Roll No: 1803

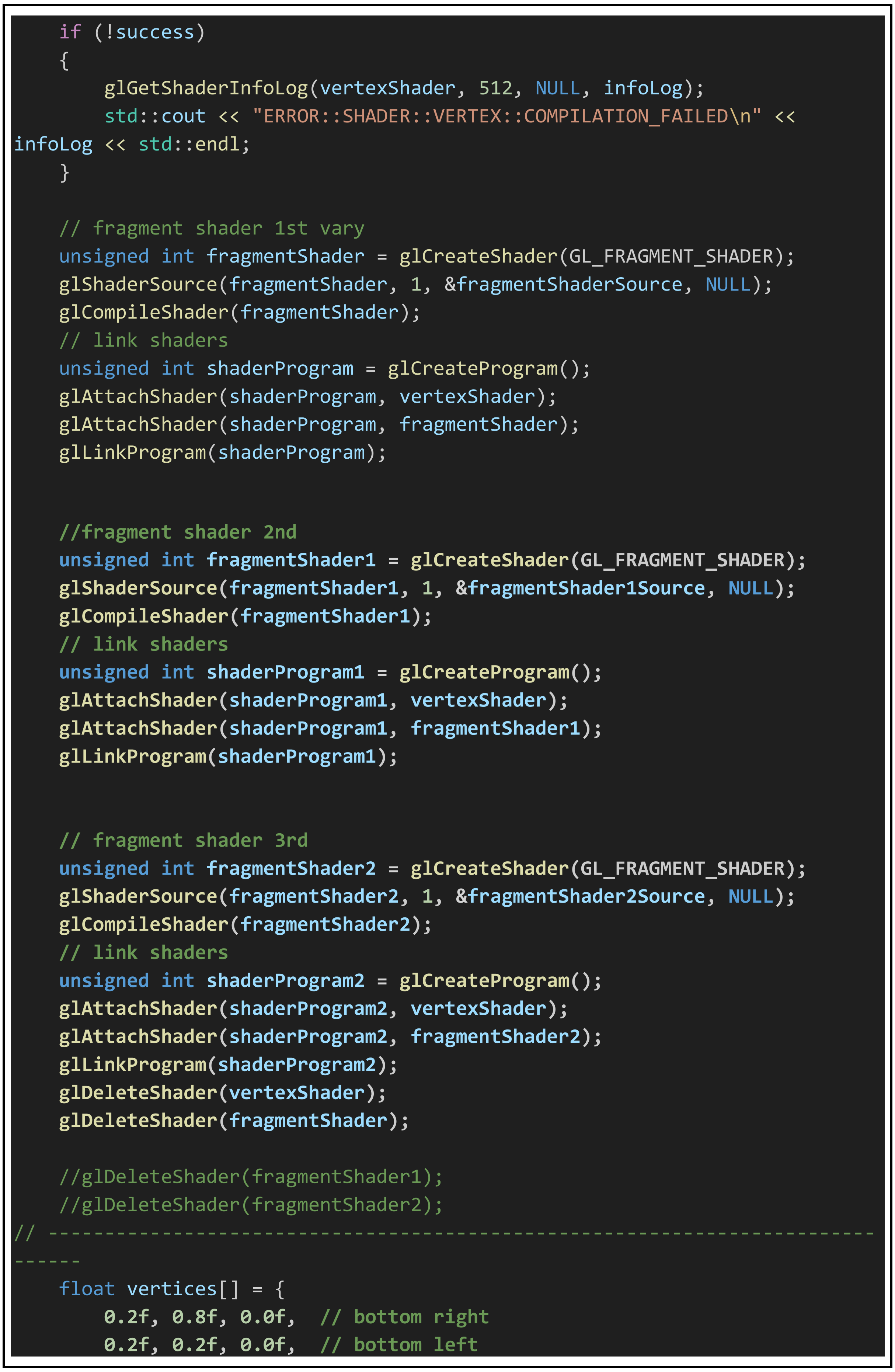
Lab Performance Test [1]
Lab Task Q[1]

Question1:

Show an OpenGL program which will show a right angle triangle with a blue color in the first quadrant.

