```cpp
#include "Shader.h"
//#include<iostream>
//#include <fstream>

Shader::Shader()
{
    shaderID = 0;
    uniformModel = 0;
    uniformProjection = 0;
}

void Shader::createFromString(const char* vertexCode, const char* fragmentCode)

{
    compileShader(vertexCode, fragmentCode);
}

void Shader::createFromFiles(const char* vertexLocation, const char*
  fragmentLocation)
{
    std::string vertexString = ReadFile(vertexLocation);
    std::string fragmentString = ReadFile(fragmentLocation);
    const char* vertexCode = vertexString.c_str();
    const char* fragmentCode = fragmentString.c_str();

    compileShader(vertexCode, fragmentCode);

}

std::string Shader::ReadFile(const char* fileLocation)
{
    std::string content;
    std::ifstream fileStream(fileLocation, std::ios::in);

    if (!fileStream.is_open()) {


        printf("Failed to read %s| File doesn not exist.", fileLocation);
        return "";
    }

    std::string line = "";
    while (!fileStream.eof())
    {
        std::getline(fileStream, line);
        content.append(line + "\n");
    }
    fileStream.close();
    return content;
}

void Shader::compileShader(const char* vertexCode, const char* fragmentCode)
```

```
52
53  {
54
55      shaderID = glCreateProgram();
56      if (!shaderID) {
57          printf("error fragment creating ...");
58
59          return;
60
61      }
62
63
64      AddShader(shaderID, vertexCode, GL_VERTEX_SHADER);
65      AddShader(shaderID, fragmentCode, GL_FRAGMENT_SHADER);
66
67
68      //chunk of code 27:30
69
70      GLint result = 0;
71      GLchar eLog[1024] = { 0 };
72
73      glLinkProgram(shaderID);
74      glGetProgramiv(shaderID, GL_LINK_STATUS, &result);
75
76      if (!result)
77      {
78          glGetProgramInfoLog(shaderID, sizeof(eLog), NULL, eLog);
79          printf("Error Linking Program  %d shader: '%s ", eLog);
80          return;
81
82      }
83      glValidateProgram(shaderID);
84      glGetProgramiv(shaderID, GL_VALIDATE_STATUS, &result);
85
86      if (!result)
87      {
88          glGetProgramInfoLog(shaderID, sizeof(eLog), NULL, eLog);
89          printf("Error Validating Program  %d shader: '%s ", eLog);
90          return;
91
92      }
93      //uniformModel = glGetUniformLocation(shader, "xMove");
94      uniformModel = glGetUniformLocation(shaderID, "model"); //idy model axata naw ↵
           variably uniformModel
95                                                              //uniformProjection
96      uniformProjection = glGetUniformLocation(shaderID, "projection"); //idy       ↵
          projection axata naw variably uniformProjection
97
98
99  }
100
101
```

```
102
103 GLuint Shader::GetProjectionLocation()
104 {
105
106     return uniformProjection;
107
108 }
109 GLuint Shader::GetModelLocation()
110 {
111
112     return uniformModel;
113
114 }
115
116 void Shader::useShader()
117
118 {
119     glUseProgram(shaderID);
120
121 }
122
123 void Shader::clearShader() {
124     if (shaderID != 0) {
125
126         glDeleteProgram(shaderID);
127         shaderID = 0;
128
129         }
130     uniformModel = 0;
131     uniformProjection = 0;
132 }
133
134 void Shader::AddShader(GLuint theProgram, const char* shaderCode, GLenum
      shaderType)
135
136 {
137
138     GLuint theShader = glCreateShader(shaderType);
139     const GLchar* theCode[1];
140     theCode[0] = shaderCode;
141
142     GLint codeLength[1];
143     codeLength[0] = strlen(shaderCode);
144
145     glShaderSource(theShader, 1, theCode, codeLength);
146     glCompileShader(theShader);//////////////////////////
147
148     GLint result = 0;
149     GLchar eLog[1024] = { 0 };
150
151
152     glGetShaderiv(theShader, GL_COMPILE_STATUS, &result);
```

```
153
154     if (!result)
155         {
156             glGetShaderInfoLog(theShader, sizeof(eLog), NULL, eLog);
157             printf("Error compiling %d shader: '%s'\n", shaderType, eLog);
158                return;
159
160     }
161     glAttachShader(theProgram, theShader);
162
163 }
164
165 Shader::~Shader()
166 {
167     clearShader();
168
169 }
170
```