Finding and fixing GL errors (C++ style)

Q: What‘s wrong with my code?

```
void Renderer::render()
{
    glClear(GL_COLOR_BUFFER_BIT);

    glEnable(GL_DEPTH);

    glViewport(0, 0, viewport_width, viewport_height);

    glUseProgram(prog);
    glBindVertexArray(vao);
    glDrawArrays(GL_TRIANGLES, 0, 3);

    swapBuffers();
}
```

A: That‘s hard to say…

OpenGL Errors

- Errors can occur during any GL command
- Often fails silently(!)
- There is no built-in feedback! Program may crash **long** after first error
- Error state has to be queried manually through **glGetError()**

Error codes in OpenGL

https://www.opengl.org/sdk/docs/man/docbook4/xhtml/glGetError.xml

Name

glGetError — return error information

C Specification

```
GLenum glGetError( void);
```

Description

`glGetError` returns the value of the error flag. Each detectable error is assigned a numeric code and symbolic name. When an error occurs, the error flag is set to the appropriate error code value. No other errors are recorded until `glGetError` is called, th
`glGetError` returns `GL_NO_ERROR`, there has been no detectable error since the last call to `glGetError`, or since the GL was initialized.

To allow for distributed implementations, there may be several error flags. If any single error flag has recorded an error, the value of that flag is returned and that flag is reset to `GL_NO_ERROR` when `glGetError` is called. If more than one flag has recorded a
Thus, `glGetError` should always be called in a loop, until it returns `GL_NO_ERROR`, if all error flags are to be reset.

Initially, all error flags are set to `GL_NO_ERROR`.

The following errors are currently defined:

`GL_NO_ERROR`

No error has been recorded. The value of this symbolic constant is guaranteed to be 0.

`GL_INVALID_ENUM`

An unacceptable value is specified for an enumerated argument. The offending command is ignored and has no other side effect than to set the error flag.

`GL_INVALID_VALUE`

A numeric argument is out of range. The offending command is ignored and has no other side effect than to set the error flag.

`GL_INVALID_OPERATION`

The specified operation is not allowed in the current state. The offending command is ignored and has no other side effect than to set the error flag.

`GL_INVALID_FRAMEBUFFER_OPERATION`

The framebuffer object is not complete. The offending command is ignored and has no other side effect than to set the error flag.

1) One exception to interpret them all

```
class GLException : public std::exception
{
private:
    GLint error_code;
public:
    GLException(GLint error) : error_code(error) {}
    virtual const char* what() const noexcept
    {
        switch (error_code)
        {
        case (GL_INVALID_ENUM):
            return "Enumeration parameter is not a legal enumeration for that function.";
        case (GL_INVALID_VALUE):
            return "Illegal parameter value for that function.";
        case (GL_INVALID_OPERATION):
            return "This operation cannot be executed in the current state of OpenGL.";
        case (GL_STACK_OVERFLOW):
            return "Stack overflow occurred.";
        case (GL_STACK_UNDERFLOW):
            return "Stack underflow occurred.";
        case (GL_OUT_OF_MEMORY):
            return "Out of memory.";
        case (GL_INVALID_FRAMEBUFFER_OPERATION):
            return "Operation could not be performed in the current state of the frame buffer.";
        default:
            return "Unknown error! Please check error code!";
        }
    }
};
```

2) Check, notify and throw (general)

```
#define GL_SAFE_CALL(A) ([&]                          \
{                                                     \
    struct error_guard                                \
    {                                                 \
        ~error_guard() noexcept(false)                \
        {                                             \
            GLenum error = glGetError();              \
            if (error != GL_NO_ERROR)                 \
            {                                         \
                GLException ex(error);                \
                std::cerr << ex.what();               \
                throw ex;                             \
            }                                         \
        }                                             \
    } guard;                                          \
    return A;                                         \
}())
```

