Aula Prática 4
Comandos de Repetição

Instruções para Submissão

Na aula prática de hoje, você terá que elaborar programas para resolver problemas diversos, conforme descrito abaixo. Cada uma das soluções deverá ser implementada em seu próprio arquivo com extensão `.c`. Por exemplo, a solução para o problema 1 deverá ser implementada em um arquivo chamado `problema1.c`, a solução para o problema 2 deverá ser implementada no arquivo `problema2.c` e assim por diante. Para os problemas que solicitarem a criação de um módulo, você deverá fornecer dois arquivos: um arquivo contendo a declaração das funções (arquivo com extensão `.h`), e um arquivo contendo a definição das funções (com extensão `.c`). Finalmente, submeta cada um dos arquivos pelo Moodle.

Problema 1 [problema1.h e problema1.c]

Escreva uma função chamada `populacao` que recebe como **parâmetros** o número de habitantes (tipo `unsigned long`) e a taxa anual de crescimento (em porcento) (tipo `double`) da população de uma cidade A, o número de habitantes (tipo `unsigned long`) e a taxa anual de crescimento (em porcento) (tipo `double`) da população de uma cidade B, e **retorna** o número de anos (tipo `int`) necessários para que a população da cidade A ultrapasse ou iguale a população da cidade B, mantidas as taxas de crescimento. Pode assumir que nos argumentos passados para a função, o número de habitantes da cidade A será menor que o da cidade B, e que a taxa anual de crescimento da população da cidade A será maior que a de B.

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema1.h`), e um arquivo contendo a definição das funções (arquivo `problema1.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```c
#include "problema1.h"
#include <stdio.h>

int main() {
    int anos = 0;
    unsigned long pop_a, pop_b;
    double taxa_a, taxa_b;

    printf("Digite: p_a, t_a, p_b, t_b\n");
    scanf("%lu, %lf, %lu, %lf", &pop_a, &taxa_a, &pop_b, &taxa_b);

    anos = populacao(pop_a, taxa_a, pop_b, taxa_b);
    printf("%d\n", anos);

    return 0;
}
```

Exemplo de execução do programa:

Digite: p_a, t_a, p_b, t_b
80000, 3, 200000, 1.5
63

Problema 2 [problema2.h e problema2.c]

Escreva uma função chamada `quant_sqrt_int` que recebe como **parâmetros** dois números (x e y) (tipos *int*). A função **retorna** a quantidade de números (tipo *int*) que possuem raiz quadrada inteira entre eles (inclusive x e y). Por exemplo, se os valores passados como argumento para a função for 1 e 20, a função deve retornar 4, já que 1, 4, 9 e 16 são os números que possuem raiz quadrada inteira no intervalo [1, 20].

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema2.h`), e um arquivo contendo a definição das funções (arquivo `problema2.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções

`scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```c
#include "problema2.h"
#include <stdio.h>

int main() {
    int x, y;
    int quant = -1;

    printf("Digite: x, y\n");
    scanf("%d, %d", &x, &y);

    quant = quant_sqrt_int(x, y);
    printf("%d\n", quant);

    return 0;
}
```

Exemplo de execução do programa:

Digite: x, y
1, 20
4

Problema 3 [problema3.h e problema3.c]

Escreva uma função chamada `fibonacci` que recebe como **parâmetro** um número n (tipo `int`) e a função **retorna** o n-ésimo termo da série de Fibonacci (tipo `int`). A série de Fibonacci é formada pela sequência (1, 1, 2, 3, 5, 8, 13, 21, 34, 55,...). Ela pode ser definida como:

$$a_n = \begin{cases} 1, & \text{se } n = 1 \text{ ou } n = 2 \\ a_{n-1} + a_{n-2}, & \text{se } n \geq 2 \end{cases}$$

