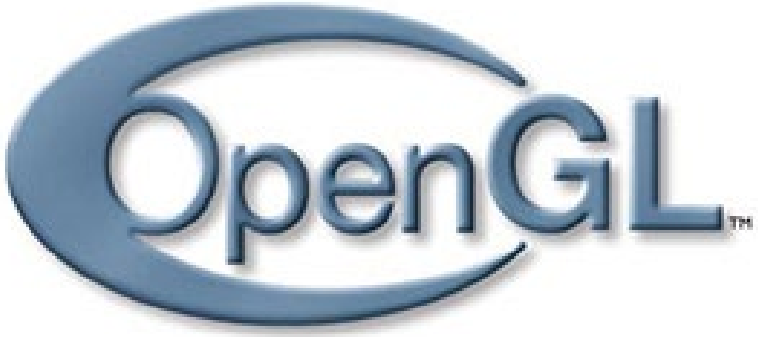
Computer Graphics
– Getting Started

Prof. Dr. habil. Kai Lawonn

OpenGL

What is OpenGL?

- OpenGL is mainly considered:
  - an API (an Application Programming Interface) that provides us with a large set of functions to manipulate graphics and images
  - OpenGL by itself is not an API, but merely a specification, developed and maintained by the Khronos Group



https://www.khronos.org/

What is OpenGL?

- OpenGL specification specifies result/output of each function
- It is up to the developers implementing this specification
- OpenGL specification does not give implementation details
- Developer of OpenGL libraries are usually the graphics card manufacturers
- Each graphics card supports specific versions of OpenGL
- On Apple systems the OpenGL library is maintained by Apple themselves
- On Linux there exists a combination of graphic suppliers' versions and hobbyists' adaptations of these libraries
- This means whenever OpenGL is showing weird behavior that it shouldn't, most likely the fault of the graphics cards manufacturers or whoever developed/maintained the library

Bug?

Whenever there is a bug in the implementation this is usually solved by updating your video card drivers!

Versions (Selection of Extensions)

- OpenGL:
  - **1.0** (1992)
  - 1.1 (1996): Texture objects, Vertex arrays
  - 1.2 (1998): 3D Textures, GL_ARB_imaging (2D image processing)
  - 1.3 (2001): Image extensions
  - 1.4 (2002): Depth textures
  - 1.5 (2003): Vertex buffer objects
  - **2.0** (2004): Shader objects, Multiple render targets, Non-power-of-two textures
  - 2.1 (2006): Pixel buffer objects, sRGB textures
  - **3.0** (2008): Framebuffer objects
  - 3.1 (2009): Uniform Buffer Objects
  - 3.2 (2009): Geometry shaders
  - 3.3 (2010): Dual-source blending

Versions (Selection of Extensions)

- OpenGL:
  - **4.0** (2010): Tessellation, Transform feedback
  - 4.1 (2010): 64-bit floating-point component vertex attributes
  - 4.2 (2011): Atomic counters, Shader image load store
  - 4.3 (2012): Compute shaders
  - 4.4 (2013): Shader storage buffers
  - 4.5 (2015): Derivative control
  - 4.6 (2017): Anisotropic filtering

State Machine

- OpenGL is a state machine: a collection of variables that define how it currently operates
- The state is referred to as the OpenGL context
- Change state by setting options, manipulating buffers and then render using the current context
- E.g., set the line width it will remain until we change the context variable
- Several state-changing functions that change the context and state-using functions that perform operations based on the current state

Objects

- OpenGL libraries are written in C
- Many of C's language-constructs do not translate that well to other higher-level languages -> OpenGL was developed with several abstractions in mind
- One of those abstractions are objects in OpenGL

Objects

- An object in OpenGL is a collection of options that represents a subset of OpenGL's state
- E.g., object that represents the settings of the drawing window (size, how many colors it supports etc.)

Objects

- Visualize an object as a C-like struct:

```
struct object_name {
        float option1;
        int option2;
        char[] name;
    };
```

Objects

- Objects generally looks something like this (OpenGL's context):

```
// The State of OpenGL
struct OpenGL_Context {
    ...
  object* object_Window_Target;
    ...
};
```

```
// create object
unsigned int objectId = 0;
glGenObject(1, &objectId);
// bind object to context
glBindObject(GL_WINDOW_TARGET, objectId);
// set options of object currently bound to
// GL_WINDOW_TARGET
glSetObjectOption(GL_WINDOW_TARGET,
GL_OPTION_WINDOW_WIDTH, 800);
glSetObjectOption(GL_WINDOW_TARGET,
GL_OPTION_WINDOW_HEIGHT, 600);
// set context target back to default
glBindObject(GL_WINDOW_TARGET, 0);
```

Objects

- This code you will frequently see in OpenGL
- First create an object and store a reference to it as an id (the real object data is stored behind the scenes)
- Then bind the object to the target location of the context (the location of the example window object target is defined as GL_WINDOW_TARGET)

```
// create object
unsigned int objectId = 0;
glGenObject(1, &objectId);
// bind object to context
glBindObject(GL_WINDOW_TARGET, objectId);
// set options of object currently bound to
// GL_WINDOW_TARGET
glSetObjectOption(GL_WINDOW_TARGET,
GL_OPTION_WINDOW_WIDTH, 800);
glSetObjectOption(GL_WINDOW_TARGET,
GL_OPTION_WINDOW_HEIGHT, 600);
// set context target back to default
glBindObject(GL_WINDOW_TARGET, 0);
```

Objects

- Next set the window options and finally we un-bind the object by setting the current object id of the window target to 0
- The options are stored in the object referenced by objectId and restored as soon as we bind the object back to GL_WINDOW_TARGET

```
// create object
unsigned int objectId = 0;
glGenObject(1, &objectId);
// bind object to context
glBindObject(GL_WINDOW_TARGET, objectId);
// set options of object currently bound to
// GL_WINDOW_TARGET
glSetObjectOption(GL_WINDOW_TARGET,
GL_OPTION_WINDOW_WIDTH, 800);
glSetObjectOption(GL_WINDOW_TARGET,
GL_OPTION_WINDOW_HEIGHT, 600);
// set context target back to default
glBindObject(GL_WINDOW_TARGET, 0);
```

Note

The code samples provided so far are only approximations of how OpenGL operates; throughout the lectures you will come across enough actual examples

Objects

- The great thing: can define more than one object in our application:
  - set their options and whenever we start an operation that uses OpenGL's state, we bind the object with our preferred settings
- E.g., objects act as container objects for 3D model data:
  - (a house or a character) whenever want to draw one of them: bind the object containing the model

Run the Examples

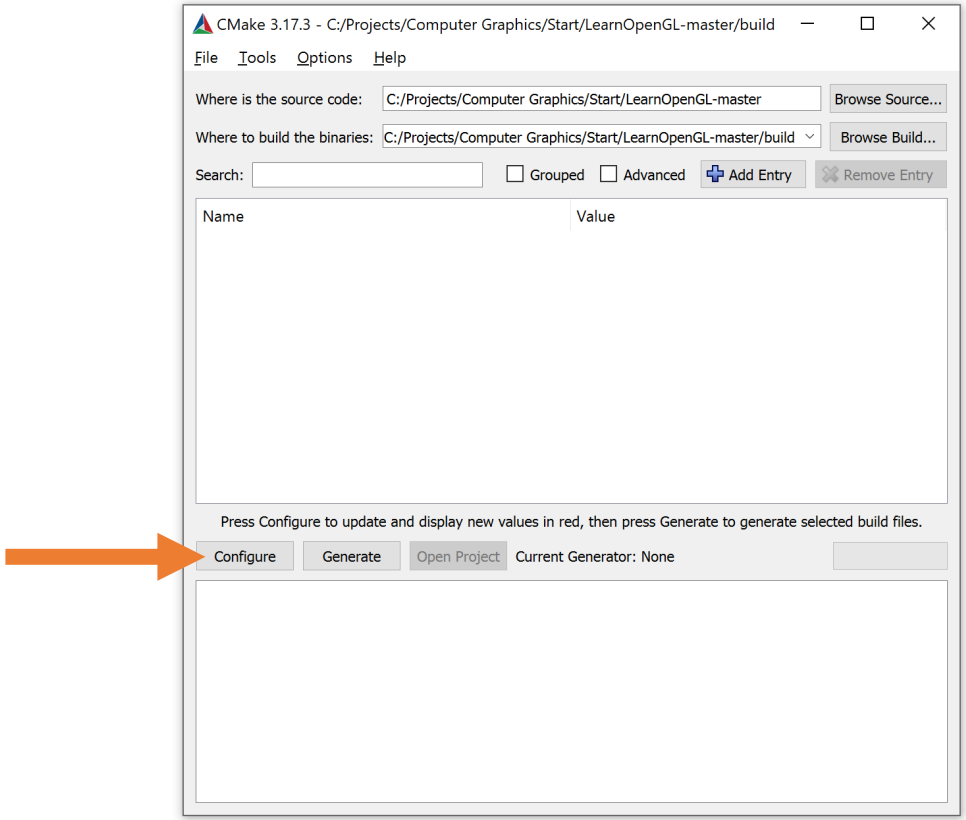
Start

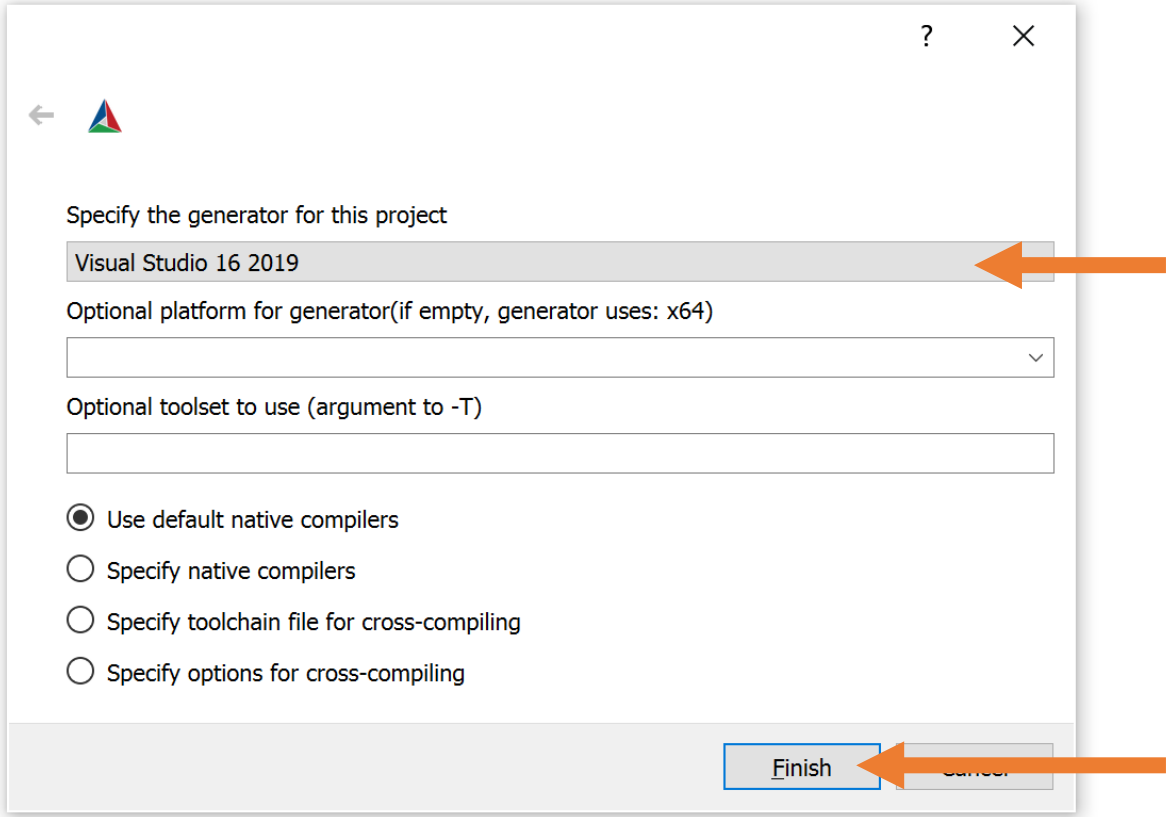
- Start the examples:
- 1. Install Visual Studio Community (free IDE)
- 2. Install CMake
- 3. Download the repository

1. https://visualstudio.microsoft.com/de/vs/community/
2. https://cmake.org/
3. https://github.com/JoeyDeVries/LearnOpenGL