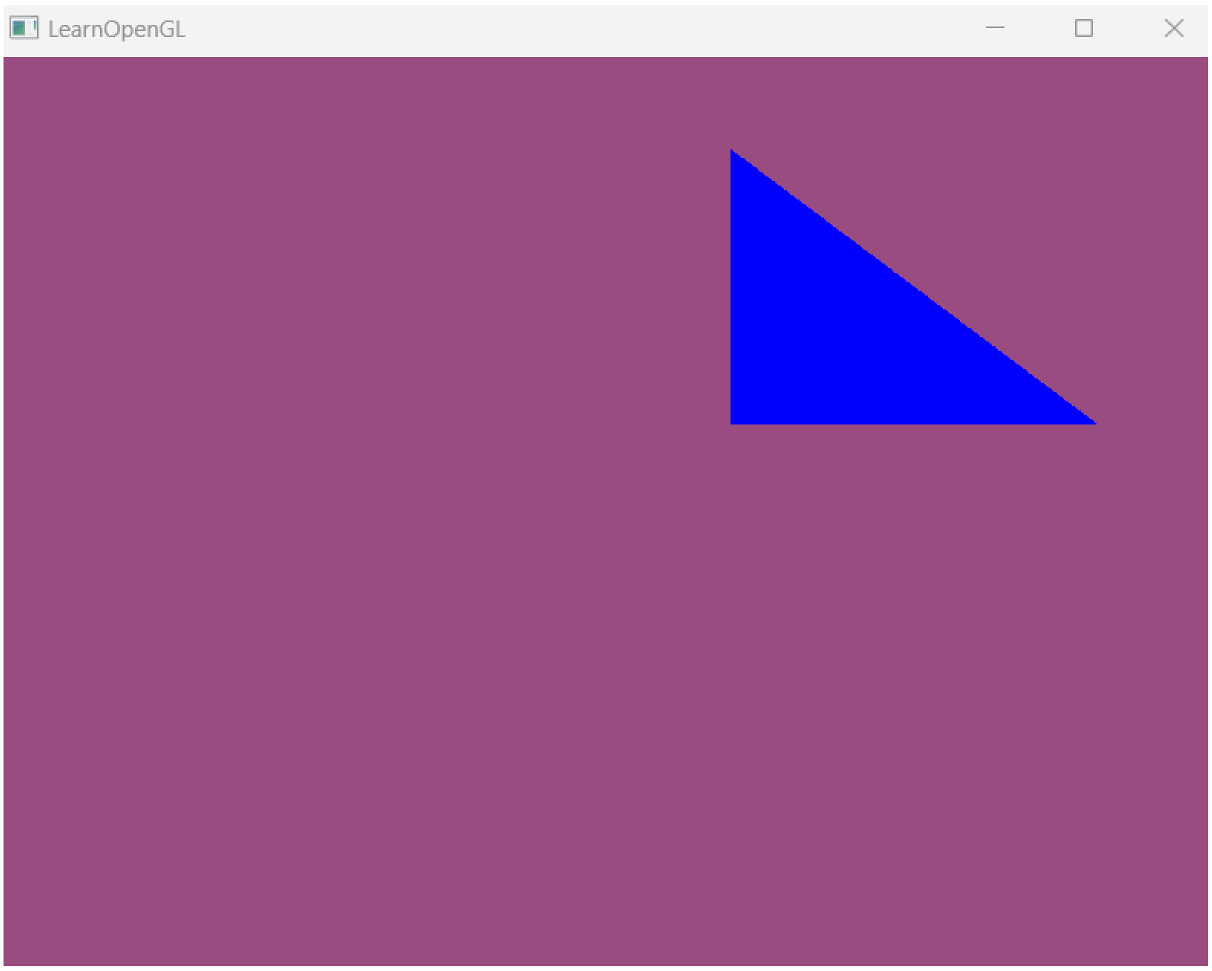
Solution (Bold your own written code):

```
#include "glad.h"
#include "glfw3.h"
// ----------------------------------------------------------------------------
------
#include <iostream>
#include <cmath>

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow *window);

const unsigned int SCR_WIDTH = 800;
const unsigned int SCR_HEIGHT = 600;

const char *vertexShaderSource ="#version 330 core\n"
    "layout (location = 0) in vec3 aPos;\n"
    "void main()\n"
    "{\n"
    "   gl_Position = vec4(aPos, 1.0);\n"
    "}\0";
// ----------------------------------------------------------------------------
------



const char *fragmentShaderSource = "#version 330 core\n"
    "out vec4 FragColor;\n"
    "uniform vec4 ourColor;\n"
    "void main()\n"
    "{\n"
    "   FragColor = ourColor;\n"
    "}\n\0";

const char *fragmentShader1Source = "#version 330 core\n"
    "out vec4 FragColor;\n"
    "void main()\n"
    "{\n"
    "   FragColor = vec4(0.0f, 0.0f, 1.0f, 1.0f);\n"
    "}\n\0";

const char *fragmentShader2Source = "#version 330 core\n"
    "out vec4 FragColor;\n"
    "void main()\n"
    "{\n"
    "   FragColor = vec4(0.0f, 0.0f, 1.0f, 1.0f);\n"
    "}\n\0";


// -------------------------------------------------------------------------
------
int main()
{
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef _APPLE_
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT,
