O objetivo aqui será criar um programa que gera uma imagem com retângulos coloridos.

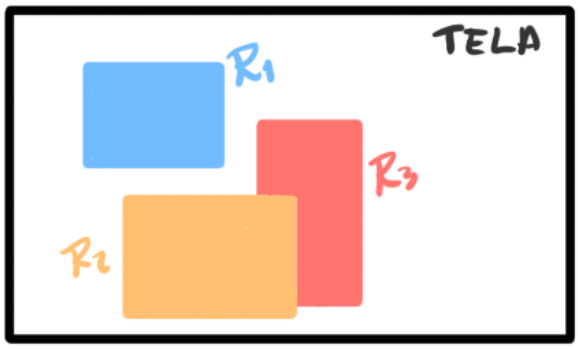

Figure 1: Tela com 3 retângulos de cores diferentes

Vamos aproveitar parte do que já foi feito: cor, ponto e retângulo. Vamos ter 4 estruturas e um conjunto de funções:

- Cor
- Ponto
- Retângulo
- Tela: representa a região onde serão desenhados os retângulos. O professor já disponibiliza os arquivos relativos à tela.

Cada uma dessas estruturas terá seu arquivo de cabeçalho (o arquivo .h) e um arquivo de implementação (.c). Por exemplo, se a estrutura for Predio, criaria um arquivo predio.h contendo:

```
#ifndef PREDIO_H
#define PREDIO_H

typedef struct Predio {
    //campos da estrutura
} Predio;

//esta função retorna um novo prédio com n andares
Predio funcao1(int n);
//esta função altera o número de vagas do prédio que reside no endereço contido em p
void funcao2(Predio *p, int novoNumeroVagas);
//e assinaturas das demais funções com comentários explicando o uso das funções

#endif
```

Continuando: criaria um arquivo predio.c contendo a implementação de cada função listada no predio.h.

Cada estrutura terá seu arquivo de teste. O comando para compilar cada teste está na última página deste PDF.

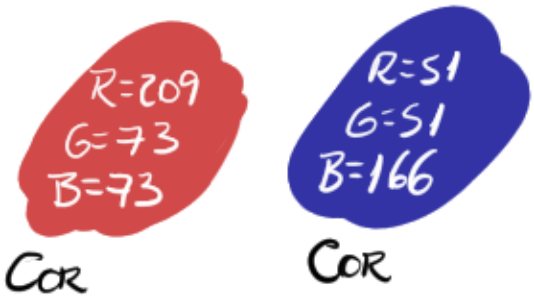

Figure 2: Duas cores

a. 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);

//retorna uma nova cor com valores r, g e b.
//use a função alterarCor para facilitar o seu trabalho em manter os valores entre 0 e 255
Cor criarCor(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 nova cor aleatoria
Cor corAleatoria();
```

   Crie um arquivo de teste testeCor.c para testar a implementação de cada uma das funções.

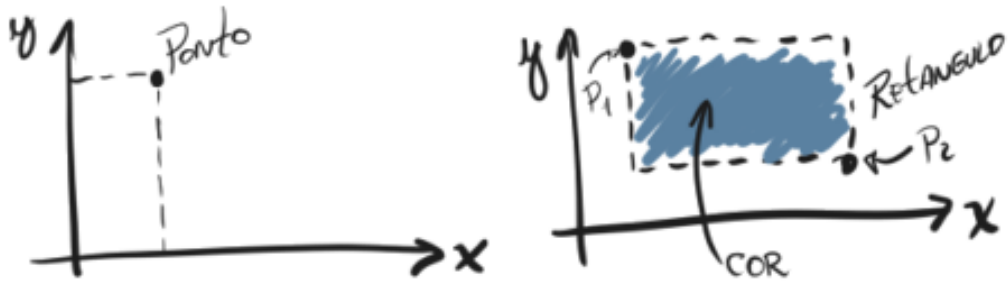

Figure 3: Ponto e Retângulo

b. 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 um arquivo de teste testePonto.c para testar a implementação de distanciaEuclidiana.

c. 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. **Ademais, acrescente um campo cor do tipo Cor**. Use **typedef** para nomear **struct Retangulo** também como **Retangulo**. Implemente as seguintes funções:

```
//retorna um novo retângulo
Retangulo criarRetangulo(float p1x, float p1y, float p2x, float p2y, Cor c);

//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);
```

Crie um arquivo de teste testeRetangulo.c para testar a implementação de cada uma das funções.

2. Crie um arquivo main.c que inclua tela.h.

```c
#include "tela.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main() {
    Tela t = criarTela(1200, 600);
    Cor c1 = criarCor(200, 100, 50);
    Cor c2 = criarCor(50, 0, 50);
    Retangulo r1 = criarRetangulo(20, 200, 290, 30, c1);
    Retangulo r2 = criarRetangulo(20, 300, 70, 20, c2);
    printRetangulo(r1);
    adicionarRetangulo(&t, r1);
    adicionarRetangulo(&t, r2);
    criarImagem(t);
    return 0;
}
```

Crie mais retângulos com cores diferentes.

3. Sumário

Arquivo	Include	Compilação
ponto.h	-	-
ponto.c	ponto.h math.h	gcc ponto.c -c
testePonto.c	ponto.h stdio.h	gcc testePonto.c ponto.o -lm
cor.h	-	-
cor.c	cor.h stdio.h stdlib.h	gcc cor.c -c
testeCor.c	cor.h	gcc testeCor.c cor.o
retangulo.h	-	-
retangulo.c	retangulo.h math.h stdio.h	gcc retangulo.c -c
testeRetangulo.c	retangulo.h	*1
main.c	tela.h	*2

(1)

```
gcc testeRetangulo.c retangulo.c ponto.c cor.c -lm
```

(2)

```
gcc main.c ponto.c retangulo.c cor.c tela.c -lm
```