Start

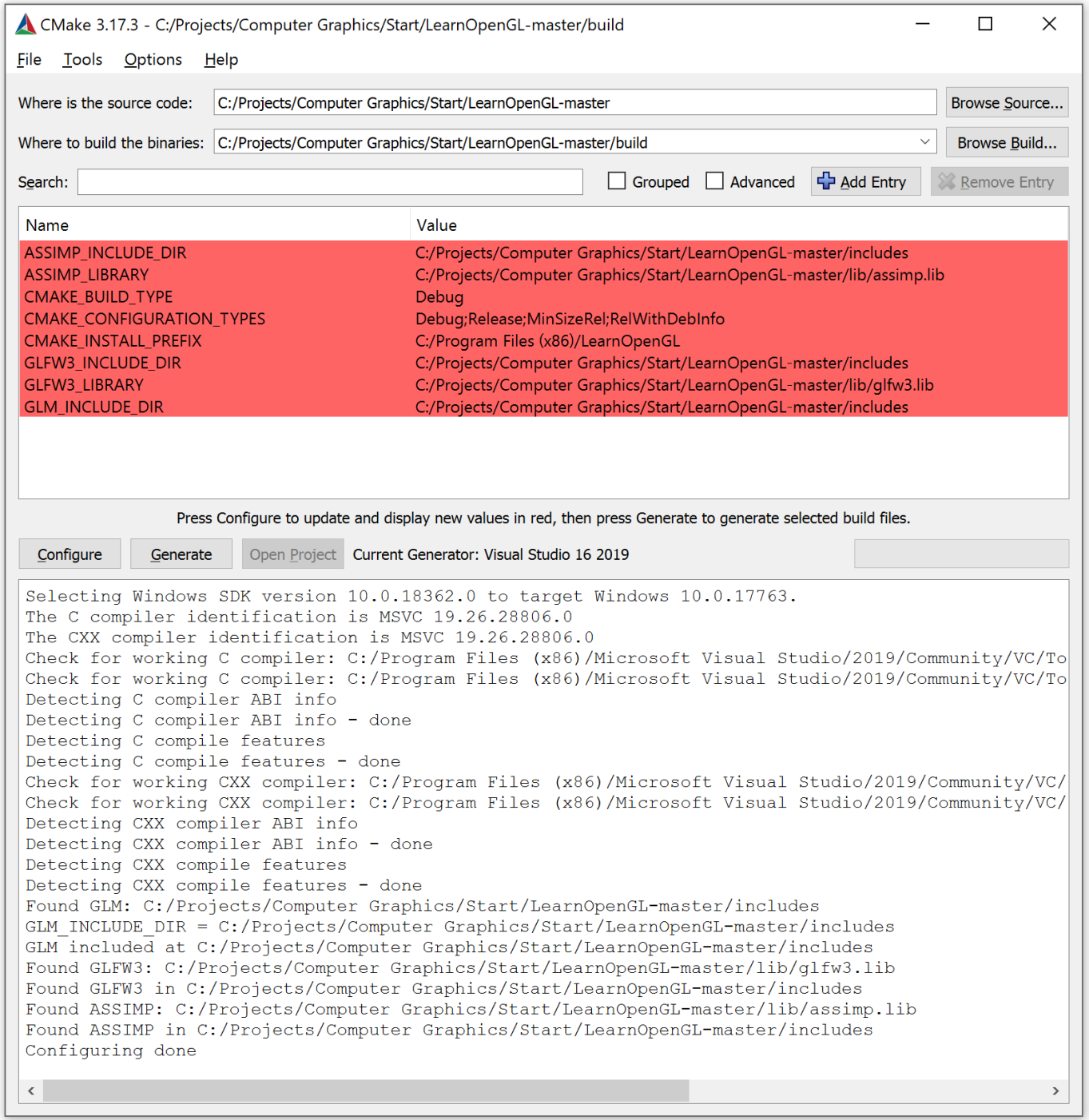
- After installation of Visual Studio, CMake and the download of the repository:





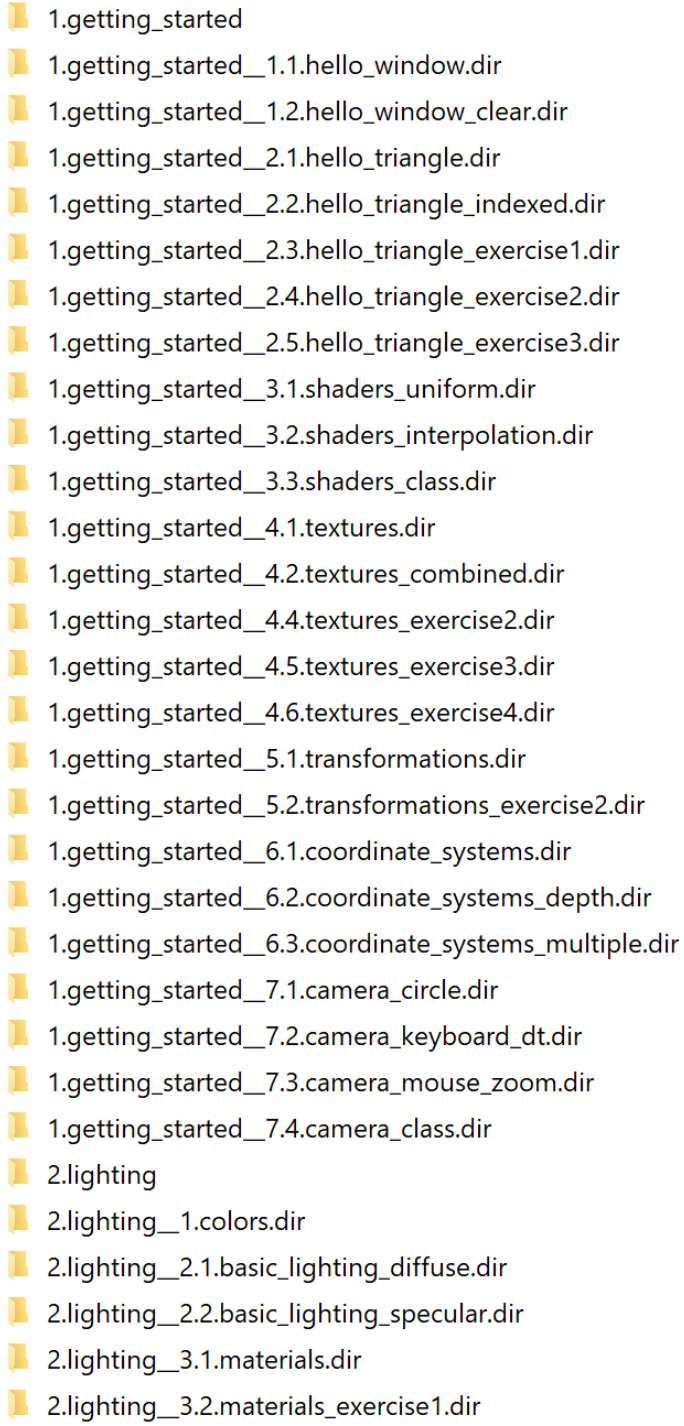
Start

- ‚Generate‘ afterwards



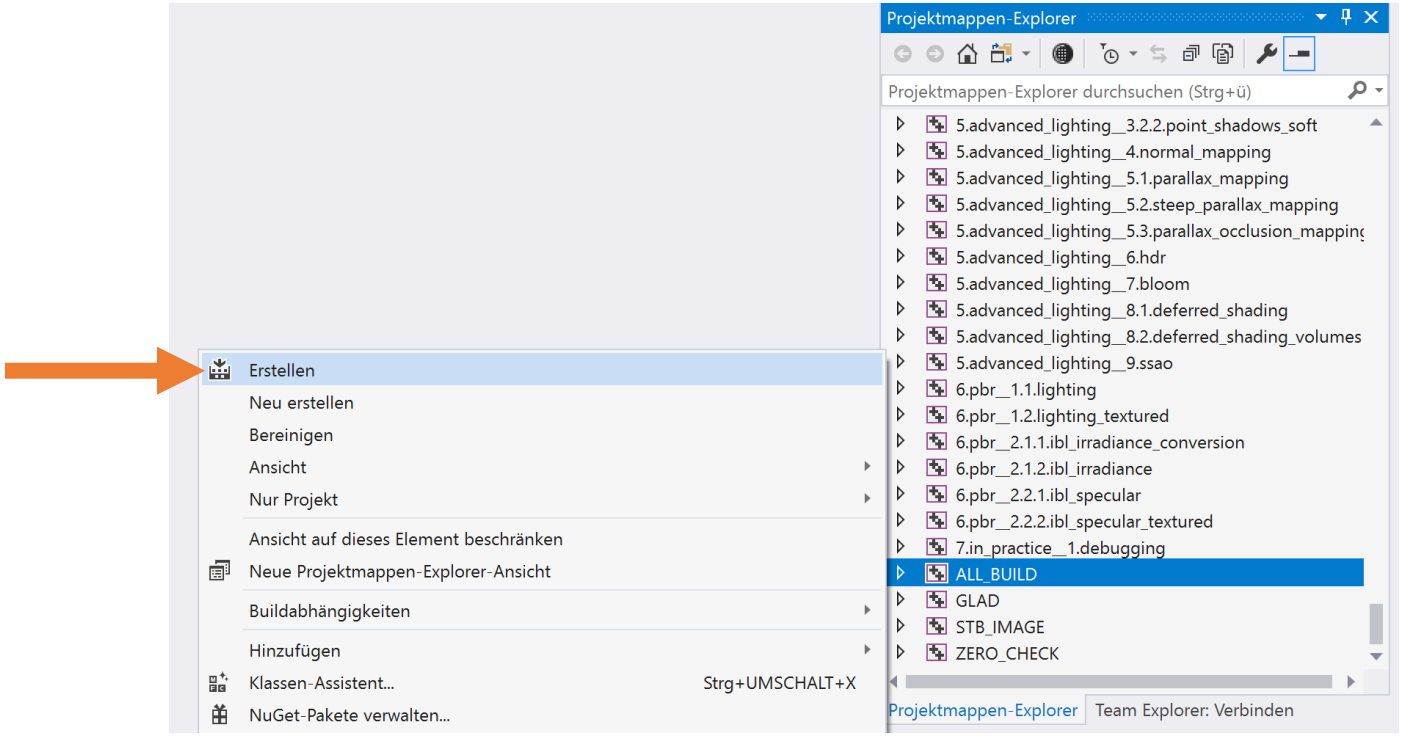
Start

- In the build folder:
- Open the solution:
  - (LearnOpenGL.sln)



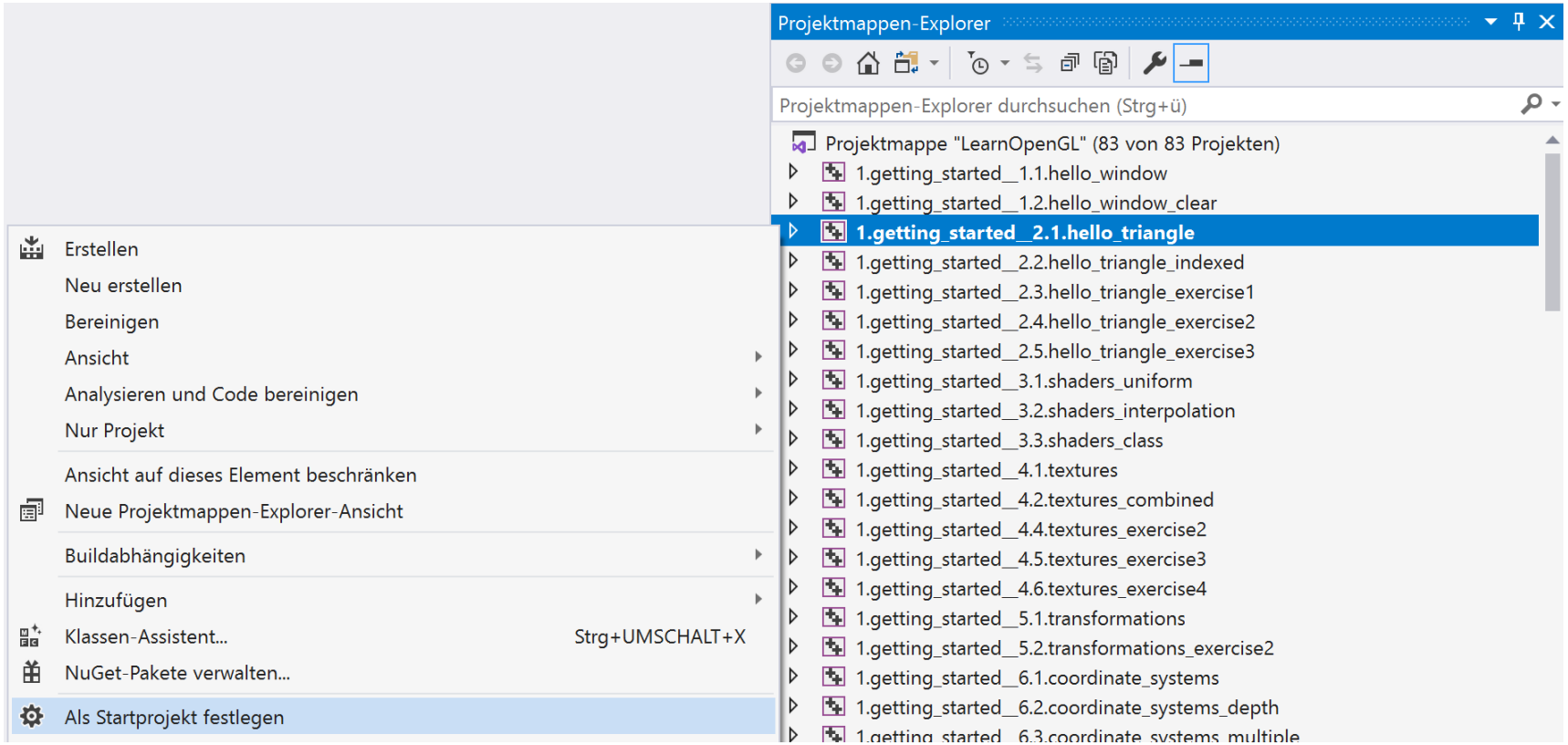
Start

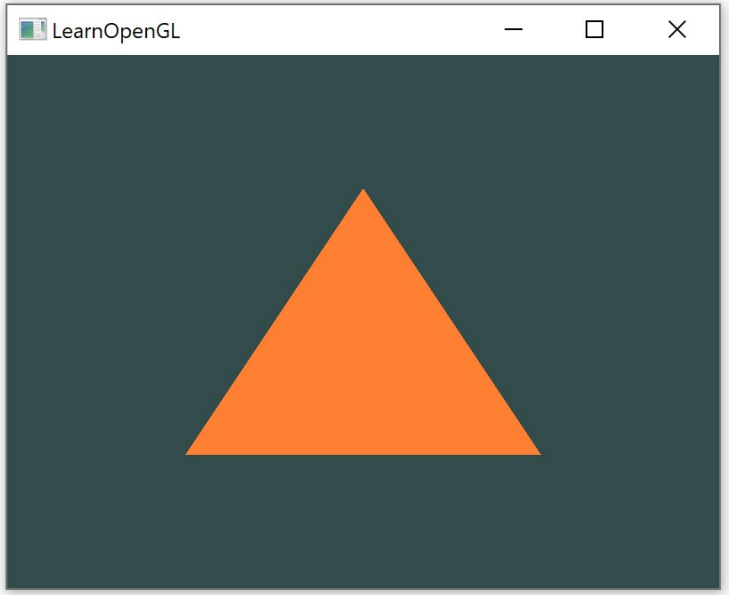
- Visual Studio:
  - Right click ALL_BUILD
  - Build/Erstellen
- …Wait….