"LearnOpenGL", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }

    unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
    glCompileShader(vertexShader);
    int success;
    char infoLog[512];
    glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
```



```
    if (!success)
    {
        glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n" <<
infoLog << std::endl;
    }

    // fragment shader 1st vary
    unsigned int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
    glCompileShader(fragmentShader);
    // link shaders
    unsigned int shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);


    //fragment shader 2nd
    unsigned int fragmentShader1 = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader1, 1, &fragmentShader1Source, NULL);
    glCompileShader(fragmentShader1);
    // link shaders
    unsigned int shaderProgram1 = glCreateProgram();
    glAttachShader(shaderProgram1, vertexShader);
    glAttachShader(shaderProgram1, fragmentShader1);
    glLinkProgram(shaderProgram1);


    // fragment shader 3rd
    unsigned int fragmentShader2 = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader2, 1, &fragmentShader2Source, NULL);
    glCompileShader(fragmentShader2);
    // link shaders
    unsigned int shaderProgram2 = glCreateProgram();
    glAttachShader(shaderProgram2, vertexShader);
    glAttachShader(shaderProgram2, fragmentShader2);
    glLinkProgram(shaderProgram2);
    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);

    //glDeleteShader(fragmentShader1);
    //glDeleteShader(fragmentShader2);
// ------------------------------------------------------------------------
------
    float vertices[] = {
        0.2f, 0.8f, 0.0f,  // bottom right
        0.2f, 0.2f, 0.0f,  // bottom left
        0.8f,  0.2f, 0.0f,   // top

    };

    unsigned int VBO, VAO;
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);

    glBindVertexArray(VAO);
    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices,
GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(0);
/*
    glBindVertexArray(VAOs[1]);
    glBindBuffer(GL_ARRAY_BUFFER, VBOs[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(sec_vertices), sec_vertices,
GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(0);
*/

// ----------------------------------------------------------------------------
------
    while (!glfwWindowShouldClose(window))
    {
        processInput(window);
// ----------------------------------------------------------------------------
------



        glClearColor(0.6f, 0.3f, 0.5f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);


/*
        // update shader uniform
        glUseProgram(shaderProgram);
        double  timeValue = glfwGetTime();
        float greenValue = static_cast<float>(sin(timeValue) / 2.0 + 0.5);
        int vertexColorLocation = glGetUniformLocation(shaderProgram,
"ourColor");
        glUniform4f(vertexColorLocation, 0.0f, greenValue, 0.0f, 1.0f);
