For the programming task you have to use C++
A pull request has to be made for the solutions(C++ code and generated images).
The pull request is in your repository from the github classroom assignment:
https://classroom.github.com/a/zh9ighUl
For questions and help refer to the course's discord server:
https://discord.gg/kkr83dZS
Or the course's e-mail:
raytracingcourse@chaos.com

Task 1.

Generate an image where each pixel is colored according to the normalized direction of the ray fired towards/through it. Experiment with different resolutions and **aspect ratios**.
Assume that:

- The coordinate system you are working in is **Right-handed** with the **Y-axis** pointing upwards.
- The position of the camera is at **(0, 0, 0)** (the "world" origin), from where the rays should originate.
- The camera faces in the **-Z** direction.
- The image plane with pixels, through which the rays need to be fired, has a **Z**-coordinate of -1, i.e., it is at a distance of 1 from/in front of the camera.

Example:
For a pixel at the end of the first row of the image (**pixels[0][width - 1]**), if you have generated a ray with a normalized direction **rayDir = CRTVector(0.9, 0.8, -0.98)**, you can record its color as **pixels[0][width - 1] = CRTColor(rayDir.x * 255, rayDir.y * 255, ?)**. Decide how to use negative values as color components.

Pseudocode for an algorithm to generate a camera/primary rays:

- Loop over the image resolution
    - For x : width; y : height
    - At each pixel
        - Find its center, based on the raster coordinates
            - *x += 0.5; y += 0.5*
        - Convert raster coordinates to NDC space [0.0, 1.0]
            - *x /= width; y /= height*
        - Convert NDC coordinates to Screen space [-1.0, 1.0]
            - *x = (2.0 * x) - 1.0*
            - *y = 1.0 - (2.0 * y)*
        - Consider the aspect ratio
            - *x *= width / height*
        - Ray direction = (x, y, -1.0)
        - Normalaize ray direction vector
        - Store the ray with the calculated direction and origin

Example C++ code, for generating .ppm file:

```
#include <fstream>

/// Output image resolution
static const int imageWidth = 1920;
static const int imageHeight = 1080;

static const int maxColorComponent = 255;

int main() {
	std::ofstream ppmFileStream("crt_output_image.ppm", std::ios::out | std::ios::binary);
	ppmFileStream << "P3\n";
	ppmFileStream << imageWidth << " " << imageHeight << "\n";
	ppmFileStream << maxColorComponent << "\n";

	for (int rowIdx = 0; rowIdx < imageHeight; ++rowIdx) {
		for (int colIdx = 0; colIdx < imageWidth; ++colIdx) {
			ppmFileStream << "0 0 255\t";
		}
		ppmFileStream << "\n";
	}

	ppmFileStream.close();

	return 0;
}
```