Start

- Select an example -> right click -> Start Project -> F5

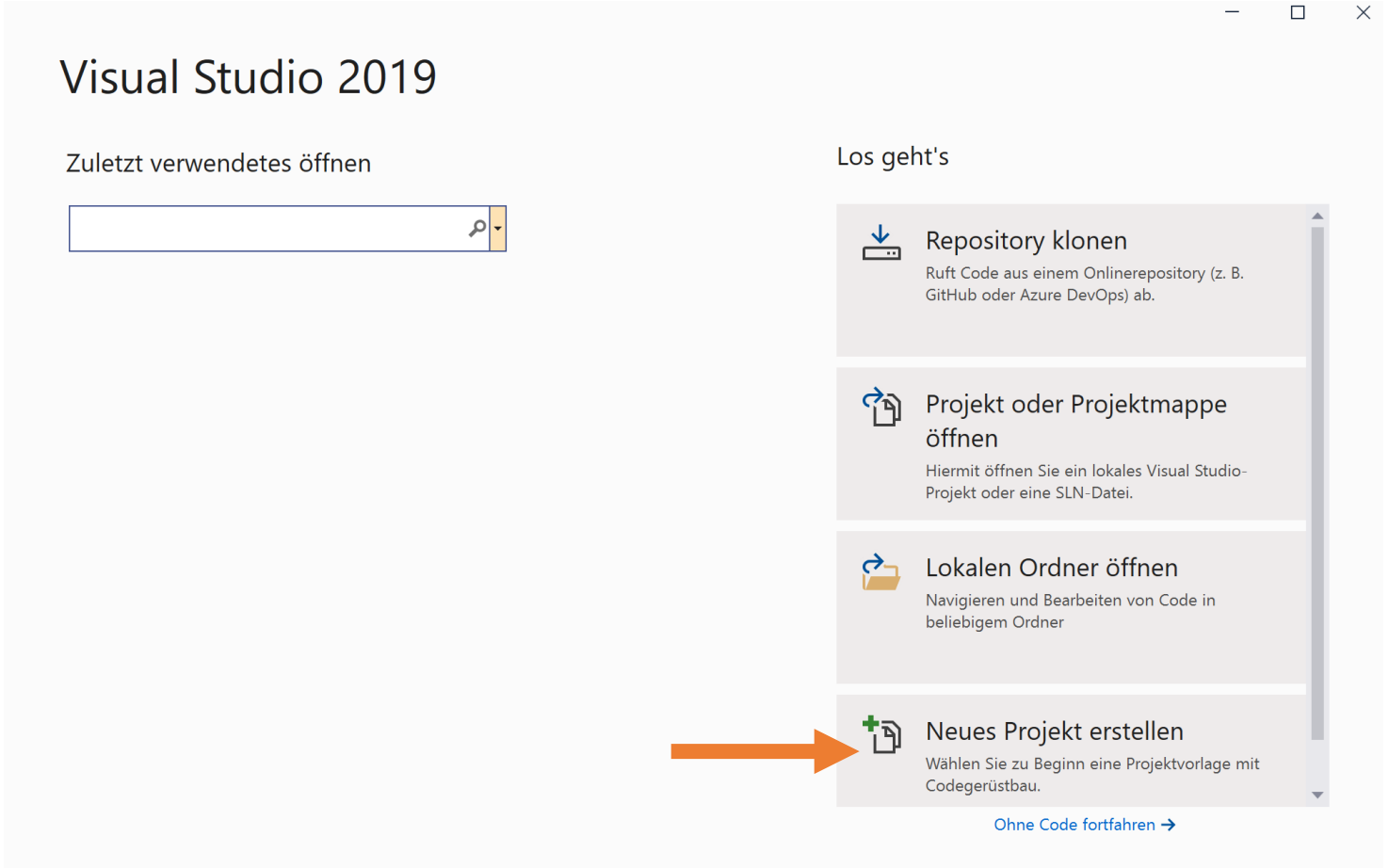




C++*

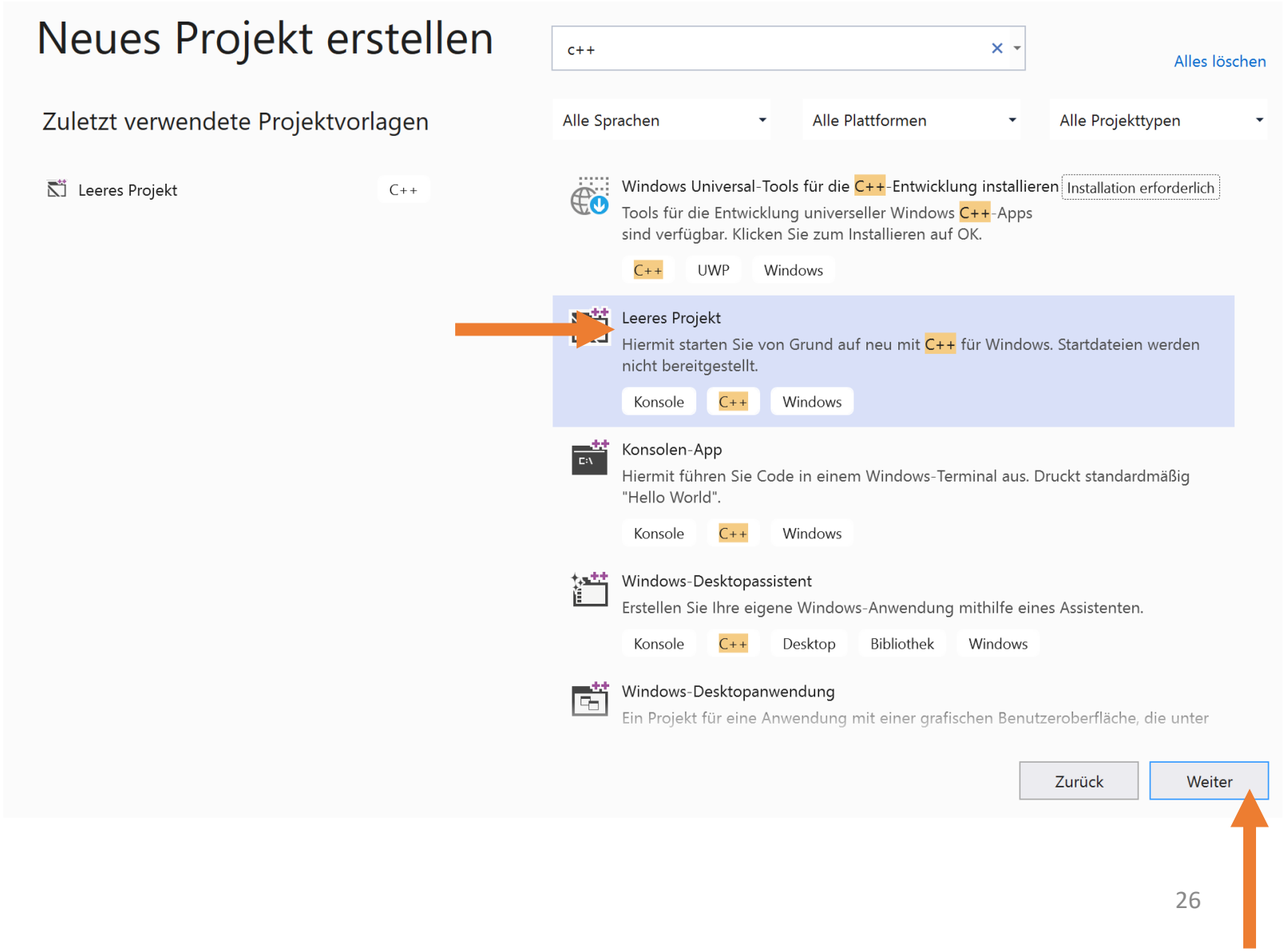
First Project

- Open visual studio and create a new project



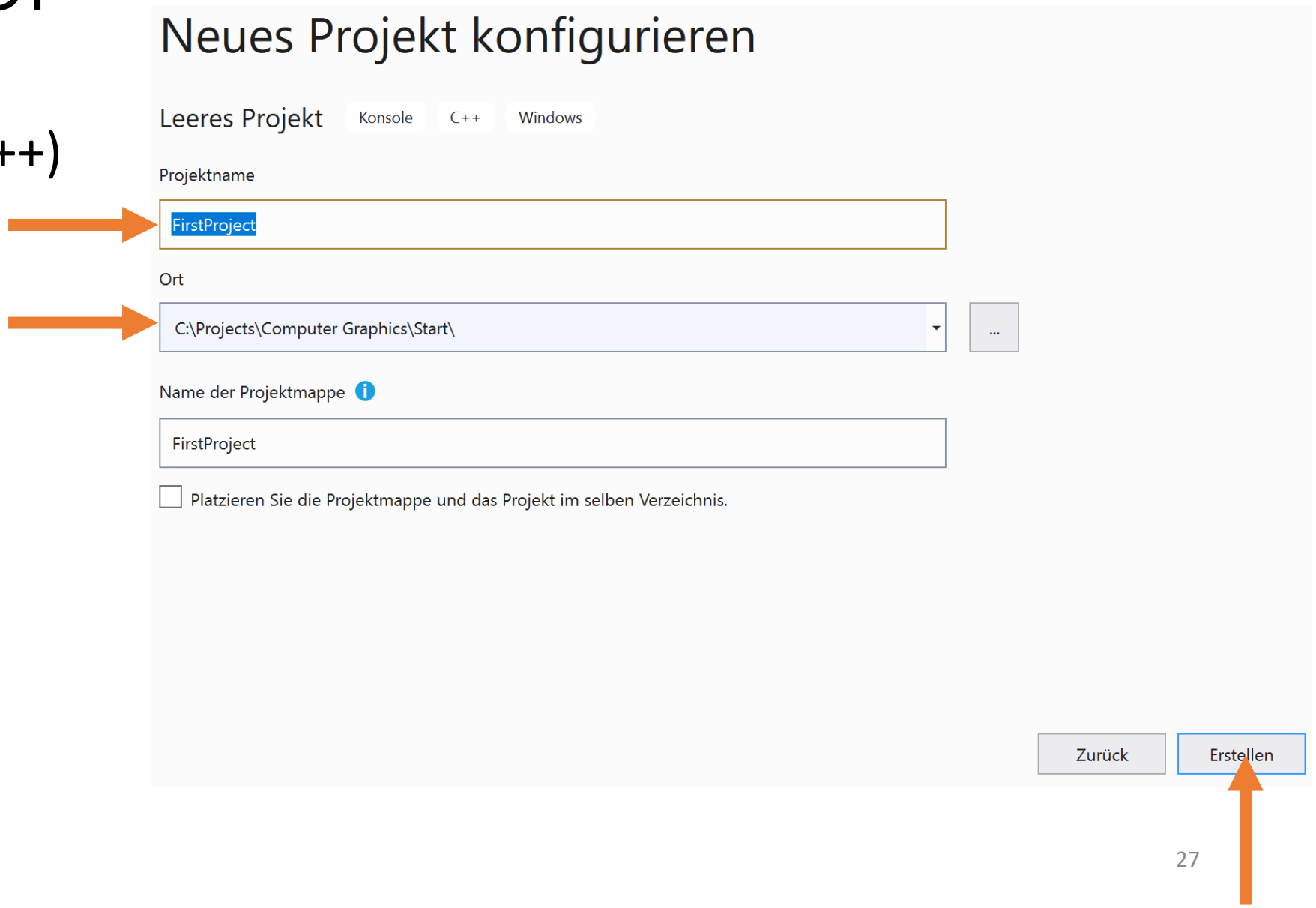
First Project

- Empty project (C++)