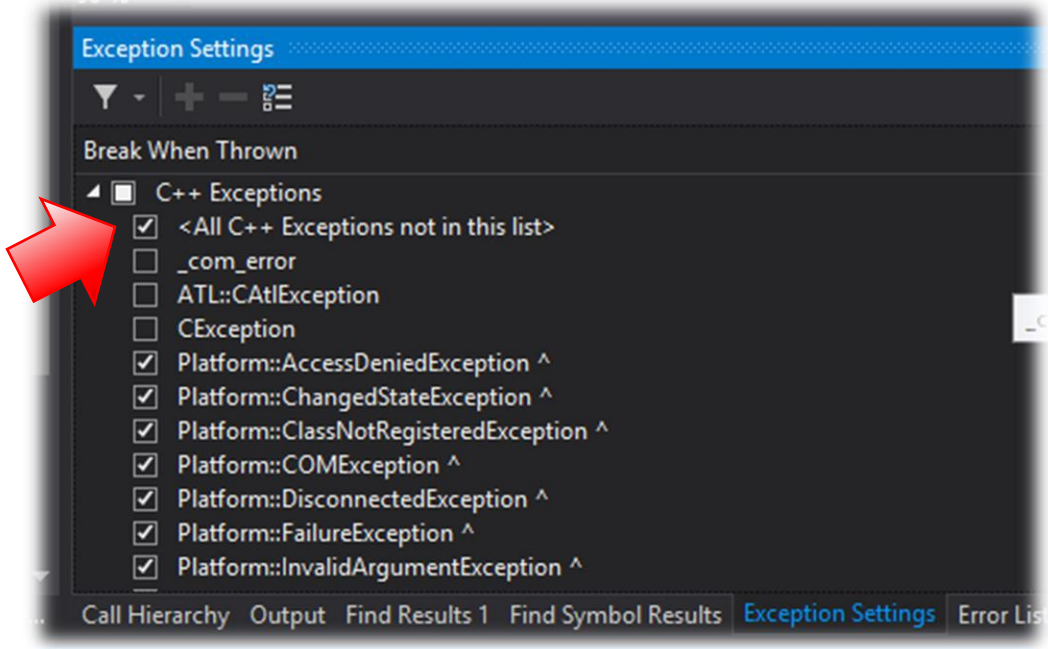
2a) Check, notify and throw (Visual Studio)

- Don‘t waste performance in Release build

```
#ifdef _DEBUG
#define GL_SAFE_CALL(A) ([&]                    \
{                                               \
    struct error_guard                          \
    {                                           \
        ~error_guard() noexcept(false)          \
        {                                       \
            GLenum error = glGetError();        \
            if (error != GL_NO_ERROR)           \
            {                                   \
                GLException ex(error);          \
                std::cerr << ex.what();         \
                throw ex;                       \
            }                                   \
        }                                       \
    } guard;                                    \
    return A;                                   \
}())
#else
#define GL_SAFE_CALL(A) (A)
#endif
```

3) Break on exception

- Visual Studio: Debug → Windows → Exception Settings
- „All C++ Exceptions not in this list“