*/
        // render the triangle
        glUseProgram(shaderProgram1);
        glBindVertexArray(VAO);
        glDrawArrays(GL_TRIANGLES, 0, 6);

        glUseProgram(shaderProgram2);
        glBindVertexArray(VAO);
        glDrawArrays(GL_TRIANGLES, 0, 6);

        // -----------------------------------------------------------------
--------------
        glfwSwapBuffers(window);
        glfwPollEvents();
        // -----------------------------------------------------------------
--------------
    }


    glDeleteVertexArrays(1, &VAO);
    glDeleteBuffers(2, &VBO);
    glDeleteProgram(shaderProgram);
    //glDeleteProgram(shaderProgram1);
    //glDeleteProgram(shaderProgram2);



    // ------------------------------------------------------------------
    glfwTerminate();
    return 0;
}

void processInput(GLFWwindow *window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}

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

Output:

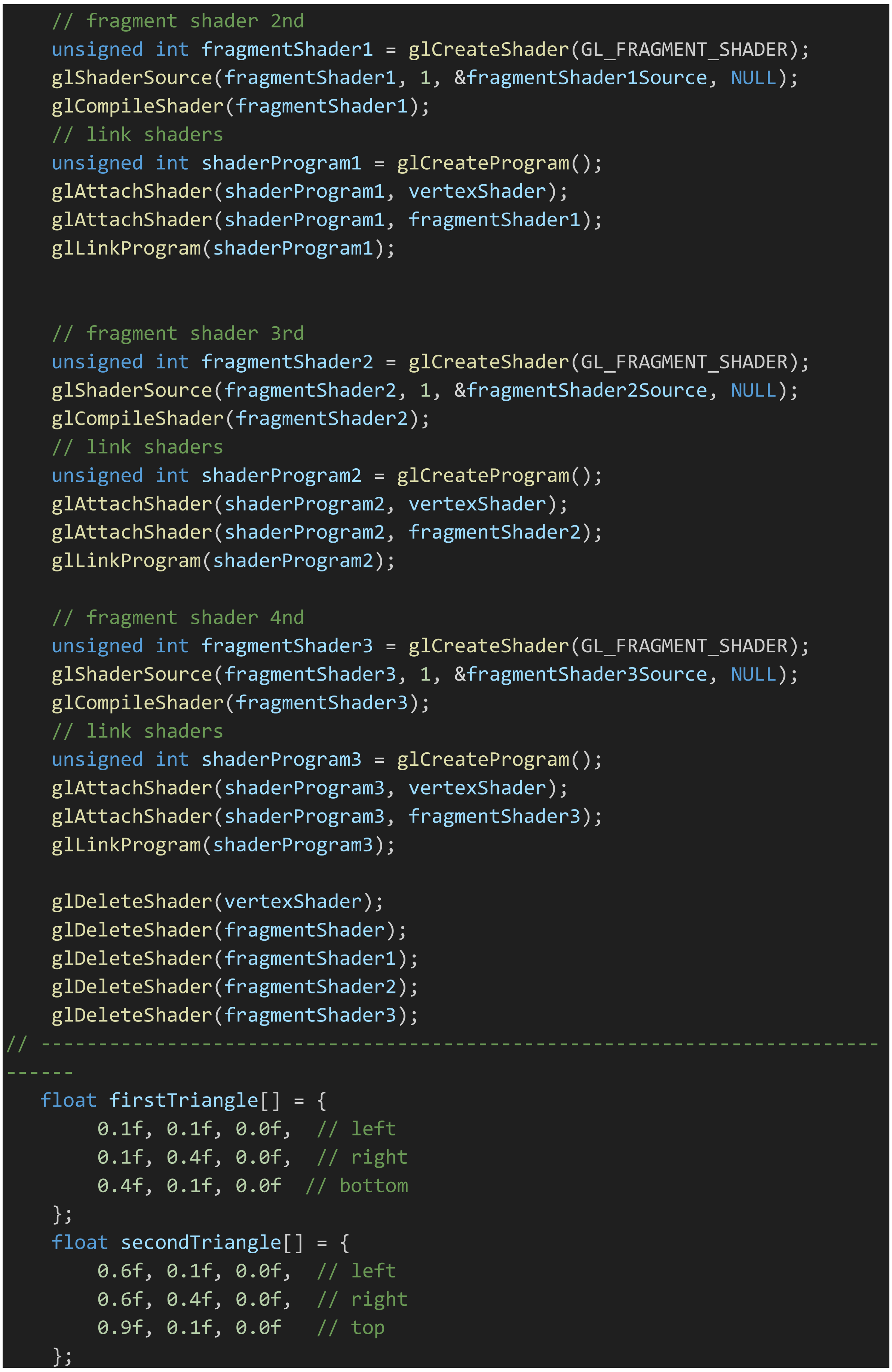


Question2:
Show an OpenGL program which will show three different triangles with red, green and blue color in blue background using different fragment shader. Two in the first and one in the fourth.

Solution (Bold your own written code):

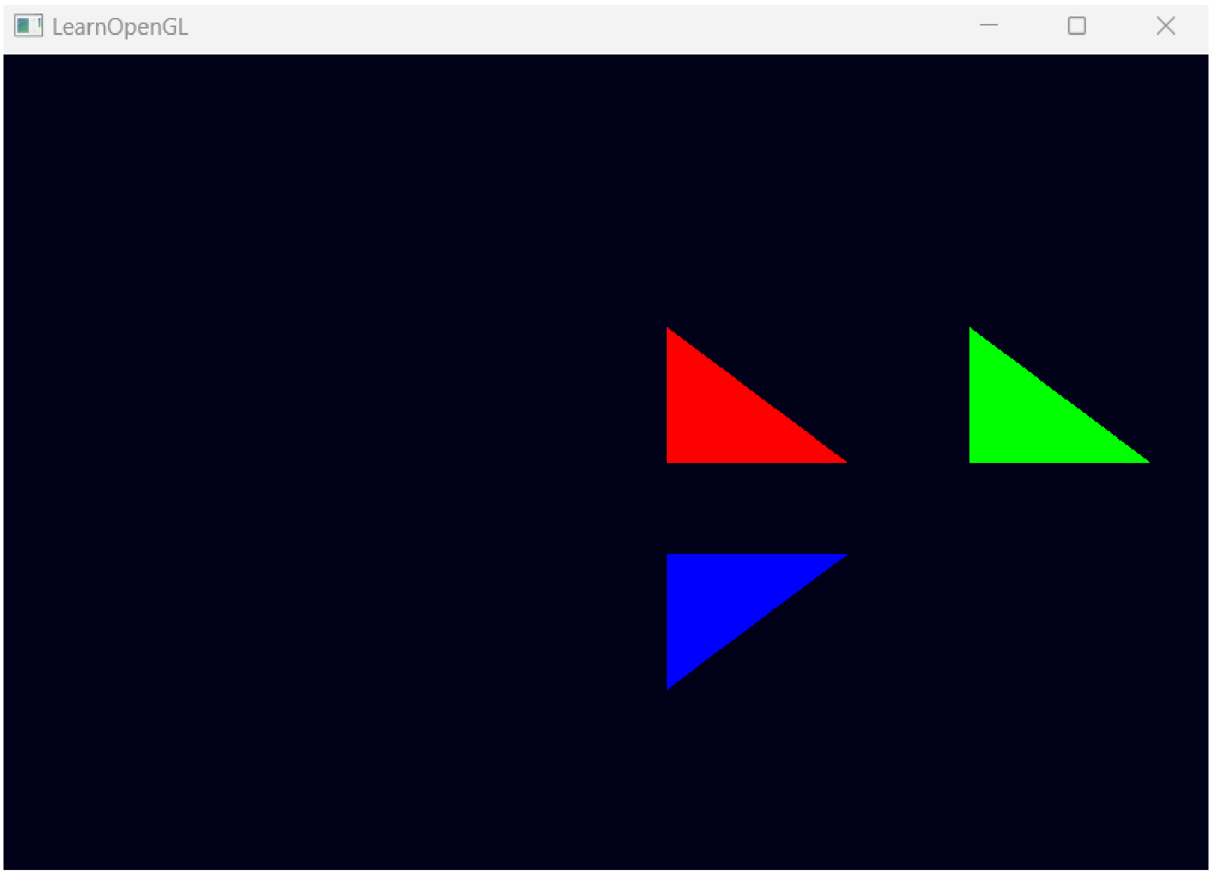
```
#include "glad.h"
#include "glfw3.h"
// ----------------------------------------------------------------------------
------
#include <iostream>
#include <cmath>