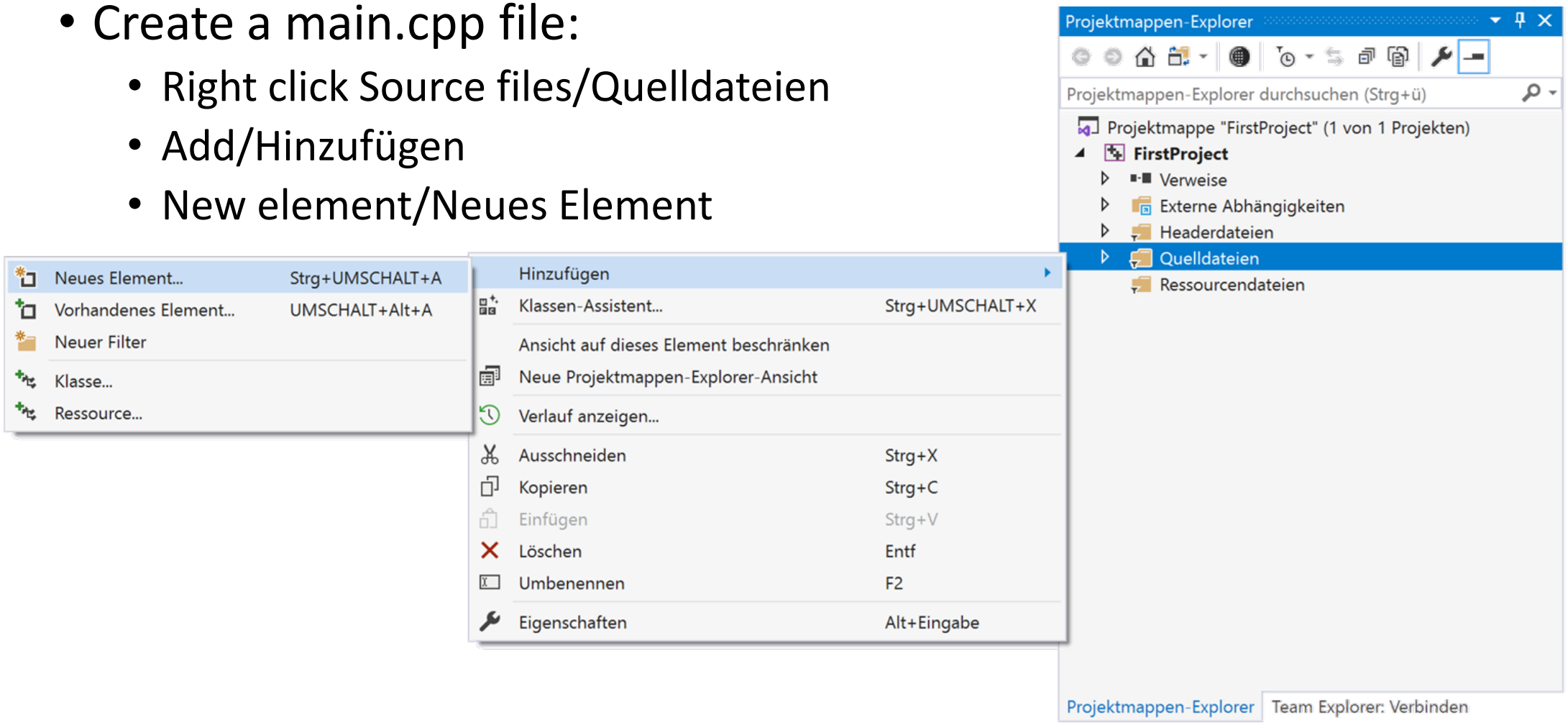
First Project

- Empty project (C++)



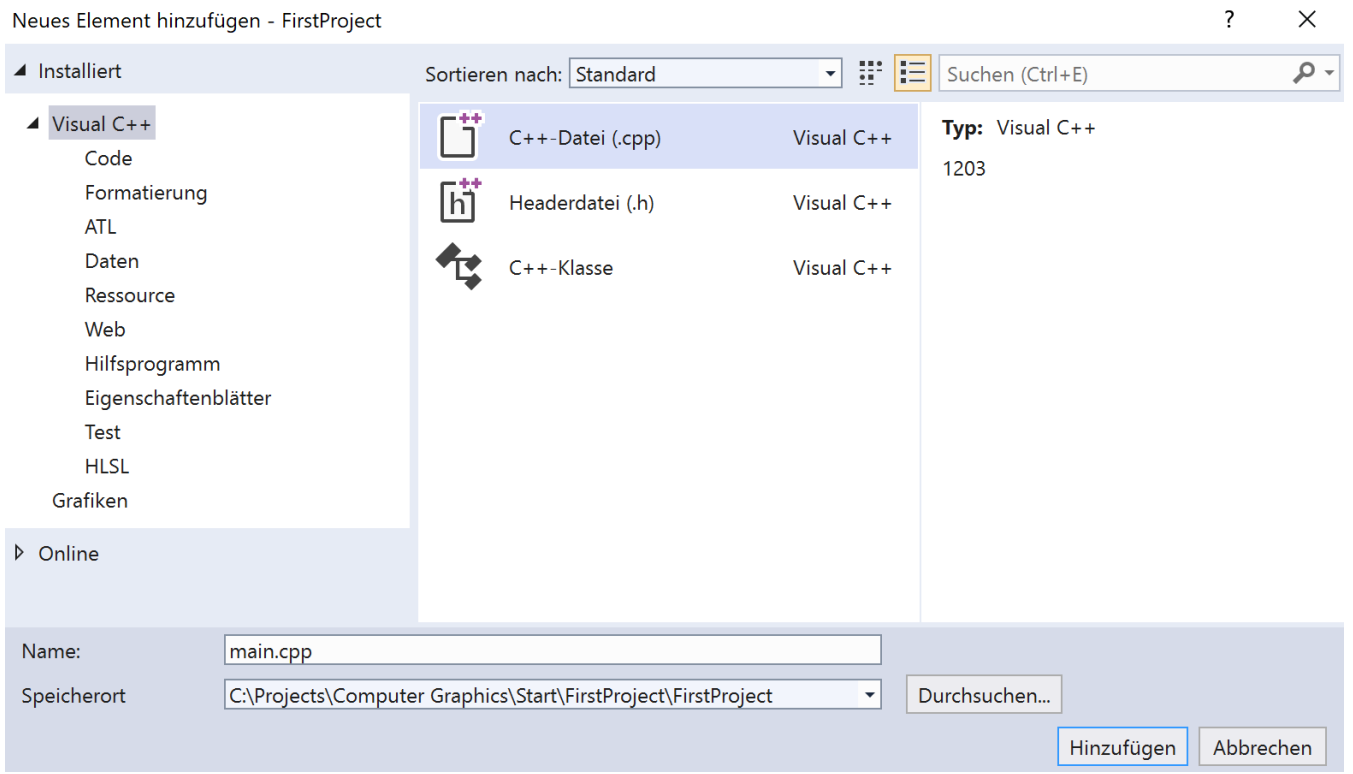
Hello World

- Create a main.cpp file:
  - Right click Source files/Quelldateien
  - Add/Hinzufügen
  - New element/Neues Element



Hello World

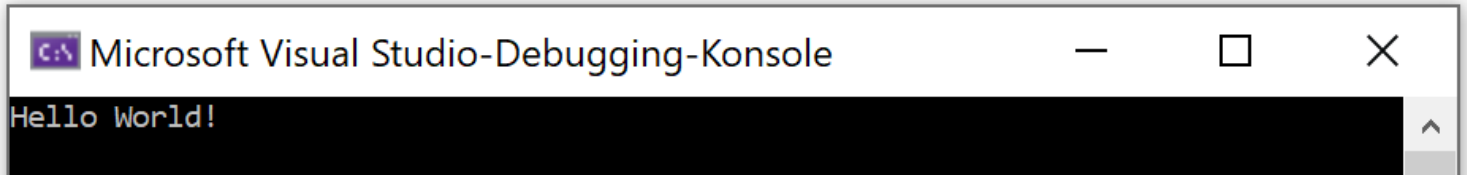
- Name it main.cpp and add it



Hello World

- Open main.cpp and paste and compile (F5):

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello World!";
    return 0;
}
```



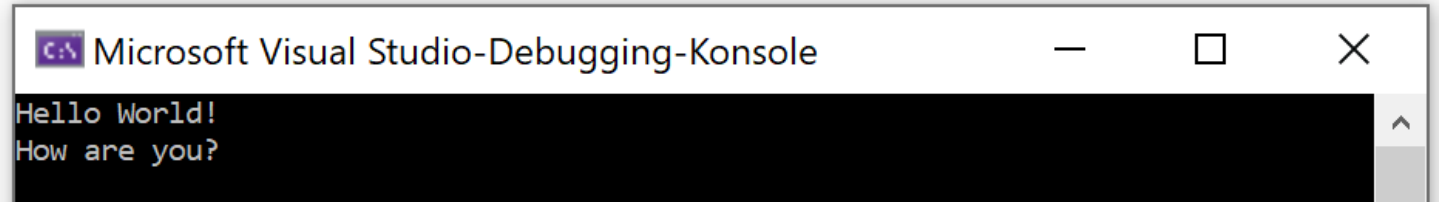
Hello World

```
#include <iostream>
//include file input-output stream: header file contains
//definitions to objects like cin, cout, cerr etc


int main() //first function that is called
{
   std::cout << "Hello World!"; //print text with cout function from iostream
   return 0; //return a value as “int main()” expects a return value
}
```

C++

- Text over several lines (new line „\n“):

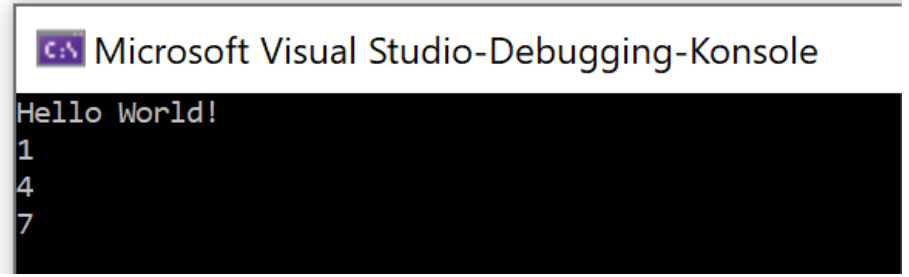
```
#include <iostream>

int main()
{
   std::cout << "Hello World!\n";
   std::cout << "How are you?\n";
   return 0;
}
```



C++

- Comments per line: „//“ over a block „/* */“

```
#include <iostream>

int main()
{
    std::cout << "Hello World!\n";
    std::cout << "1\n";
    //std::cout << "2\n";
    //std::cout << "3\n";
    std::cout << "4\n";
    /*std::cout << "5\n";
    std::cout << "6\n";*/
    std::cout << "7\n";
    return 0;
}
```



C++

- Define variables

```
#include <iostream>

int main()
{
   int a, b=5;
   a = 4;
   std::cout << "a+b=" << a+b <<"\n";
   return 0;
}
```

```
a+b=9
```

C++

- Input

```cpp
#include <iostream>

int main()
{
   int a, b;
   std::cout << "Enter first number: ";
   std::cin >> a;
   std::cout << "Enter second number: ";
   std::cin >> b;
   std::cout<<"a+b="<<a+b<<"\n";
   return 0;
}
```

```
Enter first number: 4
Enter second number: 7
a+b=11
```

C++

- Functions with return value

```
#include <iostream>

int square(int a)
{
   return a * a;
}

int main()
{
   int a;
   std::cout << "Enter first number: ";
   std::cin >> a;
   std::cout<<"a*a="<<square(a)<<"\n";
   return 0;
}
```

```
Enter first number: 7
a*a=49
```

C++

- Functions with return value

Error!

```
#include <iostream>

int main()
{
   int a;
   std::cout << "Enter first number: ";
   std::cin >> a;
   std::cout<<"a*a="<<square(a)<<"\n";
   return 0;
}

int square(int a)
{
   return a * a;
}
```

Correct!

