1. 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 é a parte real e b
//é a parte imaginaria (use duas casas decimais de precisão)
//se a parte imaginaria for negativa, omita o sinal de +
//exemplo: para a = 3 e b = -2, a funcao deve escrever 3.00-2.00i e nao 3.00+-2.00i
void printNumeroComplexo(NumeroComplexo x);

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

   Na função main faça os seguintes testes:

   1. Crie 3 números complexos: 2+3i, -3+5i e 2+3i
   2. Crie um novo número complexo que é resultado da soma dos dois primeiros e escreva-o na tela
   3. Crie um novo número complexo que é resultado da diferença dos dois primeiros e escreva-o na tela
   4. Crie um novo número complexo que é resultado do produto dos dois primeiros e escreva-o na tela
   5. Zere a parte imaginária do terceiro número e escreva-o, depois, na tela

2. ▷ Crie uma estrutura Ponto com dois números reais (x e y, ambos float) que representam um ponto no plano cartesiano. Use **typedef** para nomear **struct Ponto** também como **Ponto**. Implemente a seguinte função:

```
//retorna a distância euclidiana entre dois pontos
float distanciaEuclidiana(Ponto p1, Ponto p2);
```

   Crie uma estrutura Retangulo para representar um retângulo, utilizando a estrutura Ponto, na qual a primeira coordenada representa o canto superior esquerdo e a segunda coordenada o canto inferior direito. Use **typedef** para nomear **struct Retangulo** também como **Retangulo**. Implemente as seguintes funções:

```
//retorna a área do retangulo r
float area(Retangulo r);

//retorna o perímetro do retangulo r
float perimetro(Retangulo r);

//retorna o comprimento da diagonal (use a função distanciaEuclidiana com os dois
//cantos opostos do retângulo)
float comprimentoDiagonal(Retangulo r);

//escreva as informações do retângulo r no seguinte formato:
```

```
//(%.02f, %.02f) -- (%.02f, %.02f)
//onde esses floats são, respectivamente, x e y do canto superior esquerdo e inferior direito
void printRetangulo(Retangulo r);

//infla alterando os cantos do retângulo r: o canto esquerdo superior em (-0.5, 0.5)
//e o canto direito inferior em (0.5, -0.5)
void inflar(Retangulo *r);
```

Na função main faça os seguintes testes:

1. Leia 4 números reais e use-os para criar um retângulo com essas coordenadas (canto superior esquerdo e canto inferior direito, nessa ordem).
2. Escreva na tela as informações do retângulo usando a função printRetangulo
3. Escreva na tela a área e o perímetro desse retângulo (use a função area e perimetro).
4. Aumente o retângulo usando a função inflar e nos próximos dois passos confira se, de fato, a função fez com que o retângulo aumentasse o perímetro e a área.
5. Escreva na tela as informações do retângulo usando a função printRetangulo.
6. Escreva na tela a área e o perímetro desse retângulo (use a função area e perimetro).

3. 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();
```

Na função main faça os seguintes testes:

1. Crie uma cor com r = 120, g = 80, b = 140

2. Escreva as informações dessa cor na tela (use printCor)
3. Tente alterar a cor para ter r = 200, g = -20, b = 300 (use alterarCor)
4. Escreva as informações dessa cor na tela (use printCor)
5. Obtenha a cor azul e escreva suas informações na tela (use printCor)
6. Obtenha a cor branca e escreva suas informações na tela (use printCor)
7. Escureça 5 vezes a cor branca (use a função escurecer) e, após cada escurecimento, escreva suas informações na tela (use printCor)
8. 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).

Explique por que as funções escurecer e clarear recebem como parâmetro um ponteiro para cor ao invés de simplesmente um parâmetro do tipo cor. A resposta deve ser escrita em um arquivo cor.txt.

4. ▷ Crie uma estrutura chamada *Pessoa* com quatro campos: nome (string de até 30 caracteres), peso, altura (reais) e idade (inteiro).

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

```
//retorna uma nova Pessoa com nome nome, peso p, altura h e idade id
//lembre-se de usar strcpy para strings
Pessoa criarPessoa(char nome[], float p, float h, int id);

//retorna o IMC da Pessoa x, calculado por peso/(altura*altura)
float obterIMC(Pessoa x);

//escreve a Pessoa x no seguinte formato:
//Nome: n, Peso: p, Altura: h, Idade: id, IMC: imc (use a função obterIMC)
void printPessoa(Pessoa x);

//altera o peso da Pessoa referenciado por ptr para que este fique com IMC 22,
//ou seja, peso deve ser 22*altura*altura
void alterarPesoIMC22(Pessoa *ptr);
```

   Na função main faça os seguintes testes:

   1. Crie uma pessoa Amanda com peso 50, altura 1.50 e 30 anos (use a função criarPessoa) e, em seguida, escreva as informações dela com a função printPessoa.
   2. Crie uma pessoa Bruno com peso 70, altura 1.60 e 45 anos (use a função criarPessoa) e, em seguida, escreva as informações dela com a função printPessoa.
   3. Altere o peso da segunda pessoa para que ela fique com o IMC 22 (use a função alterarPesoIMC22) e, em seguida, escreva novamente as informações dela com a função printPessoa para ver se, de fato, o peso foi alterado de forma que o IMC ficasse 22.