```cpp
//Shader.h
#pragma once

#ifndef SHADER_H
#define SHADER_H

#include<GL/glew.h>

#include<string>
#include<fstream>
#include<sstream>
#include<iostream>

class Shader
{
public:
	//shader program ID
	unsigned int ID;
	//构造函数，读取shader代码、构建着色器
	Shader(const GLchar* vertexShaderPath, const GLchar* frgmentSahderPath);
	//使用/激活shader program
	void use();
	//uniform工具函数
	void setBool(const std::string& name, bool value) const;
	void setInt(const std::string& name, int value) const;
	void setFloat(const std::string& name, float value) const;

private:
	void checkCompileErrors(GLuint ID, std::string type);
};

#endif
```

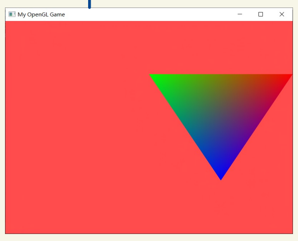
```cpp
//Shader.cpp
#include "Shader.h"

Shader::Shader(const GLchar* vertexShaderPath, const GLchar* fragmentShaderPath)
{
	//1、从文件路径获取顶点/片段着色器
	std::string vertexCode;
	std::string fragmentCode;
	std::ifstream vShaderFile;
	std::ifstream fShaderFile;
	//保证ifstream对象可以抛出异常
	vShaderFile.exceptions(std::ifstream::failbit | std::ifstream::badbit);  //failbit：逻辑上打不开文件，badbit：文档坏了
	fShaderFile.exceptions(std::ifstream::failbit | std::ifstream::badbit);
	try  //try-catch：try中遇到exception，就会立即跳入catch
	{
		//打开文件
		vShaderFile.open(vertexShaderPath);
		fShaderFile.open(fragmentShaderPath);
		std::stringstream vShaderStream, fShaderStream;
		//读取文件的缓冲到数据流中
		vShaderStream << vShaderFile.rdbuf();
		fShaderStream << fShaderFile.rdbuf();
		//关闭文件处理器
		vShaderFile.close();
		fShaderFile.close();
		//转换数据流到string
		vertexCode = vShaderStream.str();
		fragmentCode = fShaderStream.str();
	}
	catch (std::ifstream::failure &e)  //出现异常，执行
	{
		std::cout << "ERROR::SHADER::FILE_NOT_SUCCESFULLY_READ: " <<
			e.what() << std::endl;
	}
	const char* vShaderCode = vertexCode.c_str();
	const char* fShaderCode = fragmentCode.c_str();

	//2、编译着色器
	unsigned int vertex, fragment;
	/*int success;
	char infoLog[512];*/

	//顶点着色器
	vertex = glCreateShader(GL_VERTEX_SHADER);
	glShaderSource(vertex, 1, &vShaderCode, NULL);  //第三个形参类型：const GLchar *const *
	glCompileShader(vertex);
	//打印编译错误（如果有的话）
	/*glGetShaderiv(vertex, GL_COMPILE_STATUS, &success);
	if (!success) {
		glGetShaderInfoLog(vertex, 512, NULL, infoLog);
		std::cout << "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n" << infoLog << std::endl;
	}*/
	checkCompileErrors(vertex, "VERTEX");

	//片段着色器
	fragment = glCreateShader(GL_FRAGMENT_SHADER);
	glShaderSource(fragment, 1, &fShaderCode, NULL);  //第三个形参类型：const GLchar *const *
	glCompileShader(fragment);
	checkCompileErrors(fragment, "FRAGMENT");

	//链接为着色器程序
	ID = glCreateProgram();
	glAttachShader(ID, vertex);
	glAttachShader(ID, fragment);
	glLinkProgram(ID);
	//打印链接错误（如果有的话）
	/*glGetProgramiv(ID, GL_LINK_STATUS, &success);
	if (!success) {
		glGetProgramInfoLog(ID, 512, NULL, infoLog);
		std::cout << "ERROR::SHADER::PROGRAM::LINKING_FAILED\n" << infoLog << std::endl;
	}*/
	checkCompileErrors(ID, "PROGRAM");

	//删除着色器代码，因为已经链接到程序中了
	glDeleteShader(vertex);
	glDeleteShader(fragment);
}

void Shader::use()
{
	glUseProgram(ID);
}

void Shader::setBool(const std::string& name, bool value) const {
	glUniform1i(glGetUniformLocation(ID, name.c_str()), (int)value);  //把value转化为int，赋值给uniform变量
}

void Shader::setInt(const std::string& name, int value) const {
	glUniform1i(glGetUniformLocation(ID, name.c_str()), value);
}

void Shader::setFloat(const std::string& name, float value) const {
	glUniform1f(glGetUniformLocation(ID, name.c_str()), value);
}

void Shader::checkCompileErrors(GLuint ID, std::string type) {
	GLint success;
	GLchar infoLog[1024];
	if (type != "PROGRAM") { //对于其他（即代码），检查编译错误
		glGetShaderiv(ID, GL_COMPILE_STATUS, &success);
		if (!success) {
			std::cout << "ERROR::SHADER_COMPILATION_ERROE of type: "
				<< type << "\n" << infoLog << std::endl;
		}
	}
	else {  //对于shader程序，检查链接错误
		glGetProgramiv(ID, GL_LINK_STATUS, &success);
		if (!success) {
			glGetProgramInfoLog(ID, 1024, NULL, infoLog);
			std::cout << "ERROR::PROGRAM_LINKING_ERROR of type: "
				<< type << "\n" << infoLog << std::endl;
		}
	}
}
```