```
#include <iostream>
int square(int a);
int main()
{
   int a;
   std::cout << "Enter first number: ";
   std::cin >> a;
   std::cout<<"a*a="<<square(a)<<"\n";
   return 0;
}

int square(int a)
{
   return a * a;
}
```

C++

- Functions with no return value

```
#include <iostream>

void square(int a)
{
        std::cout << "a*a = " << a * a << "\n";
}

int main()
{
    int a;
    std::cout << "Enter first number: ";
    std::cin >> a;
    square(a);
    return 0;
}
```

```
Enter first number: 7
a*a = 49
```

C++

- Functions with two arguments and no return value:

```
#include <iostream>
void multiply(int a, int b)
{
   std::cout << a <<" * " << b << " = " << a * b << "\n";
}

int main()
{
    int a,b;
    std::cout << "Enter two numbers:\n";
    std::cin >> a;
    std::cin >> b;
    multiply(a,b);
    return 0;
}
```

```
Enter two numbers:
4
7
4 * 7 = 28
```

C++

- Increment/Decrement operations:

```
#include <iostream>

int main()
{
int a=5,b;
b = ++a; // a is incremented to 6, a is evaluated to the
//value 6, and 6 is assigned to b
std::cout << a << " " << b <<"\n";

int c = 5, d;
d = c++; // c is incremented to 6, copy of original c is
//evaluated to the value 5, and 5 is assigned to d
std::cout << c << " " << d;
return 0;
}
```

```
6 6
6 5
```

C++

- Increment/Decrement operations:

```cpp
#include <iostream>

int main()
{
    int x=5;
    int y=5;
    std::cout << x << " " << y << '\n';
    std::cout << ++x << " " << --y << '\n'; // prefix
    std::cout << x << " " << y << '\n';
    std::cout << x++ << " " << y-- << '\n'; // postfix
    std::cout << x << " " << y << '\n';

    return 0;
}
```

```
5 5
6 4
6 4
6 4
7 3
```

C++

- If/else statements:

```
#include <iostream>

int main()
{
    int x, y;
    std::cin >> x;
    std::cin >> y;
    if (x >= y)
    {
        std::cout << x;
    }
    else
    {
        std::cout << y;
    }
    return 0;
}
```

```
3
5
5
```

C++

- Relational operators:

```
#include <iostream>

int main()
{
    int x,y;
    std::cout << "Enter an integer: ";
    std::cin >> x;

    std::cout << "Enter another integer: ";
    std::cin >> y;

    if (x == y)
        std::cout << x << " equals " << y << '\n';
    if (x != y)
        std::cout << x << " does not equal " << y << '\n';
    if (x > y)
        std::cout << x << " is greater than " << y << '\n';
    if (x < y)
        std::cout << x << " is less than " << y << '\n';
    if (x >= y)
        std::cout << x << " is greater than or equal to " << y << '\n';
    if (x <= y)
        std::cout << x << " is less than or equal to " << y << '\n';

    return 0;
}
```

```
Enter an integer: 6
Enter another integer: 8
6 does not equal 8
6 is less than 8
6 is less than or equal to 8
```

C++

- Logical operators (! = logical not, && = logical and, || = logical or):

```
#include <iostream>

int main()
{
    int x = 5, y = 4;

    if (!(x == y))
        std::cout << x << " does not equal " << y << '\n';
    if (x == 5 && y == 4)
        std::cout << x << " = 5 AND " << y << " = 4" <<'\n';
    if (x == 3 || y == 4)
        std::cout << x << " = 3 OR " << y << " = 4" << '\n';

    return 0;
}
```

```
5 does not equal 4
5 = 5 AND 4 = 4
5 = 3 OR 4 = 4
```

C++

- Arrays

```cpp
#include <iostream>

int main()
{
    double list[3]{}; // allocate 3 doubles
    list[0] = 1.0;
    list[1] = 2.0;
    list[2] = 3.3;

    std::cout << list[0] << " " << list[1] << " " << list[2] << '\n';

    return 0;
}
```

```
1 2 3.3
```

C++

- Structs:

```cpp
#include <iostream>

int main()
{
    struct human {
        double height;
        double weight;
    };
    human persons[2]{};
    persons[0].height = 1.93;
    persons[0].weight = 87.3;
    persons[1].height = 1.75;
    persons[1].weight = 71.0;
    std::cout << persons[0].height << " " << persons[1].weight << '\n';

    return 0;
}
```

