4];
//       strcpy (y, "ABC");
// O código da função equivale a
//       for (i = 0; (y[i] = x[i]) != 0; ++i) ;
// que por sua vez equivale a
//       while (*y++ = *x++) ;
// Uso típico: strcpy (y, x);
char *strcpy (char *y, char *x);
// Se strlen(x) < n então copia a string x (inclusive o \0
// final) para o espaço y. Se strlen(x) >= n então copia
// para y os n primeiros bytes de x e não acrescenta \0 ao
// final de y. Cabe ao usuário garantir que o espaço
// alocado a y tem pelo menos strlen(x) + 1 ou pelo menos
```

```
// n bytes. A função devolve y. Exemplo:
//       char y[5];
//       strncpy (y, "ABCDE", 4);
// O código da função strncpy equivale a
//       for (i = 0; i < n && x[i] != '\0'; i++)
//           y[i] = x[i];
//       for  (; i < n; i++)
//           y[i] = '\0';
// Uso típico: strncpy (y, x, n);
char *strncpy (char *y, char *x, size_t n);
// Concatena as strings x e y, isto é, acrescenta y ao
// final de x. Devolve o endereço da string resultante,
// ou seja, devolve x. Cabe ao usuário garantir que o
// espaço alocado a x é suficiente para comportar strlen(y)
// bytes adicionais (após o \0 que marca o fim de x).
// Exemplo:
//       char x[7];
//       strcpy (x, "ABC");
//       strcat (x, "DEF");
// O código da função equivale a
//       strcpy (x + strlen (x), y);
// Uso típico: strcat (x, y);
char *strcat (char *, char *);
// Compara lexicograficamente as strings x e y. Devolve um
// número estritamente negativo se x vem antes de y,
// devolve 0 se x é igual a y e devolve um número
// estritamente positivo se x vem depois de y. O código da
// função equivale a
//       for (i = 0; x[i] == y[i]; ++i)
//          if (x[i] == 0) return 0;
//       return x[i] - y[i];
// que por sua vez equivale a
//       while (*x++ == *y++)
//          if (*(x-1) == 0) return 0;
//       return *(x-1) - *(y-1);
// Uso típico: if (strcmp (x, y) == 0) ... ;
int strcmp (char *x, char *y);
// Esta função é análoga à strcmp mas não usa a ordenação
// dos caracteres imposta pela ordem crescente dos números
// Unicode. Em vez disso, a ordenação dos caracteres
// depende do valor da variável de ambiente LC_COLLATE.
// Uso típico: if (strcoll (x, y) == 0) ... ;
int strcoll (char *x, char *y);
// Extrai tokens da string s. Um token é qualquer segmento
// maximal de s sem delimitadores. Um delimitador é
// qualquer byte da string d.
//    A string s pode ser vista como uma sequência de zero
// ou mais delimitadores, seguida de um ou mais não-
// delimitadores, seguida de um ou mais delimitadores, ...,
// seguida de um ou mais não-delimitadores, e finalmente
// seguida de zero ou mais delimitadores.
//    A função strtok transforma cada token em uma string
// (colocando um \0 na posição seguinte ao último byte do
// token) e devolve o (endereço do) token. Uma chamada a
// strtok com s != NULL devolve o primeiro token de s.
// Chamadas subsequentes com s == NULL devolvem o segundo,
// terceiro, etc. tokens.  O segundo argumento, d, pode ser
// diferente em cada chamada a strtok. Depois que todos os
// tokens forem encontrados, strtok devolve NULL.
//    Segue minha versão caseira de strtok, restrita aos
```

```
// delimitadores ' ' e ',':
//    char *mystrtok (char *s) {
//       static char *restart;
//       char *token;
//       if (s == NULL) s = restart;
//       while (*s == ' ' || *s == ',') s++;
//       if (*s == '\0') return NULL;
//       token = s;
//       while (*s != ' ' && *s != ',' && *s != '\0') s++;
//       restart = s;
//       if (*s != '\0') {
//          *s = '\0';
//          restart++;
//       }
//       return token;
//    }
// Usos típicos: strtok (s, d); strtok (NULL, d);
char *strtok (char *s, const char *d);
```