Lista 9 (EXERCÍCIOS)

1) Utilize conversão de tipos e conserte o código abaixo para que ele resulte na saída esperada:

```
int x = 9, y = 10;

float z = x/y;

Printf(“%f\n”, z);
```

2) O código abaixo não funciona como o esperado. Identifique e corrija o erro. Porque isso ocorre?

```
int i = -10;

unsigned int stop_val = 0;


while(i <= stop_val)

{

   printf("%d\n", i);

   i = i + 1;

}
```

3) Altere o código abaixo para que seja possível modificar o valor em memória da variável constante. Altere apenas a região entre os dois printfs, o restante do código deve permanecer intocado.

```
const int a = 10;

printf("%d\n", a);

/* INSIRA CÓDIGO AQUI */

printf("%d\n", a);
```

4) Pesquise sobre a cifra de César. Crie arquivos de cabeçalho e código (cifra_cesar.h e cifra_cesar.c) com as implementações. Defina duas funções com os protótipos:

```
char * encripta(const char *entrada, unsigned int tamanho, unsigned int chave);

char * decripta(const char *entrada, unsigned int tamanho, unsigned int chave);
```

5) Escreva um programa que leia um arquivo texto definido pelo usuário e encripte o conteúdo utilizando a cifra de César. O conteúdo deverá ser salvo em um arquivo de mesmo nome porém adicionando “-encriptado” antes da extensão. Exemplo: um arquivo “entrada.txt” deverá ser salvo como “entrada-encriptado.txt”.

6) Modifique o programa anterior para ler um arquivo texto definido pelo usuário e decriptar o conteúdo utilizando a cifra de César. O conteúdo deverá ser salvo em um arquivo de mesmo nome porém adicionando “-decriptado” antes da extensão. Exemplo: um arquivo “entrada.txt” deverá ser salvo como “entrada-decriptado.txt”.

7) Pesquise sobre a cifra de Vigenère. Crie arquivos de cabeçalho e código (cifra_vigenere.h e cifra_vigenere.c) com as implementações. Defina duas funções com os protótipos:

char * encripta(const char *entrada, unsigned int tamanho, const char * chave);

char * decripta(const char *entrada, unsigned int tamanho, const char * chave);

8) Modifique o programa dos exercícios 5 e 6 para funcionar com o exercício 7.

Observação: As modificações serão mínimas caso os princípios de programação modular sejam bem aplicados.

9) Escreva uma função que recebe um vetor e retorna outro vetor com o mesmo conteúdo ordenado. O vetor retornado deverá ser alocado dinamicamente.

10) Escreva uma função que recebe dois vetores, X e Y, e retorna um vetor alocado dinamicamente com os valores repetidos em X e Y.

11) Escreva um programa que demonstra todos os erros descritos na aula de memória dinâmica (vazamento de memória, ponteiro nulo, etc).

12) Embora seja uma abordagem comum, utilizar vetores de tamanho fixo para armazenar strings não é eficiente em relação ao uso de memória. Considerando que a entrada máxima de caracteres é de 256 caracteres, faça um programa que leia strings e as armazene em um vetor de strings, utilizando apenas alocações dinamicas. Exemplo: para as entradas “alfa” e “omega”, ocorrerão duas alocações, uma para 5 bytes, outra para 6 bytes e uma alocação de 2*sizeof(char*) para o vetor, tal que vetor[0] = “alfa” e vetor[1]=”beta”.

Dica: o vetor de strings terá tipo char**.

13.1) Modifique o programa anterior para ler e salvar dados a partir de um arquivo.

13.2) Modifique o programa anterior para adicionar, remover ou mover entradas de acordo com seus índices.

13.3) Modifique o programa anterior para incluir uma opção de ordenar as entradas alfabeticamente.

14) Escreva uma função recursiva que calcula a soma de números entre 1 e n com o protótipo: int soma(int n)

15) Escreva uma função recursiva que encontra e retorna o menor elemento em um vetor, onde o vetor e o número de elementos são parâmetros: int minimo(int *a, int n)

16) Escreva uma função recursiva que calcula o fatorial de um número.

17) Escreva uma função recursiva que calcula a série de Fibonacci.

18) Faça um programa que utilize uma variável global. Um arquivo de código deverá ter uma função incrementa, que aumenta a variável global. Outro arquivo de código deverá ter uma função acessa, que imprime o valor da variável global.

19) Modifique os exercícios 17 e 18 da Lista 8 para considerar um mapa de tamanho variável.

20) Crie uma biblioteca para alocação dinâmica de vetores. Para atingir este objetivo, crie arquivos de cabeçalho e código (vetor_dinamico.h e vetor_dinamico.c). Como sugestão, inclua funções para:

- criação do vetor dinâmico (alocação);
- destruição do vetor dinâmico (liberação);
- movimentação, inserção e remoção arbitrária de elementos.

Dica: Reaproveite código dos exercícios 12 e 13 ou vice-versa.

Observação: Tente escrever esta biblioteca da maneira mais geral possível, para que possa ser utilizada com inteiros, caracteres ou outros tipos de dados.