```
1.93 71
```

C++

- Loops:

```
#include <iostream>

int main()
{
    double list[5]{1,2,3,4,5};

    for (int i = 0; i < 5; ++i)
    {
        std::cout << list[i] << "\n";
    }

    return 0;
}
```

```
1
2
3
4
5
```

C++

- Loops:

```
#include <iostream>

int main()
{
    int count = 0;
    while (count < 10)
    {
        std::cout << count << “\n”;
        ++count;
    }
    std::cout << "done!";

    return 0;
}
```

```
0
1
2
3
4
5
6
7
8
9
done!
```

C++

- Pointer ☹:

```
#include <iostream>

int main()
{
    int a=5;

    std::cout << a << "\n"; // print the value of a
    std::cout << &a << "\n";// print the memory address of a

    return 0;
}
```

```
5
009CFC78
```

C++

- Declaring a pointer:

```
#include <iostream>

int main()
{
  int* a; // define a pointer
  int b;

  a = &b; // a = address of b
  *a = 5; // value pointed to 5
  std::cout << b << "\n"; // print the value of b

  return 0;
}
```

```
5
```

C++

- References as function parameters:

```
#include <iostream>

void square(double &x)
{
    x = x * x;
    std::cout << x << "\n";
}

int main()
{
    double x = 2;
    square(x);
    square(x);
    square(x);

    return 0;
}
```

```
4
16
256
```

C++

- References as function parameters (avoid copies):

```
#include <iostream>

void print(const int& n)
{
    std::cout << n << "\n";
}

int main()
{
    int n = 2;
    print(n);

    return 0;
}
```

```
2
```

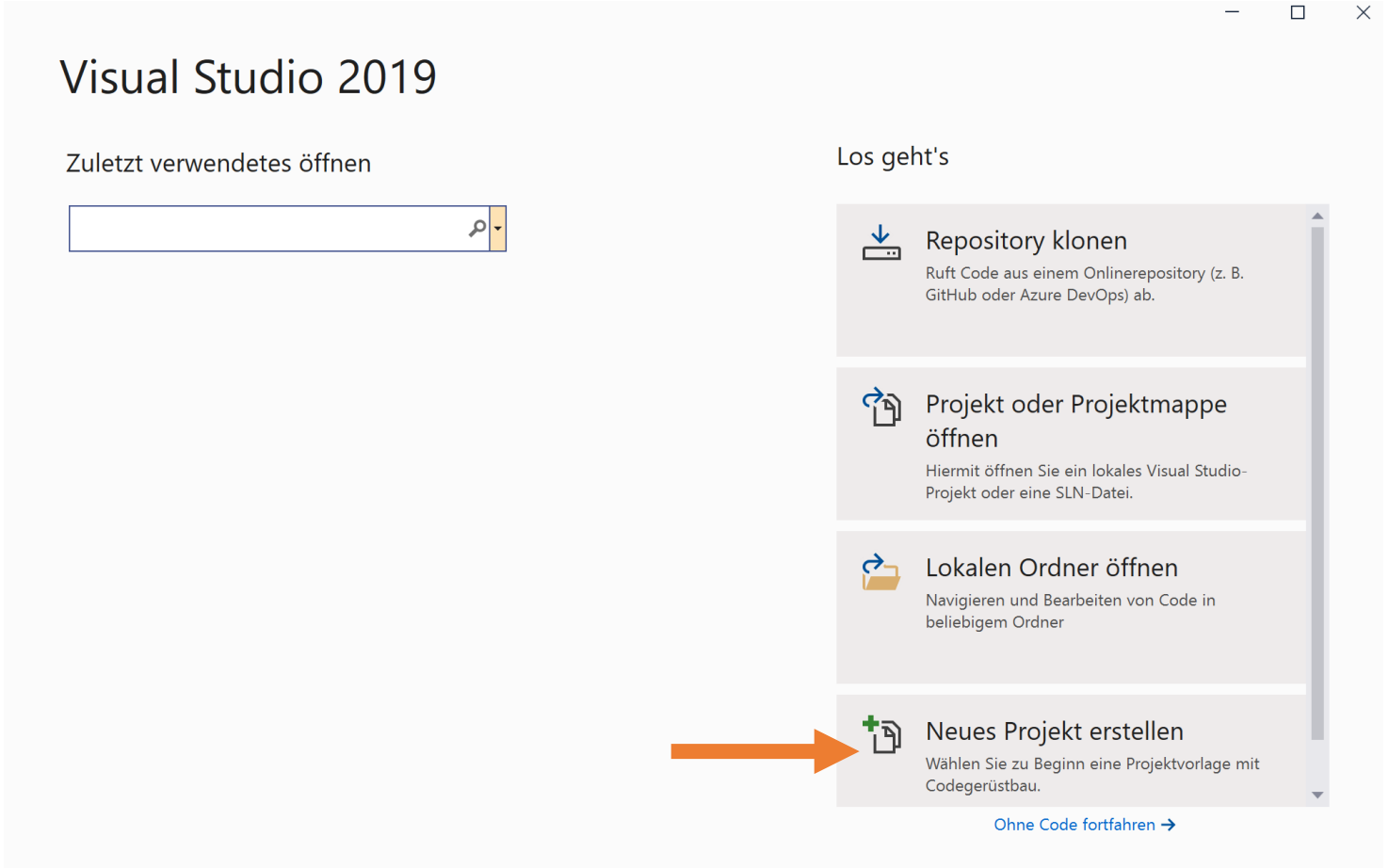
C++

- For more information on C++:
  - https://www.learncpp.com/
  - http://www.cplusplus.com/

Hello Window

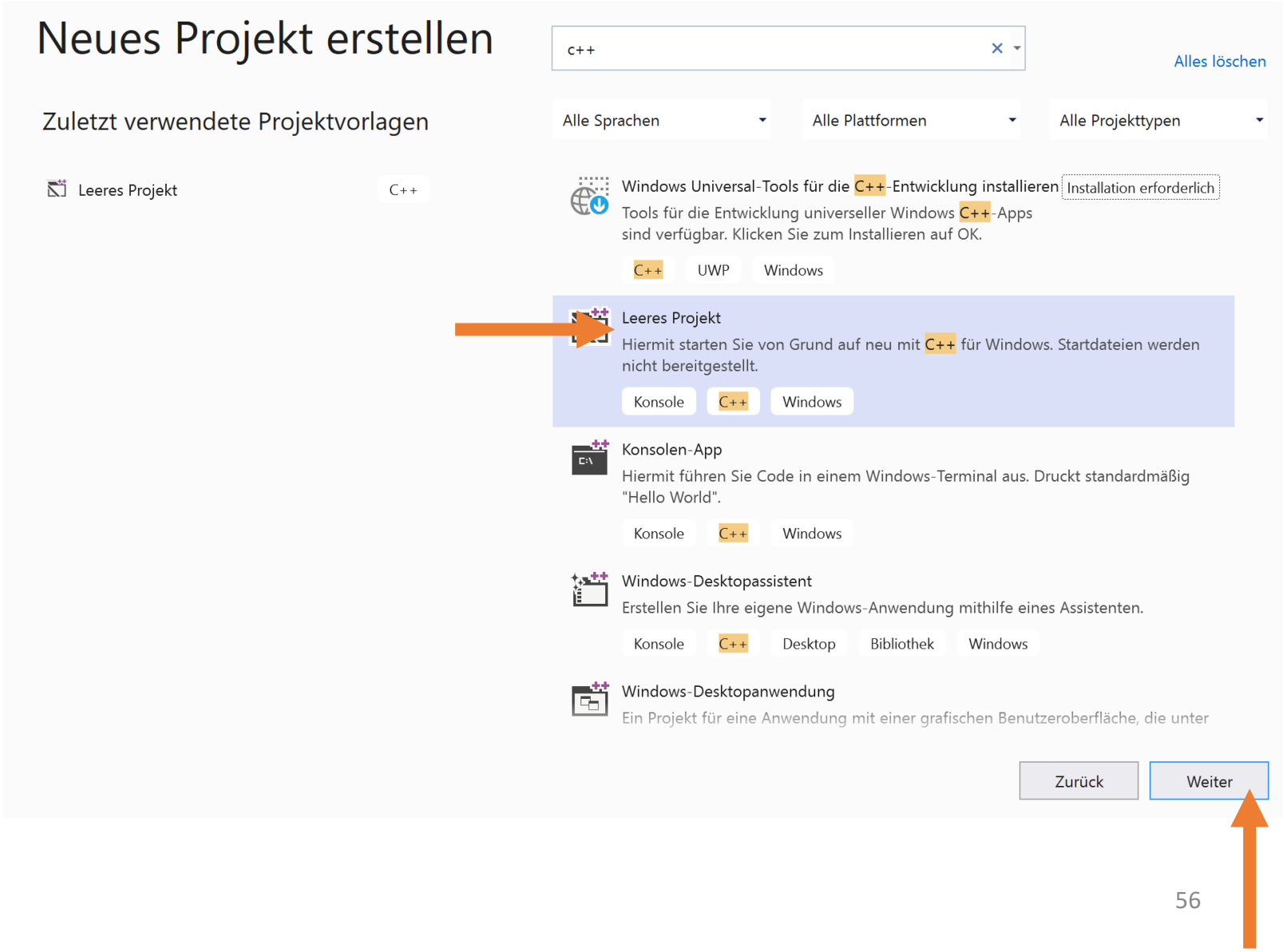
First Project

- Open visual studio and create a new project



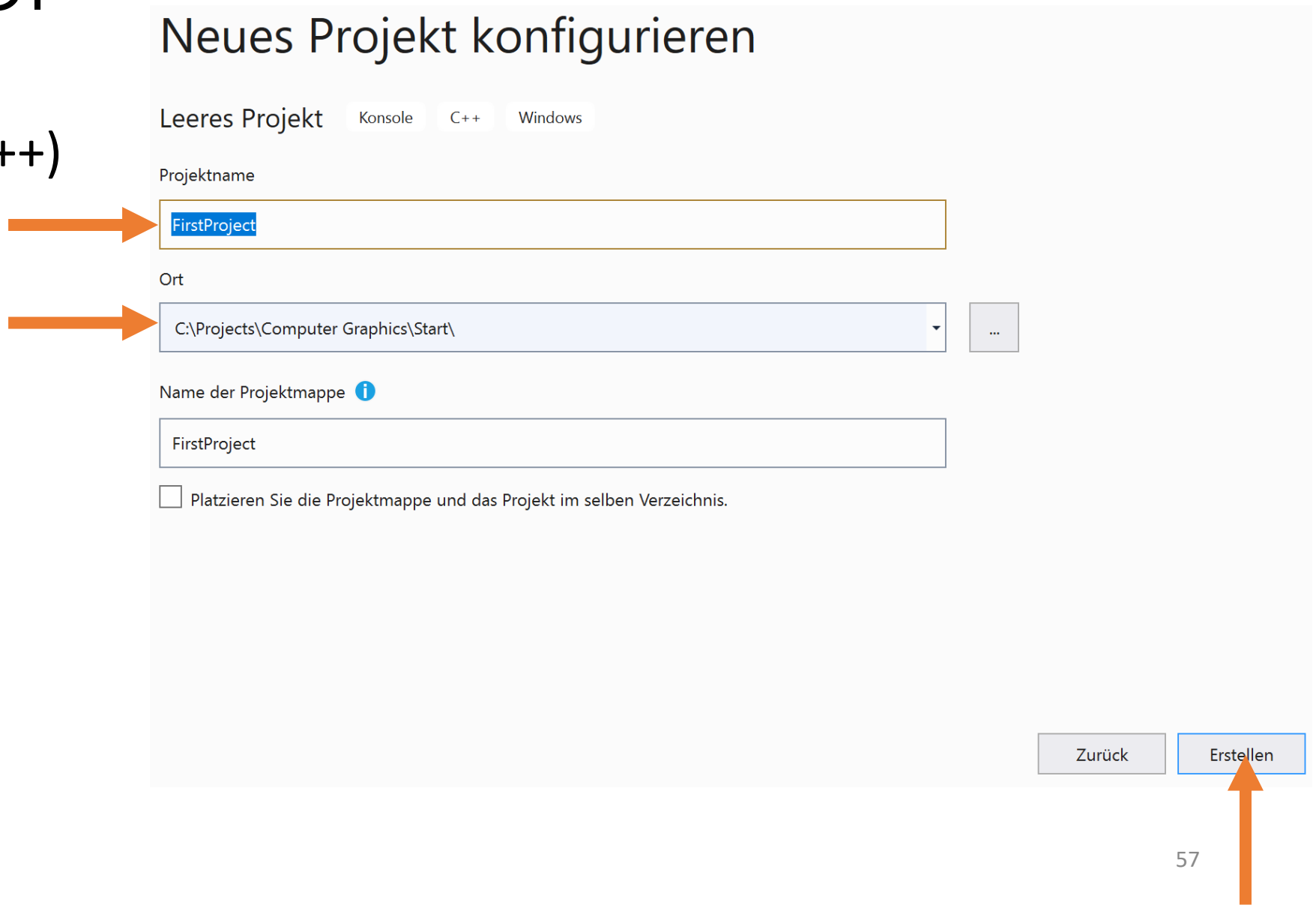
First Project

- Empty project (C++)



First Project

- Empty project (C++)



GLFW

- First, need to create an OpenGL context and an application window to draw in
- Those operations are specific per operating system and OpenGL purposefully tries to abstract from these operations -> Have to create a window, define a context and handle user input all by ourselves
- Few libraries out there that already provide the functionality
- Some of the more popular libraries are:
  - GLUT, SDL, SFML, and **GLFW**

GLFW

- GLFW is a library, written in C, specifically targeted at OpenGL
- Allows to create an OpenGL context, define window parameters and handle user input

GLFW

- Download (pre-compiled binaries) GLFW

Download

The current version is **3.3.2**, which was released on **January 20, 2020** . See the release notes for details.

Source package

This package contains the complete source code with CMake build files, documentation, examples and test programs. It is the recommended download for all platforms and offers the most control.

All development is done on GitHub. The `master` branch is our integration branch for the next feature release while the `3.3-stable` branch only adds bug fixes for patch releases.

Source package

GitHub repository

Windows pre-compiled binaries

These packages contain the GLFW header files, documentation and release mode static libraries, DLLs and import libraries for Visual C++ 2010-2019, MinGW-w64 and plain MinGW.

Binaries for Visual C++ 2010 and plain MinGW are only available in the 32-bit package.

64-bit Windows binaries

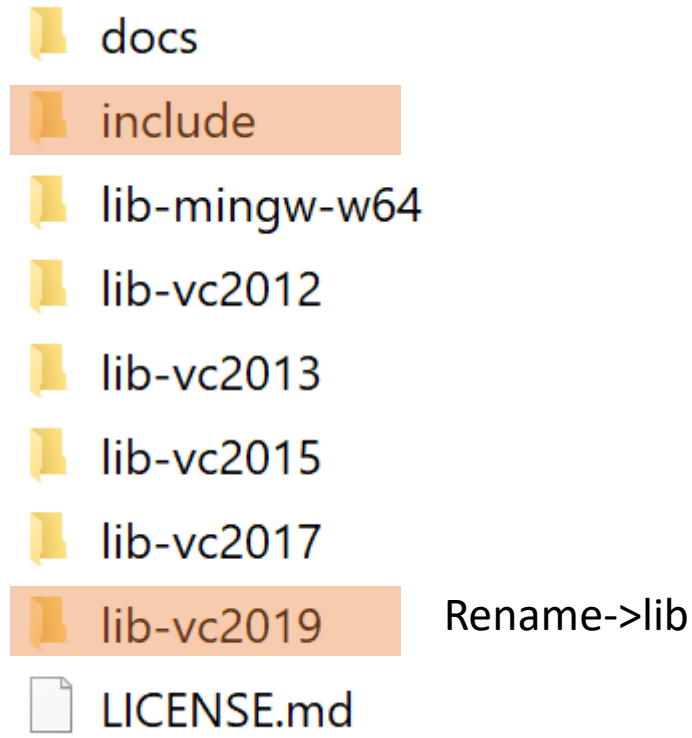
32-bit Windows binaries

https://www.glfw.org/download.html

GLFW

1. Unzip and Copy:

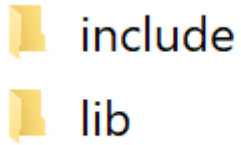
(rename lib-vc2019->lib)



Rename->lib

2. Create:

3. Paste:



GLAD

- OpenGL is up to the driver manufacturer to implement the specification to a driver that the specific graphics card supports
- -> Many different versions of OpenGL drivers (location of its functions not known at compile-time)
- The task of the developer to retrieve the location of the functions:

