UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
DIMAp/INSTITUTO METRÓPOLE DIGITAL
Lista de exercícios Modularização

Questão 1. (q1.c) Crie uma estrutura chamada *NumeroComplexo* com dois campos: real e imaginaria (float), que representam a parte real e a parte imaginária de um número complexo.

Use **typedef** para nomear **struct NumeroComplexo** também como **NumeroComplexo**. Implemente as seguintes funções:

```
//retorna um novo numero complexo que eh a soma x + y
NumeroComplexo soma(NumeroComplexo x, NumeroComplexo y);

//retorna um novo numero complexo que eh a diferenca x - y
NumeroComplexo diferenca(NumeroComplexo x, NumeroComplexo y);

//retorna um novo numero complexo que eh o produto xy
NumeroComplexo produto(NumeroComplexo x, NumeroComplexo y);

//retorna um novo numero complexo que eh o conjugado de x
NumeroComplexo conjugado(NumeroComplexo x);

//escreve o numero complexo x no seguinte formato: a+bi, onde a parte real e b
   parte imaginaria
//se a parte imaginaria for negativa, omita o sinal de +
//exemplo: para a = 3 e b = -2, a funcao deve escrever 3-2i e nao 3+-2i
void printNumeroComplexo(NumeroComplexo x);

//retorna 1 ou 0 dependendo se x e y possuem mesmos valores ou nao
int saoIguais(NumeroComplexo x, NumeroComplexo y);

//zera a parte imaginaria do numero complexo referenciado por ptr
void zerarImaginaria(NumeroComplexo *ptr);
```

Modularize seu código em um arquivo complexo.h e um arquivo complexo.c. Em um arquivo testecomplexo.c teste cada uma das funções para demonstrar que estão corretamente implementadas.

Questão 2 na página seguinte.

Questão 2. (q2.c)

Crie uma estrutura chamada *cor* com três campos: red (R), green (G) e blue (B), os três do tipo inteiro, que representam o componente vermelho, verde e azul de uma cor digital. Cada componente deve estar no intervalo de 0 a 255. A combinação desses 3 inteiros caracteriza uma cor. Por exemplo, o amarelo é representado por r = 255, g = 255 e b = 0.

Use **typedef** para nomear **struct cor** também como **cor**. Implemente as seguintes funções:

```
//altera a cor referenciado por ptr para os valores passados tambem como
   parametro
//caso algum desses componentes for maior que 255, deixe 255
//caso algum desses componentes for menor que 0, deixe 0
//exemplo: para r = 120, g = -40 e b = 300, os componentes serao alterados para
   120, 0 e 255, respectivamente
void alterarCor(Cor *ptr, int r, int g, int b);

//escreve as informacoes da cor na tela no seguinte formato: cor (r, g, b), onde
   r, g e b sao os valores de cada um dos componentes
void printCor(Cor x);

//retorna uma cor com r = 255, g = 255, b = 255
Cor obterBranco();

//retorna uma cor com r = 0, g = 0, b = 255
Cor obterAzul();

//reduz em 10 a intensidade de cada componente da cor referenciada por ptr
//caso algum dos componentes resulte em um valor negativo, deixe 0
void escurecer(Cor *ptr);

//aumenta em 10 a intensidade de cada componentes da cor referenciada por ptr
//caso algum dos componentes resulte em um valor maior que 255, deixe 255
void clarear(Cor *ptr);

//retorna uma nova cor aleatoria
Cor corAleatoria();
```

1. Modularize seu código em um arquivo cor.h e um arquivo cor.c. Em um arquivo testecor.c teste cada uma das funções para demonstrar que estão corretamente implementadas.
2. Em um arquivo coresAleatorias.c, leia um inteiro **n**, aloque dinamicamente um vetor de **n** cores, preencha-o com cores aleatórias (use a função corAleatoria) e, por fim, escreva na tela todas as cores (use a função printCor).
3. Explique por que as funções escurecer e clarear recebem como parâmetro um ponteiro para cor ao invés de simplesmente um parâmetro do tipo cor.