```cpp
/*
 * Aplicacion TArea Julia
 *
 *  Created on: 2/13/2014
 *      Author: Miguel Tlapa Juarez
 */

#include <iostream>
#include <GL/glew.h>
#include <GL/freeglut.h>
#include "Utils.h"
using namespace std;

GLuint program_id;
GLuint vertex_position_loc;
GLuint vertex_color_loc;



GLuint vertexArrayIds[1];

void initShader(){
    GLuint v_shader_id = Utils::compileShader("Shaders/position_color.vsh", GL_VERTEX_SHADER);
    if(!Utils::shaderCompiled(v_shader_id)) return;
    GLuint f_shader_id = Utils::compileShader("Shaders/julia.fsh", GL_FRAGMENT_SHADER);
    if(!Utils::shaderCompiled(f_shader_id)) return;

    program_id = glCreateProgram();
    glAttachShader(program_id, v_shader_id);
    glAttachShader(program_id, f_shader_id);
    glLinkProgram(program_id);
    vertex_position_loc = glGetAttribLocation(program_id, "vertex_position");
    vertex_color_loc = glGetUniformLocation(program_id, "position_window");

}

void createRectangle() {
    glGenVertexArrays(1,vertexArrayIds);
    glBindVertexArray(vertexArrayIds[0]);
    GLuint bufferIds[2];
    glGenBuffers(2, bufferIds);
    float rectanglePos[] = {-0.99, -0.99,
                             0.99, -0.99,
                            -0.99,  0.99,
                             0.99,  0.99
    };

    float rectangleColor[] = {0, 0, 0,
                              0, 0, 0,
                              0, 0, 0,
                              0, 0, 0
    };
    glBindBuffer(GL_ARRAY_BUFFER, bufferIds[0]);
//  Transferir los datos del triángulo al servidor (GPU)
    glBufferData(GL_ARRAY_BUFFER, sizeof(rectanglePos), rectanglePos, GL_STATIC_DRAW);
    glVertexAttribPointer(vertex_position_loc, 2, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(vertex_position_loc);

    glBindBuffer(GL_ARRAY_BUFFER, bufferIds[1]);
    glBufferData(GL_ARRAY_BUFFER, sizeof(rectangleColor), rectangleColor, GL_STATIC_DRAW);
    glVertexAttribPointer(vertex_color_loc, 3, GL_FLOAT, GL_FALSE, 0, 0);
    glEnableVertexAttribArray(vertex_color_loc);
}


void keyboardFunc3(unsigned char key, int x, int y) {
    cout << key << " " << (int) key << endl;
```

```
    if(key == 27) exit(0);
}

void specialFunc3(int key, int x, int y) {
//  Si se presionó F4 y la tecla Alt está activa
    if(key == GLUT_KEY_F4 && glutGetModifiers() == GLUT_ACTIVE_ALT)
        exit(0);
}

void displayFunc3() {
//  Limpiar búfer de color con el color indicado por glClearColor
    glClear(GL_COLOR_BUFFER_BIT);
    glUseProgram(program_id);
    glBindVertexArray(vertexArrayIds[0]);
//  Dibujar la primitiva (draw-call)
    glUniform2f(vertex_color_loc, glutGet(GLUT_WINDOW_WIDTH),glutGet(GLUT_WINDOW_HEIGHT));
    glDrawArrays(GL_TRIANGLE_STRIP, 0, 4);
//  Intercambiar búfer frontal / posterior
    glClearColor(0.0, 0.0, 0.0, 1.0);
    glutSwapBuffers();
    glutPostRedisplay();
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);

    int ww = 600;
    int wh = 600;
    int sw = glutGet(GLUT_SCREEN_WIDTH);
    int sh = glutGet(GLUT_SCREEN_HEIGHT);
    glutInitWindowPosition((sw - ww) / 2, (sh - wh) / 2);
    glutInitWindowSize(ww, wh);

//  Que soporte doble búfer y pruebas de profundidad
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_DEPTH);
//  Crear la ventana en memoria
    glutCreateWindow("Fractal Julia");
//  Para pantalla completa
//  glutEnterGameMode();

//  Cuando se presione la tecla ESC, salir de la aplicación
    glutKeyboardFunc(keyboardFunc3);
//  Qué hacer cuando se repinte la ventana: dibujar los arreglos de vértices
    glutDisplayFunc(displayFunc3);
    glewInit();
    initShader();
    if(GLEW_VERSION_3_3) cout << "Soporta OpenGL 3.30" << endl;
    else cout << "No soporta OpenGL 3.30" << endl;
    createRectangle();

    cout << "Versión: " << glGetString(GL_VERSION) << endl;

//  Establecemos color de fondo: se utilizará cuando se limpie el búfer de color
//  Formato: r, g, b, a


    //  Desplegar la ventana continuamente
    glutMainLoop();
    return 0;
}
```