```
// define the function's prototype
typedef void (*GL_GENBUFFERS) (GLsizei, GLuint*);
// find the function and assign it to a function pointer
GL_GENBUFFERS glGenBuffers = (GL_GENBUFFERS)wglGetProcAddress("glGenBuffers");
// function can now be called as normal
unsigned int buffer;
glGenBuffers(1, &buffer);
```

GLAD

- The code looks complex and it’s cumbersome to do this for each function
- Thankfully, GLAD is a popular and up-to-date library for this task

```
// define the function’s prototype
typedef void (*GL_GENBUFFERS) (GLsizei, GLuint*);
// find the function and assign it to a function pointer
GL_GENBUFFERS glGenBuffers = (GL_GENBUFFERS)wglGetProcAddress("glGenBuffers");
// function can now be called as normal
unsigned int buffer;
glGenBuffers(1, &buffer);
```

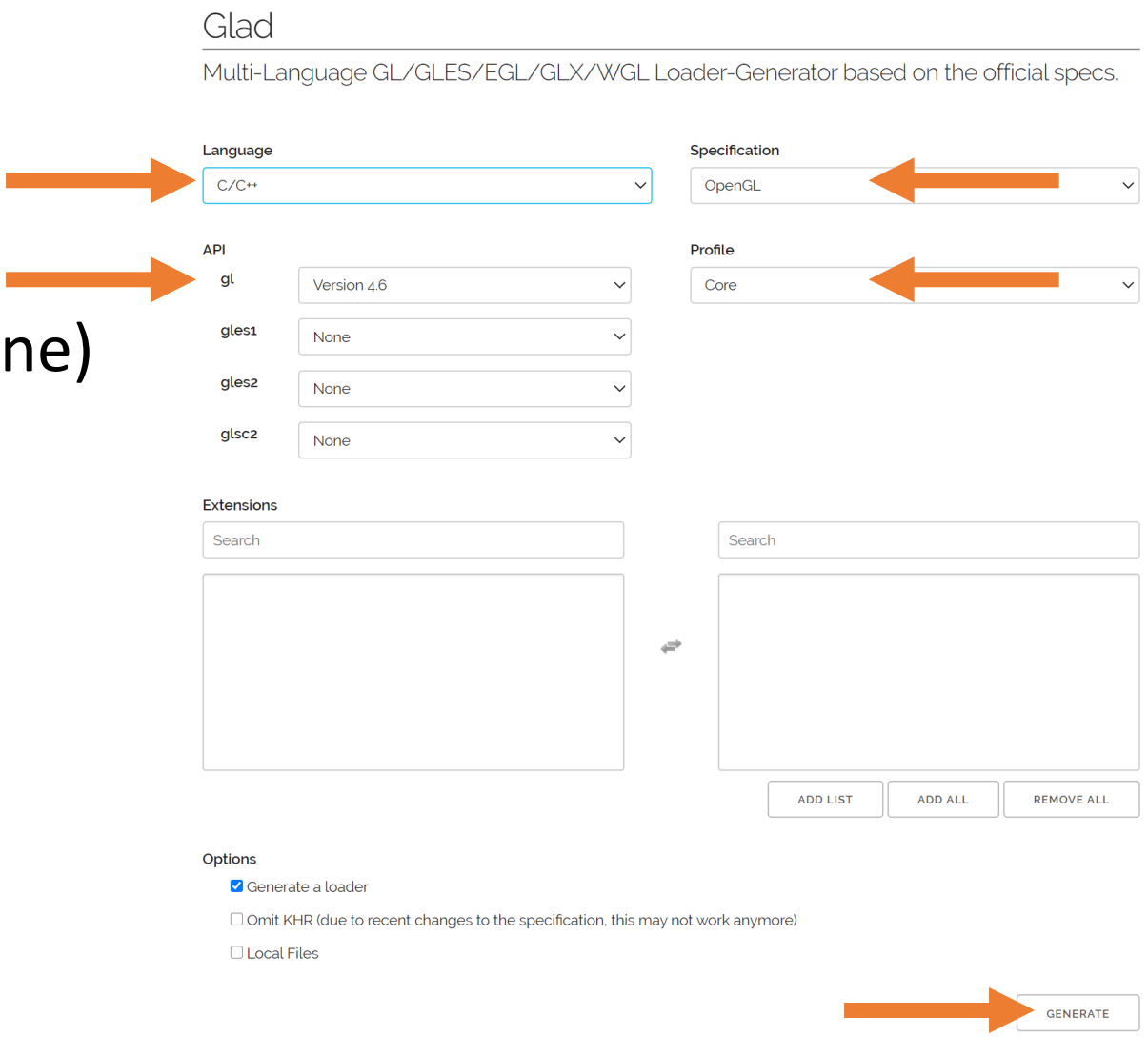
GLAD

- GLAD is open source and manages the work
- It uses a web services, where we can get all the functions by specifying C++ and the OpenGL version
- … but the repository provides us already with the necessary files

https://glad.dav1d.de/

GLAD

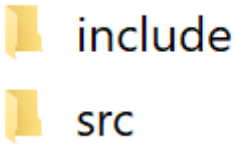
- Download GLAD
- gl Version at least 3.3 (higher is fine)



https://glad.dav1d.de/

GLAD

1. Unzip and copy:



2. Paste:

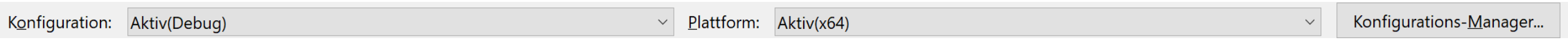
All together

C:\Projects\Computer Graphics\resources\

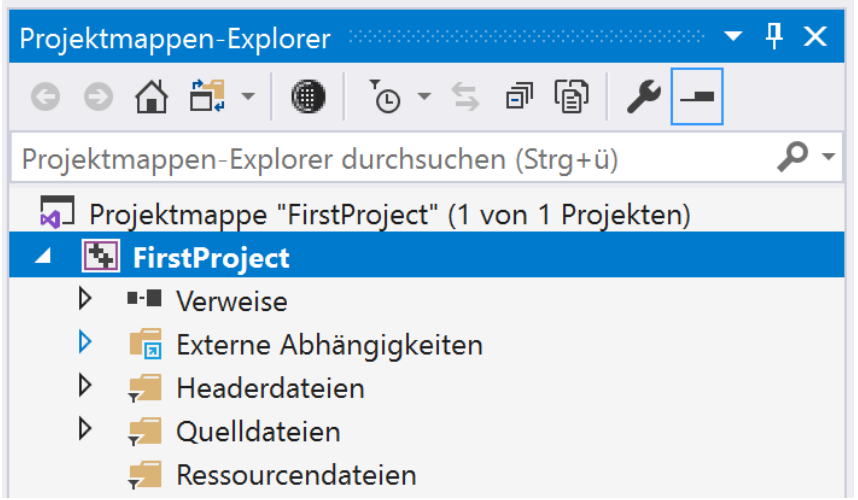
- include\
  - glad\glad.h
  - GLFW\{glfw3.h, glfw3native.h}
  - KHR\khrplatform.h
- lib\{glfw3.dll, glfw3.lib, glfw3dll.lib}
- src\glad.c

Back to Visual Studio

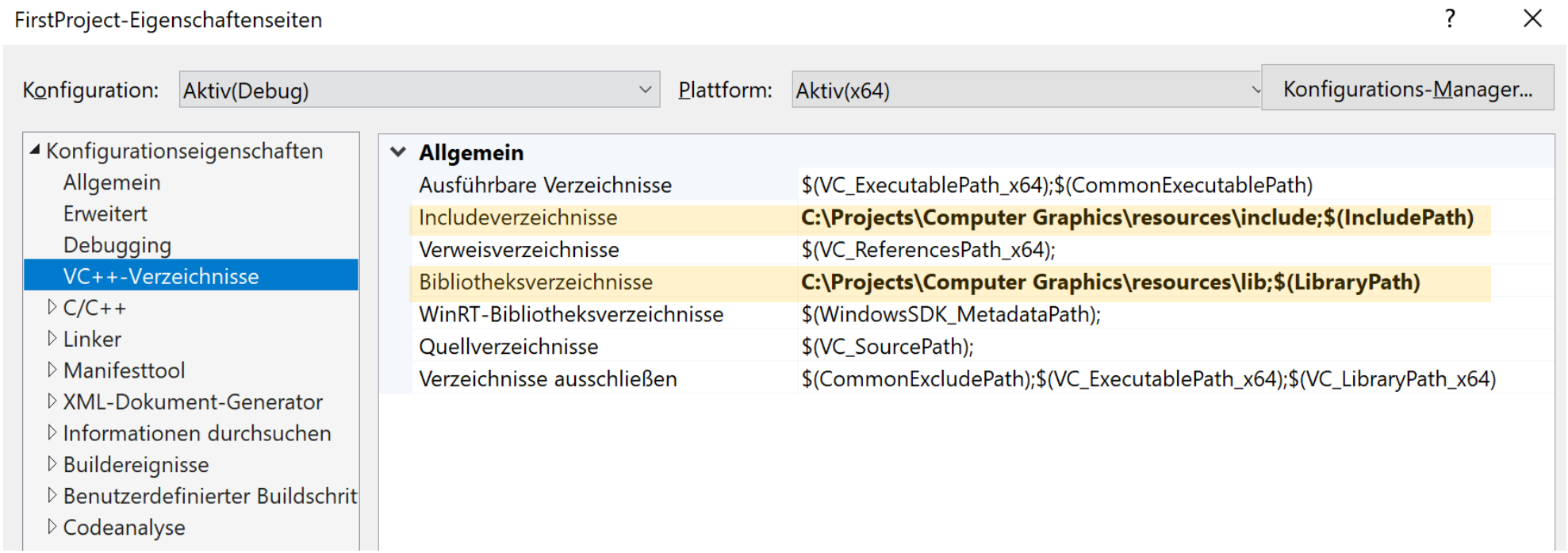
- Be sure for the following steps to choose the correct platform
- If you downloaded the 64-bit binaries the use the x64 platform in the upcoming settings



First Project



- Right click ‚FirstProject‘ -> Properties
- Or Alt+Return
- Add include and libraries from the repository

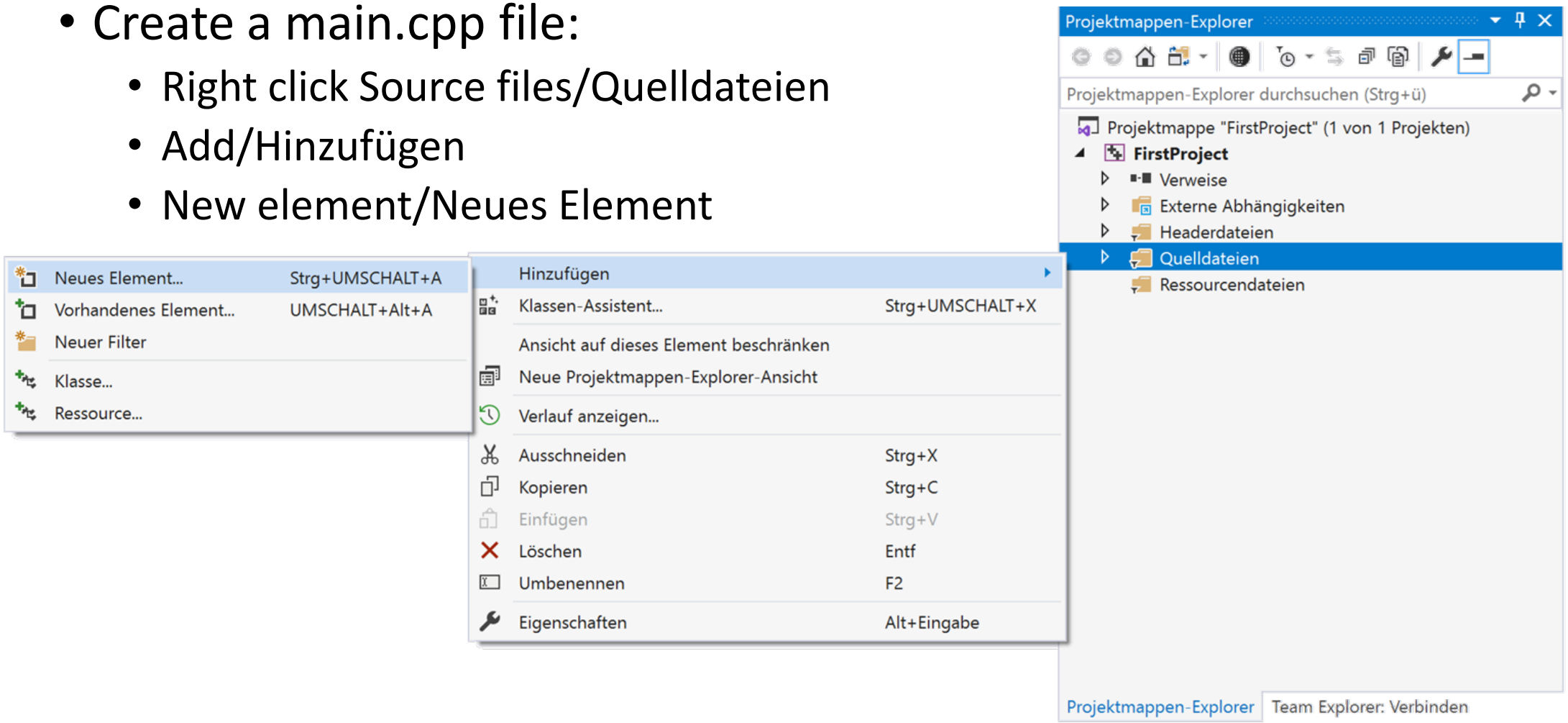


First Project

- Add glfw3.lib and opengl32.lib in the linker/input tab

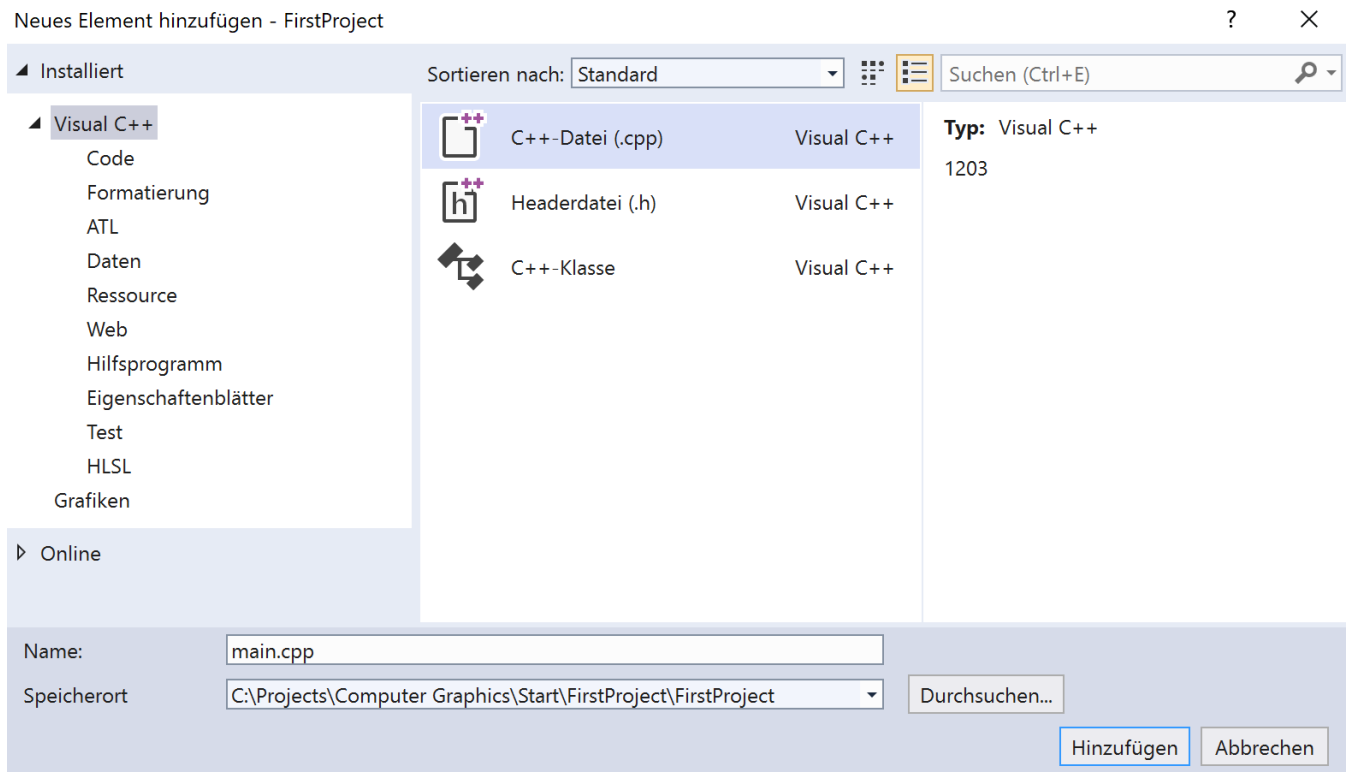
Hello World

- Create a main.cpp file:
  - Right click Source files/Quelldateien
  - Add/Hinzufügen
  - New element/Neues Element



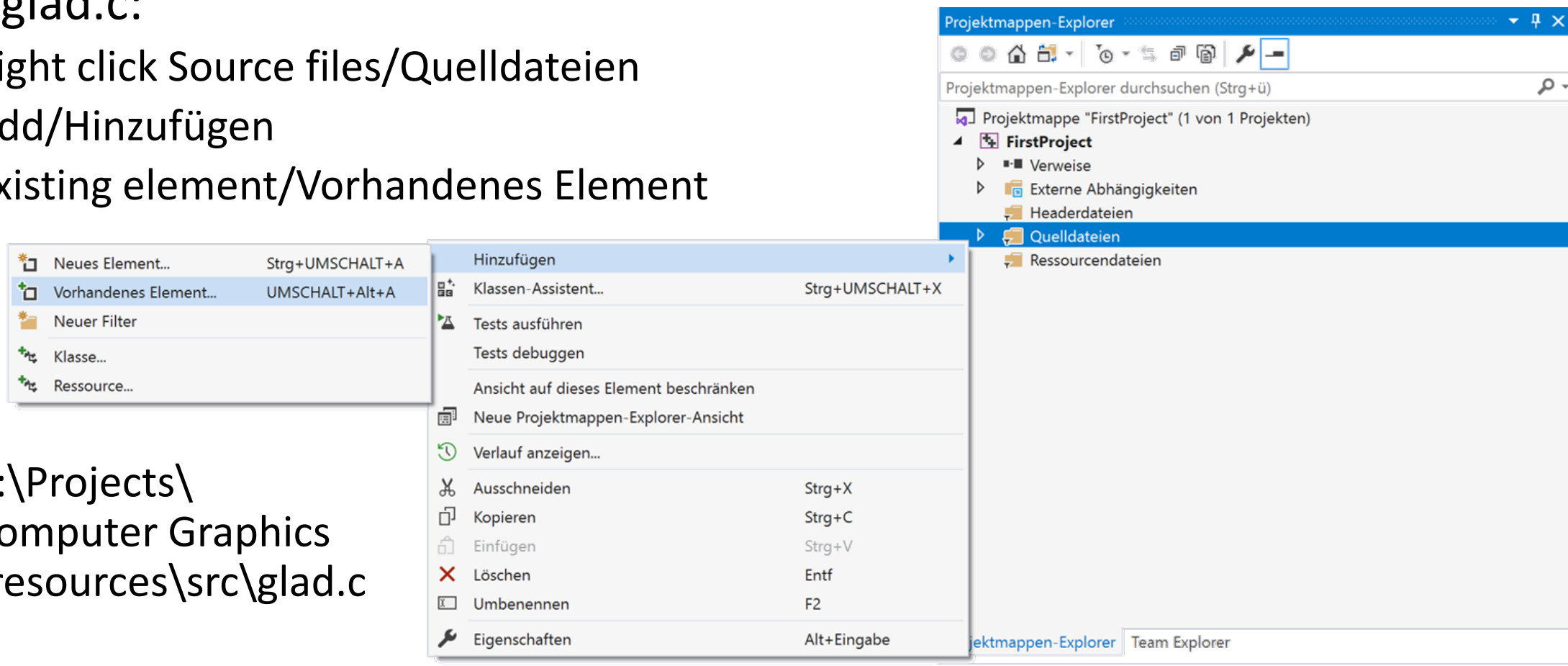
Hello Window

- Name it main.cpp and add it



Hello Window

- Add glad.c:
  - Right click Source files/Quelldateien
  - Add/Hinzufügen
  - Existing element/Vorhandenes Element



  - C:\Projects\Computer Graphics\resources\src\glad.c

Hello Window

- Open main.cpp and paste:

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>

int main()
{
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    //glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE); //for Mac OS X
    return 0;
}
```