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema3.h`), e um arquivo contendo a definição das funções (arquivo `problema3.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```c
#include "problema3.h"
#include <stdio.h>

int main() {
    int n, num = 0;;

    printf("Digite: n\n");
    scanf("%d", &n);

    num = fibonacci(n);
    printf("%d\n", num);

    return 0;
}
```

Exemplo de execução do programa:

Digite: n
7
13

Problema 4 [problema4.h e problema4.c]

Escreva uma função chamada `fatorial` que recebe como **parâmetro** um número n (tipo `int`) e a função **retorna** o valor do fatorial de n, ou seja, n! (tipo `unsigned long long`). Por exemplo, o fatorial de 6 é $1 \times 2 \times 3 \times 4 \times 5 \times 6 = 720$.

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema4.h`), e um arquivo contendo a definição das funções (arquivo `problema4.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```c
#include "problema4.h"
#include <stdio.h>

int main() {
    int n = 0;
    unsigned long long fat = 0;

    printf("Digite: n\n");
```

```
    scanf("%d", &n);

    fat = fatorial(n);
    printf("%llu\n", fat);

    return 0;
}
```

Exemplo de execução do programa:

Digite: n
6
720

Problema 5 [problema5.h e problema5.c]

Escreva uma função chamada `soma_divisores` que recebe como **parâmetro** um número n positivo (tipo `unsigned int`) e **retorna** a soma de todos os divisores de n (inclusive n) (tipo `unsigned int`). Ou seja, considere o intervalo $[1, n]$.

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema5.h`), e um arquivo contendo a definição das funções (arquivo `problema5.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```
#include "problema5.h"
#include <stdio.h>

int main() {
    unsigned int n;
    unsigned int soma = 0;

    printf("Digite: n\n");
    scanf("%u", &n);

    soma = soma_divisores(n);
    printf("%u\n", soma);

    return 0;
}
```

Exemplo de execução do programa:

Digite: n
4
7

Problema 6 [problema6.h e problema6.c]

Escreva uma função chamada `primo` que recebe como **parâmetro** um número n positivo (tipo `unsigned int`) e **retorna** 1, caso n seja um número primo, caso contrário, a função deve retornar 0 (tipo `int`). Um número primo possui apenas dois divisores, 1 e ele mesmo. Lembrando que 1 não é um número primo.

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema6.h`), e um arquivo contendo a definição das funções (arquivo `problema6.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```c
#include "problema6.h"
#include <stdio.h>

int main() {
    unsigned int n;
    int res = -1;

    printf("Digite: n\n");
    scanf("%u", &n);

    res = primo(n);
    printf("%d\n", res);

    return 0;
}
```

Exemplo de execução do programa:

Digite: n
11
1

Problema 7 [problema7.h e problema7.c]

Escreva uma função chamada `mdc` que recebe como **parâmetros** dois números (a e b) (tipos `int`) e **retorna** (tipo `int`) o máximo divisor comum dos dois números.

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema7.h`), e um arquivo contendo a definição das funções (arquivo `problema7.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```c
#include "problema7.h"
#include <stdio.h>

int main() {
    int a, b, res = 0;

    printf("Digite: a, b\n");
    scanf("%d, %d", &a, &b);

    res = mdc(a, b);
    printf("%d\n", res);

    return 0;
}
```

Exemplo de execução do programa:

Digite: a, b
12, 8
4

Problema 8 [problema8.h e problema8.c]

Escreva uma função chamada `mmc` que recebe como **parâmetros** dois números (a e b) (tipos `int`) e **retorna** (tipo `int`) o mínimo múltiplo comum (MMC) dos dois números. Por exemplo, você provavelmente já viu o método mostrado mais adiante para calcular o MMC.

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

360,	420	2
180,	210	2
90,	105	2
45,	105	3
15,	35	3
5,	35	5
1,	7	5
1,	7	7
1,	1	**MMC** = 2 * 2 * 2 * 3 * 3 * 5 * 7 = **2520**

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema8.h`), e um arquivo contendo a definição das funções (arquivo `problema8.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```c
#include "problema8.h"
#include <stdio.h>

int main() {
    int a, b, res = 0;

    printf("Digite: a, b\n");
    scanf("%d, %d", &a, &b);

    res = mmc(a, b);
    printf("%d\n", res);

    return 0;
}
```

Exemplo de execução do programa:

Digite: a, b
4, 6
12

Problema 9 [problema9.h e problema9.c]

Escreva uma função chamada `soma_digitos_pares` que recebe como **parâmetro** um número positivo n (tipo `unsigned int`) e **retorna** a soma dos dígitos pares do número n (tipo `int`).