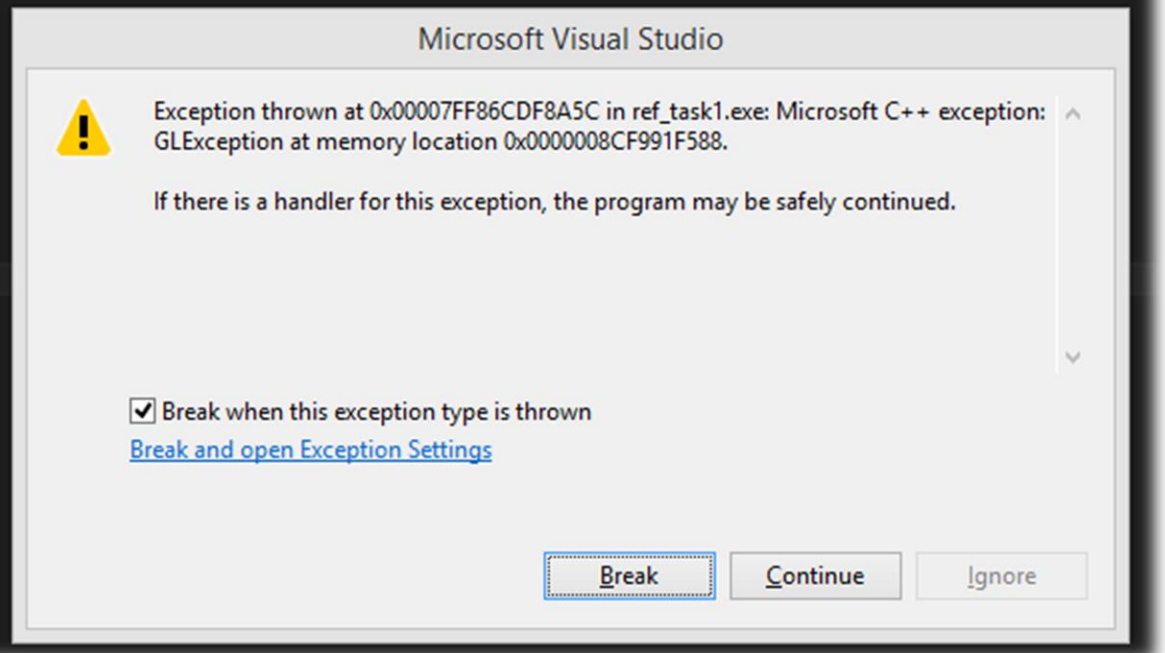
4) Make code safe

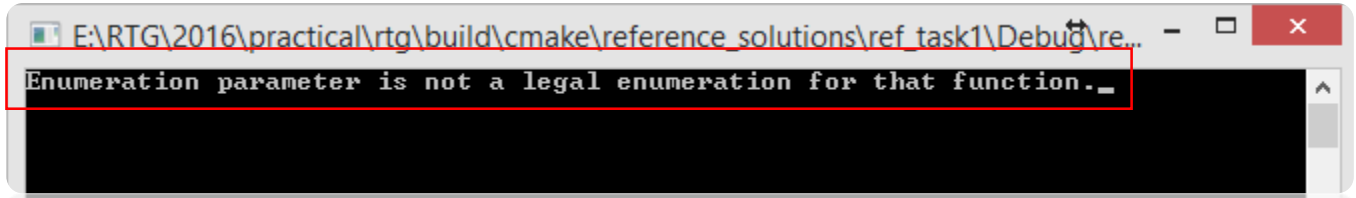
- Or even better, write safe code from the start!

```
void Renderer::render()
{
    GL_SAFE_CALL(glClear(GL_COLOR_BUFFER_BIT));

    GL_SAFE_CALL(glEnable(GL_DEPTH));

    GL_SAFE_CALL(glViewport(0, 0, viewport_width, viewport_height));

    GL_SAFE_CALL(glUseProgram(prog));
    GL_SAFE_CALL(glBindVertexArray(vao));
    GL_SAFE_CALL(glDrawArrays(GL_TRIANGLES, 0, 3));

    swapBuffers();
}
```

5) Debugger breaks on error and outputs message

```
77  void Renderer::resize(int width, int height)
78  {
79      viewport_width = width;
80      viewport_height = height;
81  }
82
83  void Renderer::render()
84  {
85      GL_SAFE_CALL(glClear(GL_COLOR_BUFFER_BIT));
86
87      GL_SAFE_CALL(glEnable(GL_DEPTH));
88
89      GL_SAFE_CALL(glViewport(0, 0, viewport_width, viewport_height));
90
91      GL_SAFE_CALL(glUseProgram(prog));
92      GL_SAFE_CALL(glBindVertexArray(vao));
93      GL_SAFE_CALL(glDrawArrays(GL_TRIANGLES, 0, 3));
94
95      swapBuffers();
96  }
97
```





6) Re-check what you were trying to do

- Usually involves looking up reference manual

https://www.opengl.org/sdk/docs/man4/html/glEnable.xhtml

Name

glEnable — enable or disable server-side GL capabilities

C Specification

void **glEnable**(GLenum *cap*);

GL_DEPTH_TEST

If enabled, do depth comparisons and update the depth buffer. Note that even if the depth buffer exists and the depth mas
glDepthRange.

GL_DITHER

If enabled, dither color components or indices before they are written to the color buffer.

GL_DEPTH instead of GL_DEPTH_TEST

```
void Renderer::render()
{
    GL_SAFE_CALL(glClear(GL_COLOR_BUFFER_BIT));

    GL_SAFE_CALL(glEnable(GL_DEPTH_TEST));

    GL_SAFE_CALL(glViewport(0, 0, viewport_width, viewport_height));

    GL_SAFE_CALL(glUseProgram(prog));
    GL_SAFE_CALL(glBindVertexArray(vao));
    GL_SAFE_CALL(glDrawArrays(GL_TRIANGLES, 0, 3));

    swapBuffers();
}
```

Fixed!

Can also be used for return values, conditionals…

```
program = GL_SAFE_CALL(glCreateProgram());
```

- Auto-completion still works this way
- Feel free to write more elegant error handling with templates if you can handle them!

Check shaders 1 / 2

- Write exception, throw when necessary

```
class ShaderException : public std::exception
{
private:
    std::string log;

public:
    ShaderException(std::string&& log) : log(std::move(log)) {}

    virtual const char* what() const noexcept
    {
        return log.c_str();
    }
};
```

Check shaders 2 / 2

```
void checkShader(const GLint shader)
{
    GLint isCompiled;
    GL_SAFE_CALL(glGetShaderiv(shader, GL_COMPILE_STATUS, &isCompiled));

    if (isCompiled != GL_TRUE)
    {
        GLint log_length;
        GL_SAFE_CALL(glGetShaderiv(shader, GL_INFO_LOG_LENGTH, &log_length));

        auto buffer = std::unique_ptr<char[]>(new char[log_length]);
        GL_SAFE_CALL(glGetShaderInfoLog(shader, log_length, &log_length, buffer.get()));

        ShaderException ex(std::string(buffer.get(), log_length - 1));
        std::cerr << ex.what();
        throw ex;
    }
}
```