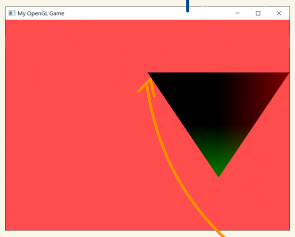
$$\begin{tikzcd}
\text{ifstream (文件)} \arrow[d] \\
\text{stringstream (流)} \arrow[d] \\
\text{string (变量)}
\end{tikzcd}$$

```cpp
//main.cpp
#include<iostream>
#define GLEW_STATIC
#include<GL/glew.h>
#include<GLFW/glfw3.h>
#include"Shader.h"
#include<cstdio>

float vertices[] = {
	0.5f,-0.5f,0.0f,    1.0f,0.0f,0.0f,
	-0.5f,-0.5f,0.0f,    0.0f,1.0f,0.0f,
	0.0f,0.5f,0.0f,   0.0f,0.0f,1.0f
};

void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
	glViewport(0, 0, width, height);
}

void processInput(GLFWwindow* window)
{
	if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)  //如果按下escape键
	{
		glfwSetWindowShouldClose(window, true);
	}
}

int main()
{

	glfwInit();  //初始化函式库
	glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);  //hint提示，主版本
	glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);  //副版本号
	glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);  //使用profile

	const char* version = (const char*)glGetString(GL_VERSION);
	printf("OpenGL Version : %s\n", version);  //为什么输出(NULL)? ? ?

	const GLubyte* name = glGetString(GL_VENDOR); //返回负责当前OpenGL实现厂商的名字
	const GLubyte* biaoshifu = glGetString(GL_RENDERER); //返回一个渲染器标识符，通常是个硬件平台
	const GLubyte* OpenGLVersion = glGetString(GL_VERSION); //返回当前OpenGL实现的版本号

	printf("OpenGL实现厂商的名字：%s\n", name);  //为什么输出(NULL)? ? ?
	printf("渲染器标识符：%s\n", biaoshifu);  //为什么输出(NULL)? ? ?
	printf("OpenGL实现的版本号：%s\n", OpenGLVersion);  //为什么输出(NULL)? ? ?

	//Open GLFW Window
	GLFWwindow* window = glfwCreateWindow(800, 600, "My OpenGL Game", NULL, NULL);
	if (window == NULL)                    //宽，高
	{
		printf("Open Window Failed!");
		glfwTerminate();
		return -1;
	}
	glfwMakeContextCurrent(window);  //tell GLFW to make the context of our window the main context on the current thread
	glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);  //回调函数

	//Init GLEW
	glewExperimental = true;
	if (glewInit() != GLEW_OK)
	{
		printf("Init GLEW Failed!");
		glfwTerminate();
		return -1;
	}

	Shader* shader = new Shader("vertexShaderSource.txt", "fragmentShaderSource.txt");

	glViewport(0, 0, 800, 600);

	unsigned int VAO;
	glGenVertexArrays(1, &VAO);
	unsigned int VBO;
	glGenBuffers(1, &VBO);
	// bind the Vertex Array Object first, then bind and set vertex buffer(s), and then configure vertex attributes(s).
	glBindVertexArray(VAO);

	glBindBuffer(GL_ARRAY_BUFFER, VBO);
	glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);    //vertices中的数据从CPU到GPU的array buffer，即VBO

	// position attribute
	glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)0);  //VBO中的坐标->VAO，0号顶点属性
	glEnableVertexAttribArray(0);
	// color attribute
	glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 6 * sizeof(float), (void*)(3 * sizeof(float)));  //VBO中的颜色->VAO，1号顶点属性
	glEnableVertexAttribArray(1);

	//渲染循环
	while (!glfwWindowShouldClose(window))
	{
		processInput(window);

		float offset = 0.5f;
		shader->setFloat("xOffset", offset);

		glClearColor(1.0f, 0.3f, 0.3f, 1.0f);
		glClear(GL_COLOR_BUFFER_BIT);//清哪个buffer

		shader->use();  //使用shader program
		glBindVertexArray(VAO);
		glDrawArrays(GL_TRIANGLES, 0, 3);

		glfwSwapBuffers(window);
		glfwPollEvents();
	}
	glDeleteVertexArrays(1, &VAO);
	glDeleteBuffers(1, &VBO);

	system("pause");
	glfwTerminate();
	return 0;
}
```

```glsl
//vertexShaderSource.txt
#version 330 core
layout(location = 0) in vec3 aPos;
layout(location = 1) in vec3 aColor;

out vec3 ourColor;
out vec3 ourPosition;

uniform float xOffset;

void main()
{
		gl_Position = vec4(aPos.x + xOffset, -aPos.y, aPos.z, 1.0f);
		ourColor = aColor;
		ourPosition = aPos;
}
```

```glsl
//fragmentShaderSource.txt
#version 330 core

in vec3 ourColor;
in vec3 ourPosition;

out vec4 FragColor;

void main()
{
	FragColor = vec4(ourColor, 1.0f);
	//FragColor = vec4(ourPosition, 1.0f);
}
```





```
/*
Answer to the question: Do you know why the bottom-left side is black?
-- --------------------------------------------------------------------
Think about this for a second: the output of our fragment's color is equal to the (interpolated)
coordinate of the triangle. What is the coordinate of the bottom-left point of our triangle? This
is (-0.5f, -0.5f, 0.0f). Since the xy values are negative they are clamped to a value of 0.0f. This
happens all the way to the center sides of the triangle since from that point on the values will be
interpolated positively again. Values of 0.0f are of course black and that explains the black side
of the triangle.
*/
```

原三角形(不加 xOffset 和 -y):