PONTIFICIA UNIVERSIDADE CATÓLICA DE MINAS GERAIS
Instituto de Ciências Exatas e Informática
Departamento de Ciência da Computação

Disciplina	Curso	Turno	Período
Algorítmos e Estruturas de Dados I	Ciência da Computação	Manhã	1°

Professor
Felipe Cunha (felipe@pucminas.br)

Lista de Exercícios 05 - Correção

1. Fazer um método recursivo que recebe um número inteiro e retorna o seu fatorial.

```c
#include <stdio.h>

// Função recursiva que calcula o fatorial de um número n
int fatorial(int n) {
    if (n < 0) {
        printf("Erro: fatorial nao definido para numeros negativos.\n");
        return 0;
    }
    if (n == 0 || n == 1) {
        return 1; // Caso base: 0! = 1! = 1
    }
    return n * fatorial(n - 1); // Passo recursivo
}

int main() {
    int numero;

    printf("Digite um numero inteiro: ");
    scanf("%d", &numero);

    int resultado = fatorial(numero);

    if (numero >= 0) {
        printf("O fatorial de %d é %d\n", numero, resultado);
    }

    return 0;
}
```

2. Fazer um método recursivo que recebe um número inteiro n e retorna o n-ésimo termo da equação de recorrência abaixo:

   $T(1) = 2$

   $T(2) = 3$

   $T(n) = 5 * n + T(n-1)^n$

```c
#include <stdio.h>
#include <math.h> // Para a função pow

// Função recursiva que retorna o n-ésimo termo
double T(int n) {
    if (n == 1) return 2;        // Caso base 1
    if (n == 2) return 3;        // Caso base 2
    double prev = T(n - 1);
    return 5 * n + pow(prev, n); // Passo recursivo: T(n) = 5*n + T(n-1)^n
}

int main() {
    int n;

    printf("Digite o valor de n: ");
    scanf("%d", &n);

    if (n < 1) {
        printf("Erro: n deve ser maior ou igual a 1.\n");
        return 1;
    }

    double resultado = T(n);
    printf("T(%d) = %.0lf\n", n, resultado);

    return 0;
}
```

3. Fazer um método recursivo que recebe um número inteiro e positivo n e calcula o somatório abaixo.

   $n + (n-1) + ... + 1 + 0$

```c
#include <stdio.h>

// Função recursiva que calcula o somatório de n até 0
int somatorio(int n) {
    if (n <= 0) {
        return 0; // Caso base: somatório de 0 é 0
    }
    return n + somatorio(n - 1); // Passo recursivo
}

int main() {
    int n;

    printf("Digite um número inteiro positivo: ");
    scanf("%d", &n);

    if (n < 0) {
        printf("Erro: o número deve ser positivo.\n");
        return 1;
    }

    int resultado = somatorio(n);
    printf("O somatório de %d até 0 é %d\n", n, resultado);

    return 0;
}
```

4. Fazer um método recursivo que imprima de um número natural em base binária

```c
#include <stdio.h>

// Função recursiva que imprime o número n em binário
void imprimirBinario(int n) {
    if (n > 1) {
        imprimirBinario(n / 2); // Chamada recursiva com n/2
    }
    printf("%d", n % 2); // Imprime o bit atual
}

int main() {
    int n;

    printf("Digite um número natural: ");
    scanf("%d", &n);

    if (n < 0) {
        printf("Erro: apenas números naturais (>= 0) são
            permitidos.\n");
        return 1;
    }

    if (n == 0) {
        printf("0"); // Caso especial para 0
    } else {
        imprimirBinario(n);
    }

    printf("\n");
    return 0;
}
```

5. Fazer um método recursivo que multiplique dois números naturais, através de somas sucessivas

```c
#include <stdio.h>

// Função recursiva que multiplica a por b usando somas sucessivas
int multiplicar(int a, int b) {
    if (b == 0) return 0;           // Caso base: multiplicar por 0
    return a + multiplicar(a, b - 1); // Passo recursivo: soma 'a' b
       vezes
}

int main() {
    int x, y;

    printf("Digite dois números naturais: ");
    scanf("%d %d", &x, &y);

    if (x < 0 || y < 0) {
        printf("Erro: apenas números naturais são permitidos.\n");
        return 1;
    }

    int resultado = multiplicar(x, y);
    printf("%d * %d = %d\n", x, y, resultado);

    return 0;
}
```

6. Fazer um método recursivo que calcule o MDC (máximo divisor comum) de dois inteiros positivos m e n

```c
#include <stdio.h>

// Função recursiva que calcula o MDC usando o Algoritmo de Euclides
int mdc(int m, int n) {
    if (n == 0) return m;         // Caso base
    return mdc(n, m % n);         // Passo recursivo
}

int main() {
    int m, n;

    printf("Digite dois inteiros positivos: ");
    scanf("%d %d", &m, &n);

    if (m <= 0 || n <= 0) {
        printf("Erro: apenas inteiros positivos são permitidos.\n");
        return 1;
    }

    int resultado = mdc(m, n);
    printf("O MDC de %d e %d é %d\n", m, n, resultado);

    return 0;
}
```

7. Fazer um método recursivo que conte os dígitos de um determinado número.

```c
#include <stdio.h>

// Função recursiva que conta os dígitos de um número
int contar_digitos(int n) {
    if (n < 10) {
        return 1; // Caso base: número com 1 dígito
    }
    return 1 + contar_digitos(n / 10); // Passo recursivo
}

int main() {
    int numero;

    printf("Digite um número inteiro positivo: ");
    scanf("%d", &numero);

    if (numero < 0) {
        printf("Erro: apenas números positivos são permitidos.\n");
        return 1;
    }

    int qtd = contar_digitos(numero);
    printf("O número %d possui %d dígito(s).\n", numero, qtd);

    return 0;
}
```

8. Fazer um método recursivo que determine se um número é ou não primo.

```c
#include <stdio.h>
#include <stdbool.h>

// Função auxiliar recursiva para verificar divisores
bool ehPrimoAux(int n, int i) {
    if (i == 1) return true;        // Caso base: nenhum divisor encontrado
    if (n % i == 0) return false;   // Encontrou um divisor ? não primo
    return ehPrimoAux(n, i - 1);    // Chamada recursiva
}

// Função principal que verifica se n é primo
bool ehPrimo(int n) {
    if (n < 2) return false;        // Números menores que 2 não são primos
    return ehPrimoAux(n, n / 2);    // Começa a verificar do n/2 para baixo
}

int main() {
    int numero;

    printf("Digite um número inteiro: ");
    scanf("%d", &numero);

    if (ehPrimo(numero)) {
        printf("O número %d é primo.\n", numero);
    } else {
        printf("O número %d não é primo.\n", numero);
    }

    return 0;
}
```