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void processInput(GLFWwindow *window);

const unsigned int SCR_WIDTH = 800;
const unsigned int SCR_HEIGHT = 600;
```

```cpp
const char *vertexShaderSource ="#version 330 core\n"
    "layout (location = 0) in vec3 aPos;\n"
    "void main()\n"
    "{\n"
    "   gl_Position = vec4(aPos, 1.0);\n"
    "}\0";
// ---------------------------------------------------------------------------
------



const char *fragmentShaderSource = "#version 330 core\n"
    "out vec4 FragColor;\n"
    "uniform vec4 ourColor;\n"
    "void main()\n"
    "{\n"
    "   FragColor = ourColor;\n"
    "}\n\0";

const char *fragmentShader1Source = "#version 330 core\n"
    "out vec4 FragColor;\n"
    "void main()\n"
    "{\n"
    "   FragColor = vec4(1.0f, 0.0f, 0.0f, 1.0f);\n"
    "}\n\0";

const char *fragmentShader2Source = "#version 330 core\n"
    "out vec4 FragColor;\n"
    "void main()\n"
    "{\n"
    "   FragColor = vec4(0.0f, 1.0f, 0.0f, 1.0f);\n"
    "}\n\0";

const char *fragmentShader3Source = "#version 330 core\n"
    "out vec4 FragColor;\n"
    "void main()\n"
    "{\n"
    "   FragColor = vec4(0.0f, 0.0f, 1.0f, 1.0f);\n"
    "}\n\0";


// ---------------------------------------------------------------------------
------
int main()
{
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT,
"LearnOpenGL", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);

    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }

    unsigned int vertexShader = glCreateShader(GL_VERTEX_SHADER);
    glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);
    glCompileShader(vertexShader);
    int success;
    char infoLog[512];
    glGetShaderiv(vertexShader, GL_COMPILE_STATUS, &success);
    if (!success)
    {
        glGetShaderInfoLog(vertexShader, 512, NULL, infoLog);
        std::cout << "ERROR::SHADER::VERTEX::COMPILATION_FAILED\n" <<
infoLog << std::endl;
    }

    // fragment shader 1st vary
    unsigned int fragmentShader = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);
    glCompileShader(fragmentShader);
    // link shaders
    unsigned int shaderProgram = glCreateProgram();
    glAttachShader(shaderProgram, vertexShader);
    glAttachShader(shaderProgram, fragmentShader);
    glLinkProgram(shaderProgram);
```



```
    // fragment shader 2nd
    unsigned int fragmentShader1 = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader1, 1, &fragmentShader1Source, NULL);
    glCompileShader(fragmentShader1);
    // link shaders
    unsigned int shaderProgram1 = glCreateProgram();
    glAttachShader(shaderProgram1, vertexShader);
    glAttachShader(shaderProgram1, fragmentShader1);
    glLinkProgram(shaderProgram1);


    // fragment shader 3rd
    unsigned int fragmentShader2 = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader2, 1, &fragmentShader2Source, NULL);
    glCompileShader(fragmentShader2);
    // link shaders
    unsigned int shaderProgram2 = glCreateProgram();
    glAttachShader(shaderProgram2, vertexShader);
    glAttachShader(shaderProgram2, fragmentShader2);
    glLinkProgram(shaderProgram2);