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno,

nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema9.h`), e um arquivo contendo a definição das funções (arquivo `problema9.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```c
#include "problema9.h"
#include <stdio.h>

int main() {
    unsigned int n;
    int soma = -1;

    printf("Digite: n\n");
    scanf("%u", &n);

    soma = soma_digitos_pares(n);
    printf("%d\n", soma);

    return 0;
}
```

Exemplo de execução do programa:

Digite: n
124
6

Problema 10 [problema10.h e problema10.c]

Escreva uma função chamada `inverte` que recebe como **parâmetro** um número positivo n (tipo `unsigned int`) e **retorna** o número n com os dígitos invertidos (tipo `unsigned int`). Por exemplo, se $n = 1234$, a função deve retornar o valor 4321.

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema10.h`), e um arquivo contendo a definição das funções (arquivo `problema10.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```
#include "problema10.h"
#include <stdio.h>

int main() {
    unsigned int n;
    unsigned int i = 0;

    printf("Digite: n\n");
    scanf("%u", &n);

    i = inverte(n);
    printf("%u\n", i);

    return 0;
}
```

Exemplo de execução do programa:

Digite: n
1234
4321

Problema 11 [problema11.h e problema11.c]

A série harmônica $S = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n} + \ldots$ é divergente. Isso significa que dado qualquer número inteiro k existe n_0 tal que $S = 1 + \frac{1}{2} + \frac{1}{3} + \ldots + \frac{1}{n_0} > k$.

Escreva uma função chamada `serie_divergente` que recebe como **parâmetro** um número k (tipo `int`) e **retorna** (tipo `int`) o menor inteiro n_0 tal que $S > k$. Por exemplo, se $k = 2$, a função deve retornar $n_0 = 4$, pois $1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} = 2.083...$ e $1 + \frac{1}{2} + \frac{1}{3} = 1.833...$

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema11.h`), e um arquivo contendo a definição das funções (arquivo `problema11.c`)

Observação 3: Você não precisa realizar a entrada e saída de dados (não precisa usar as funções `scanf()` e `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```
#include "problema11.h"
#include <stdio.h>
```

```c
int main() {
    int k;
    int i = 0;

    printf("Digite: k\n");
    scanf("%d", &k);

    i = serie_divergente(k);
    printf("%d\n", i);

    return 0;
}
```

Exemplo de execução do programa:

Digite: k
3
11

Problema 12 [problema12.h e problema12.c]

Escreva um procedimento chamado `decompoe` que recebe como **parâmetro** um número n (tipo `int`) e **imprime** na tela a decomposição de n em seus fatores primos. Por exemplo, se o procedimento receber 60 como entrada, ele deve imprimir **60 = 2 * 2 * 3 * 5**.

Observação 1: O cabeçalho da função deve ser exatamente como especificado acima (tipo de retorno, nome da função, tipos e quantidade de parâmetros).

Observação 2: Você terá que implementar um módulo com a função descrita acima. Logo, você deverá submeter dois arquivos: um arquivo contendo a declaração das funções (arquivo `problema12.h`), e um arquivo contendo a definição das funções (arquivo `problema12.c`)

Observação 3: Você não precisa realizar a entrada de dados (não precisa usar a função `scanf()`). No entanto, a mensagem exibida para o usuário deverá ser exatamente conforme especificado (mensagem exibida com a função `printf()`).

O programa que irá testar o seu módulo no Moodle é apresentado abaixo:

```c
#include "problema12.h"
#include <stdio.h>

int main() {
    int n;

    scanf("%d", &n);

    decompoe(n);
```

```
    return 0;
}
```

Exemplo 1 de execução do programa:

60
60 = 2 * 2 * 3 * 5

Exemplo 2 de execução do programa:

90
90 = 2 * 3 * 3 * 5