Check programs 1 / 2

- Déjà-vu

```
class ProgramException : public std::exception
{
private:
    std::string log;

public:
    ProgramException(std::string&& log) : log(std::move(log)) {}

    virtual const char* what() const noexcept
    {
        return log.c_str();
    }
};
```

Check programs 2 / 2

```
void checkProgram(const GLint program)
{
    GLint isLinked;
    GL_SAFE_CALL(glGetProgramiv(program, GL_LINK_STATUS, &isLinked));

    if (isLinked != GL_TRUE)
    {
        GLint log_length;
        GL_SAFE_CALL(glGetProgramiv(program, GL_INFO_LOG_LENGTH, &log_length));

        auto buffer = std::unique_ptr<char[]>(new char[log_length]);
        GL_SAFE_CALL(glGetProgramInfoLog(program, log_length, &log_length, buffer.get()));

        ProgramException ex(std::string(buffer.get(), log_length - 1));
        std::cerr << ex.what();
        throw ex;
    }
}
```

Example: Checking a shader

- Recommendation: use raw string literals for your shaders (no file loading needed)

```
const char* shader_src =
R"""(

<shader code goes here>

)""";
```

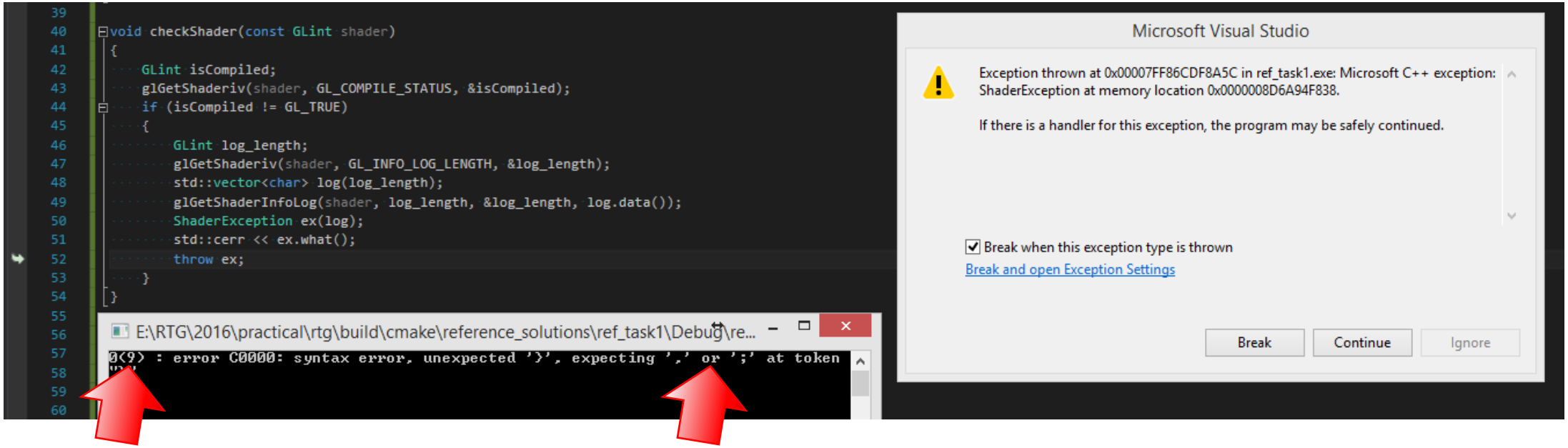
Why is my screen blank?

```
const char* fragment_shader_src =
R"""(
#version 430

layout(location = 0) out vec4 target;

void main()
{
    target = vec4(1.0f, 1.0f, 1.0f, 1.0f)
}
)""";
```

Checking the shader and the output

```
GLint fs = GL_SAFE_CALL(glCreateShader(GL_FRAGMENT_SHADER));
GL_SAFE_CALL(glShaderSource(fs, 1, &fragment_shader_src, 0));
GL_SAFE_CALL(glCompileShader(fs));
checkShader(fs);
```

- Run



Line where error was **noticed (9)**
(not where it **occurred!**)

expecting `,` or `;`

There it is.

```
const char* fragment_shader_src =
R"""(
#version 430

layout(location = 0) out vec4 target;

void main()
{
   target = vec4(1.0f, 1.0f, 1.0f, 1.0f)
}
)""";
```

All fixed.

```
const char* fragment_shader_src =
R"""(
#version 430

layout(location = 0) out vec4 target;

void main()
{
   target = vec4(1.0f, 1.0f, 1.0f, 1.0f);
}
)""";
```

If you run into problems

- Is your harware ready for OpenGL 4.3? If not, we can help you out
- Use the above measures to pinpoint errors
- Consult reference manual
- If you send us code that obviously wasn‘t debugged (e.g. no GL_SAFE_CALL), it will be returned to sender