    // fragment shader 4nd
    unsigned int fragmentShader3 = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(fragmentShader3, 1, &fragmentShader3Source, NULL);
    glCompileShader(fragmentShader3);
    // link shaders
    unsigned int shaderProgram3 = glCreateProgram();
    glAttachShader(shaderProgram3, vertexShader);
    glAttachShader(shaderProgram3, fragmentShader3);
    glLinkProgram(shaderProgram3);

    glDeleteShader(vertexShader);
    glDeleteShader(fragmentShader);
    glDeleteShader(fragmentShader1);
    glDeleteShader(fragmentShader2);
    glDeleteShader(fragmentShader3);
// -------------------------------------------------------------------------
------
   float firstTriangle[] = {
        0.1f, 0.1f, 0.0f,  // left
        0.1f, 0.4f, 0.0f,  // right
        0.4f, 0.1f, 0.0f  // bottom
    };
    float secondTriangle[] = {
        0.6f, 0.1f, 0.0f,  // left
        0.6f, 0.4f, 0.0f,  // right
        0.9f, 0.1f, 0.0f   // top
    };
    float thirdTriangle[] = {
        0.1f, -0.1f, 0.0f,  // left
        0.1f, -0.4f, 0.0f,  // right
        0.4f, -0.1f, 0.0f  // bottom
    };

    unsigned int VBOs[3], VAOs[3];

    glGenVertexArrays(3, VAOs);
    glGenBuffers(3, VBOs);

    glBindVertexArray(VAOs[0]);
    glBindBuffer(GL_ARRAY_BUFFER, VBOs[0]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(firstTriangle), firstTriangle,
GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(0);

    glBindVertexArray(VAOs[1]);
    glBindBuffer(GL_ARRAY_BUFFER, VBOs[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(secondTriangle), secondTriangle,
GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(0);

    glBindVertexArray(VAOs[2]);
    glBindBuffer(GL_ARRAY_BUFFER, VBOs[2]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(thirdTriangle), thirdTriangle,
GL_STATIC_DRAW);
    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float),
(void*)0);
    glEnableVertexAttribArray(0);
// -----------------------------------------------------------------------------
------
    while (!glfwWindowShouldClose(window))
    {
        processInput(window);
// -----------------------------------------------------------------------------
------


        glClearColor(0.0f, 0.0f, 0.1f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT);
        // update shader uniform
         glUseProgram(shaderProgram);
         double  timeValue = glfwGetTime();
         float greenValue = static_cast<float>(sin(timeValue) / 2.0 + 0.5);
         int vertexColorLocation = glGetUniformLocation(shaderProgram,
"ourColor");
         glUniform4f(vertexColorLocation, 0.0f, greenValue, 0.0f, 1.0f);

        // render the triangle
        glUseProgram(shaderProgram1);
        glBindVertexArray(VAOs[0]);
        glDrawArrays(GL_TRIANGLES, 0, 3);

        glUseProgram(shaderProgram2);
        glBindVertexArray(VAOs[1]);
        glDrawArrays(GL_TRIANGLES, 0, 3);

        glUseProgram(shaderProgram3);
        glBindVertexArray(VAOs[2]);
        glDrawArrays(GL_TRIANGLES, 0, 3);
        // -----------------------------------------------------------------
--------------
        glfwSwapBuffers(window);
        glfwPollEvents();
        // -----------------------------------------------------------------
--------------
    }
    glDeleteVertexArrays(3, VAOs);
    glDeleteBuffers(3, VBOs);
    glDeleteProgram(shaderProgram);
    glDeleteProgram(shaderProgram1);
    glDeleteProgram(shaderProgram2);
    glDeleteProgram(shaderProgram3);
    // ------------------------------------------------------------------
    glfwTerminate();
    return 0;
}

void processInput(GLFWwindow *window)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
}

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

Output:


LearnOpenGL