UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL
ITP + PTP: Recursividade e Modularização

Lista de Exercícios
Unidade II- 30.05.2017

Instruções Gerais

1. Esta é uma lista de exercícios para você melhorar suas habilidades em programação e, por isso, deve ser encarada com a mesma seriedade de uma avaliação;
2. Leia atentamente o enunciado de cada questão antes de iniciar a sua resposta;
3. Utilizar papel e caneta, apesar de não ser obrigatório, pode ser de grande ajuda para planejar o algoritmo que solucione os problemas. Essa prática costuma poupar tempo no momento da implementação de suas respostas.
4. Embora, nesta atividade, não sejam aplicado decréscimos de nota, lembre-se sempre e procure aplicar os mesmos padrões críticos das avaliações, evitando cometer as seguintes faltas:
   - Legibilidade comprometida (falta de indentação, etc)
   - Programa compila com mensagens de aviso (warnings)
   - Programa apresenta erros de compilação, não executa ou apresenta saída incorreta
   - Plagiarismo (no caso de cópia de colega, ambos serão penalizados)

Questão 1. Implemente uma função recursiva como solução para efetuar o cálculo da seguinte sequência, dado um valor inteiro N passado por parâmetro:

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... + \frac{1}{2^N} \tag{1}$$

Questão 2. Escreva uma função recursiva que permita somar os elementos de um vetor de inteiros.

Questão 3. Escreva uma função recursiva que determine quantas vezes um dígito K ocorre em um número natural N. Por exemplo, o dígito 2 ocorre 3 vezes em 762021192.

Questão 4. O fatorial quádruplo de um número N é dado por

$$\frac{(2n!)}{n!} \tag{2}$$

Implemente uma função recursiva que receba um número inteiro positivo N e retorne o fatorial quádruplo desse número.

Questão 5. Os números de Pell são definidos pela seguinte recursão

$$p(n) = \begin{cases} 0 & \text{se } n = 0 \\ 1 & \text{se } n = 1 \\ 2p(n-1) + p(n-2) & \text{se } n > 2 \end{cases} \quad (3)$$

Alguns números desta sequência são: 0, 1, 2, 5, 12, 29, 70, 169, 408, 985... Faça uma função recursiva que receba um número N e retorne o N-ésimo número de Pell.

Questão 6. Crie um programa em C, que contenha uma função recursiva para encontrar o menor elemento em um vetor. A leitura dos elementos do vetor e impressão do menor elemento devem ser feitas no programa principal.

Questão 7. Dado um número **N** na base decimal, escreva uma função recursiva em C que converte este número para binário.

Questão 8. Escreva uma função recursiva que calcule a soma dos dígitos de um número inteiro. Por exemplo, se a entrada for 123, a saída deverá ser 1+2+3 = 6.

Questão 9. Roberto é um aluno de ITP+PTP e acaba de aprender sobre recursividade. Porém Roberto ainda tem muitas dúvidas sobre como implementar suas funções de maneira recursiva. Atualmente Roberto está travado em sua lista de exercícios com um problema que pede para implementar um programa que lê um inteiro X e retorna o maior número inteiro primo anterior ao valor do fatorial de X. Por exemplo, se X = 5, temos que fatorial de X = 120, logo o maior número inteiro primo anterior a X é 113. Seguindo a sugestão do professor, Roberto modularizou o programa em um conjunto de funções fatorial(), primo(), primalidade() e menorPrimoAnterior(). Roberto sabe que algumas destas funções devem ser implementadas de forma recursiva, mas não sabe bem como. Ele já iniciou o seu código, baseado na função principal main() que o professor passou em aula para testar a implementação das funções. Mostre que você entende de recursividade e ajude Roberto a completar o seu código de maneira correta, respeitando os comentários feitos por Roberto.

```c
#include <stdio.h>

int fatorial(const int n) {
    /* Calcula recursivamente o fatorial de n */
    /* Insira o codigo aqui */
}

int primalidade(const int n, int i) {
    /* Testa recursivamente os divisores impares de 3 ate a raiz
       quadrada do N */
    /* Insira o codigo aqui */
}

int primo(const int n) {
    /* Testa se e divisivel por 2 executa a funcao primalidade() para
       testar os divisores impares de 3 ate a raiz quadrada do N */
```

```
15      if (n%2==0)
16          return 0;
17    return primalidade(n,3);
18  }
19
20  int maior_primo_anterior (int valor) {
21      /* Busca recursiva pelo maior valor primo anterior a valor */
22      /* Insira o codigo aqui */
23  }
24
25  int main(int argc, char const *argv[])
26  {
27      int x = 5;
28      // Imprime: X =5 (5! = 120) => 113
29      printf("X =%d (%d! = %d) => %d\n",
30      x, x, fatorial(x), maior_primo_anterior(fatorial(x)));
31      x = 3;
32      // Imprime: X =3 (3! = 6) => 5
33      printf("X =%d (%d! = %d) => %d\n",
34      x, x, fatorial(x), maior_primo_anterior(fatorial(x)));
35      x = 9;
36      // Imprime: X =9 (9! = 362880) => 362867
37      printf("X =%d (%d! = %d) => %d\n",
38      x, x, fatorial(x), maior_primo_anterior(fatorial(x)));
39      return 0;
40  }
```

Questão 10. Embora a questão anterior apresente uma modularização interna adequada, ela ainda carece de uma modularização externa. Diante disso, modularize completamente a sua solução da questão anterior, incluindo o uso de **Makefile** para a compilação automatizada de todos os elementos.

Questão 11. (DESAFIO) Mostre que você já domina a modularização de programas em linguagem C e organize as questões desta lista em pastas (ex: ./questao01,./questao02, etc) e arquivos (.h/.c - quando necessários). A seguir, crie um Makefile (ou um conjunto) que permita compilar e gerar os binários/executáveis de todas as questões, de uma só vez.