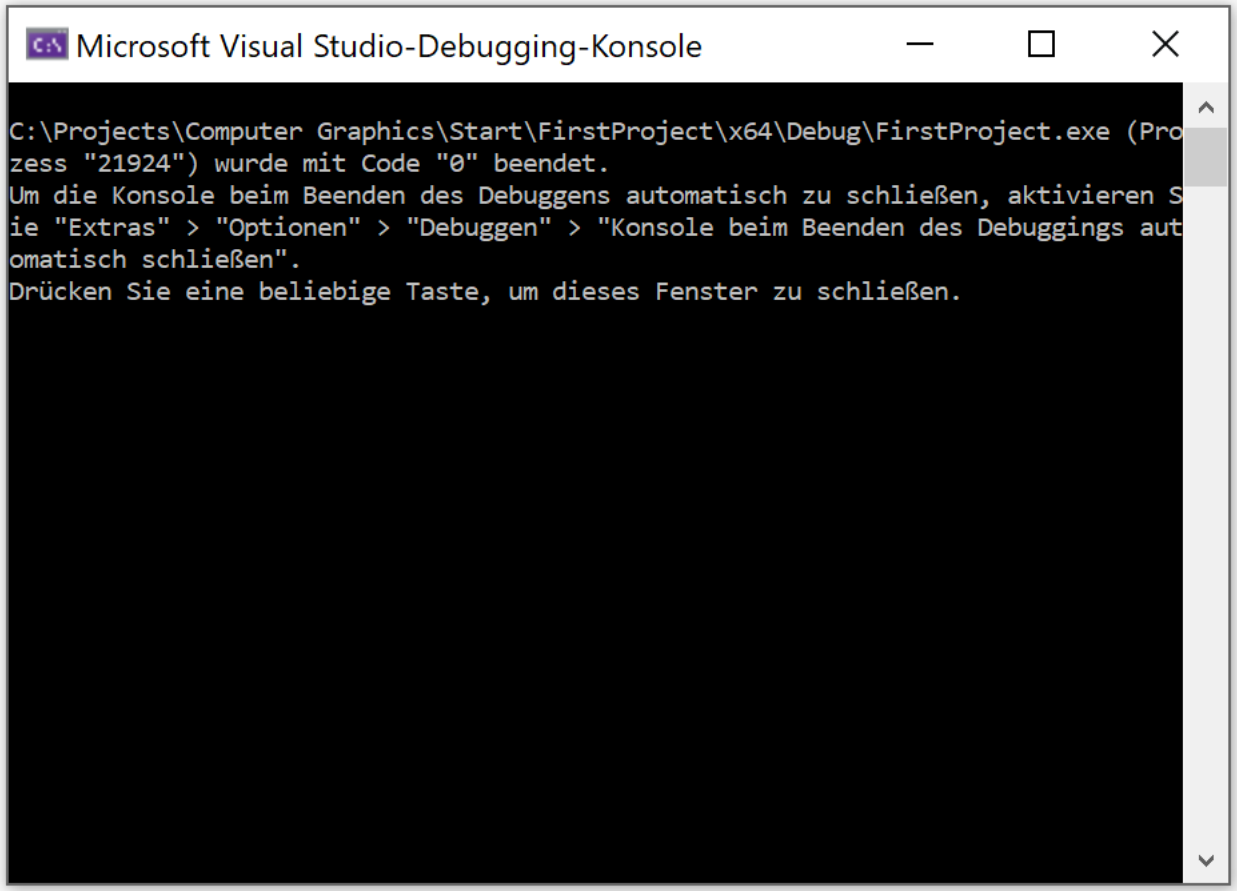
Hello Window

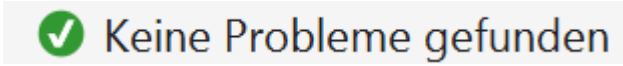
Be sure to include GLAD before GLFW!

Hello Window

- Press F5 and you get:

- Unspectacular, but it works
- (If not and a lot of *undefined reference* errors occur -> didn't successfully link the GLFW library)





Hello Window

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>

int main()
{
    glfwInit();
    glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);
    //glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE); //for Mac OS X
    return 0;
}
```

- First initialize GLFW with glfwInit
- Then, configure GLFW with glfwWindowHint -> first argument tells what to configure and set it as the second argument
- Here, the client API (application programming interface) version is set
- All options can be found at the GLFW's window handling documentation

- We use OpenGL 3.3 for now -> major & minor set to 3
- We also want to explicitly use the core-profile -> get access to a smaller subset of OpenGL features (without backwards-compatible features we no longer need)

https://www.glfw.org/docs/latest/window.html#window_hints

Hello Window

Make sure you have OpenGL versions 3.3 or higher installed.
Otherwise the application will crash or display undefined behavior.

To find the OpenGL version:

Linux: call glxinfo

Windows: use the OpenGL Extension Viewer

Hello Window

- Now, we really want to create a window

```
#include <iostream>
…
…
GLFWwindow* window = glfwCreateWindow(800, 600, "LearnOpenGL", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
```

Hello Window – GLAD

- GLAD manages function pointers for OpenGL: initialize GLAD before we call any OpenGL function

```
// glad: load all OpenGL function pointers
// ---------------------------------------
if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
{
    std::cout << "Failed to initialize GLAD" << std::endl;
    return -1;
}
```

Hello Window – Viewport

- OpenGL needs to know the size of the rendering window
- Set those dimensions via the glViewport function

```
glViewport(0, 0, 800, 600);
```

- The first two parameters set the location of the lower left corner of the window
- The third and fourth parameter set the width and height of the rendering window in pixels (retrieve from GLFW)
- Could set the viewport dimensions at smaller values -> then OpenGL rendering displayed in a smaller window (could, e.g., display other elements outside the viewport)

Hello Window – Viewport

OpenGL uses the glViewport data to transform the 2D coordinates it processed to coordinates on your screen.

Note that processed coordinates in OpenGL are between -1 and 1 so (-0.5,0.5) is mapped to (200,450).

Hello Window – Resize

- If user resizes the window the viewport should be adjusted
- Register a callback function on the window that gets called each time the window is resized:

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

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

Hello Window – Resize

- Have to tell GLFW to call this function on every window resize by registering it:

```
glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
```

- When the window is first displayed framebuffer_size_callback gets called as well with the resulting window dimensions
- (For retina displays width and height will end up significantly higher than the original input values)

Hello Window

- Many callbacks functions can set to register own functions:
  - E.g., Can make a callback function to process joystick input changes
  - Process error messages etc.
- Register the callback functions after the window was created and before the game loop is initiated

Hello Window

We compile the source and…

Hello Window

We compile the source and…

… the LearnOpenGL window pops out and closes immediately

Hello Window

- A single image is shown and then quits
- The application should keep drawing images and handling user input until the user quits it
- For this reason we have to create a while loop (render loop), that keeps on running until we tell GLFW to stop:

```
while (!glfwWindowShouldClose(window))
    {
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
```

Hello Window

```
while (!glfwWindowShouldClose(window))
    {
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
```

- The glfwWindowShouldClose function checks if GLFW has been instructed to close, if so, the game loop stops running
- The glfwPollEvents function checks if events are triggered (keyboard input, mouse movement events…), and calls the corresponding functions (via callback methods)
- The glfwSwapBuffers will swap the color buffer (front buffer, back buffer)

Hello Window – Double Buffer

When an application draws in a single buffer the resulting image might display flickering issues.

To circumvent these issues, windowing applications apply a double buffer for rendering: front buffer contains the final output image, while all the rendering commands draw to the back buffer.

As soon as all the rendering commands are finished we swap the back buffer to the front buffer.

Hello Window – One last thing

- As soon as we exit the render loop we would like to properly clean/delete all resources that were allocated:

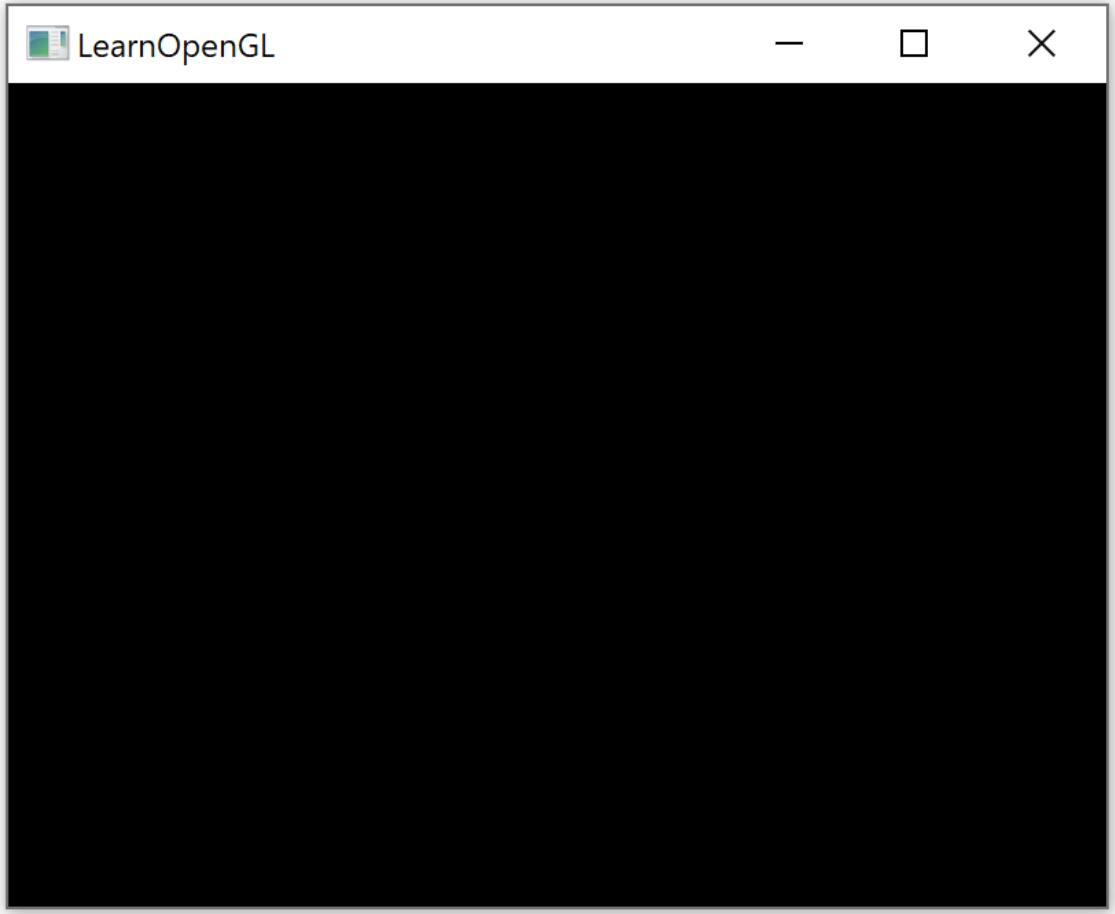
```
glfwTerminate();
return 0;
```

Hello Window

We compile the source and…

Hello Window

We compile the source and…



Hello Window – All

```
#include <glad/glad.h>
#include <GLFW/glfw3.h>
#include <iostream>

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

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

    GLFWwindow* window = glfwCreateWindow(800, 600,"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;
    }
    while (!glfwWindowShouldClose(window))
    {
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
    glfwTerminate();
    return 0;
}
void framebuffer_size_callback(GLFWwindow* window, int width, int height)
{
    glViewport(0, 0, width, height);
}
```

Hello Window – Input

- Input a key with GLFW's glfwGetKey function (press ESC closes):

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

```
while (!glfwWindowShouldClose(window))
    {
        processInput(window);
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
```

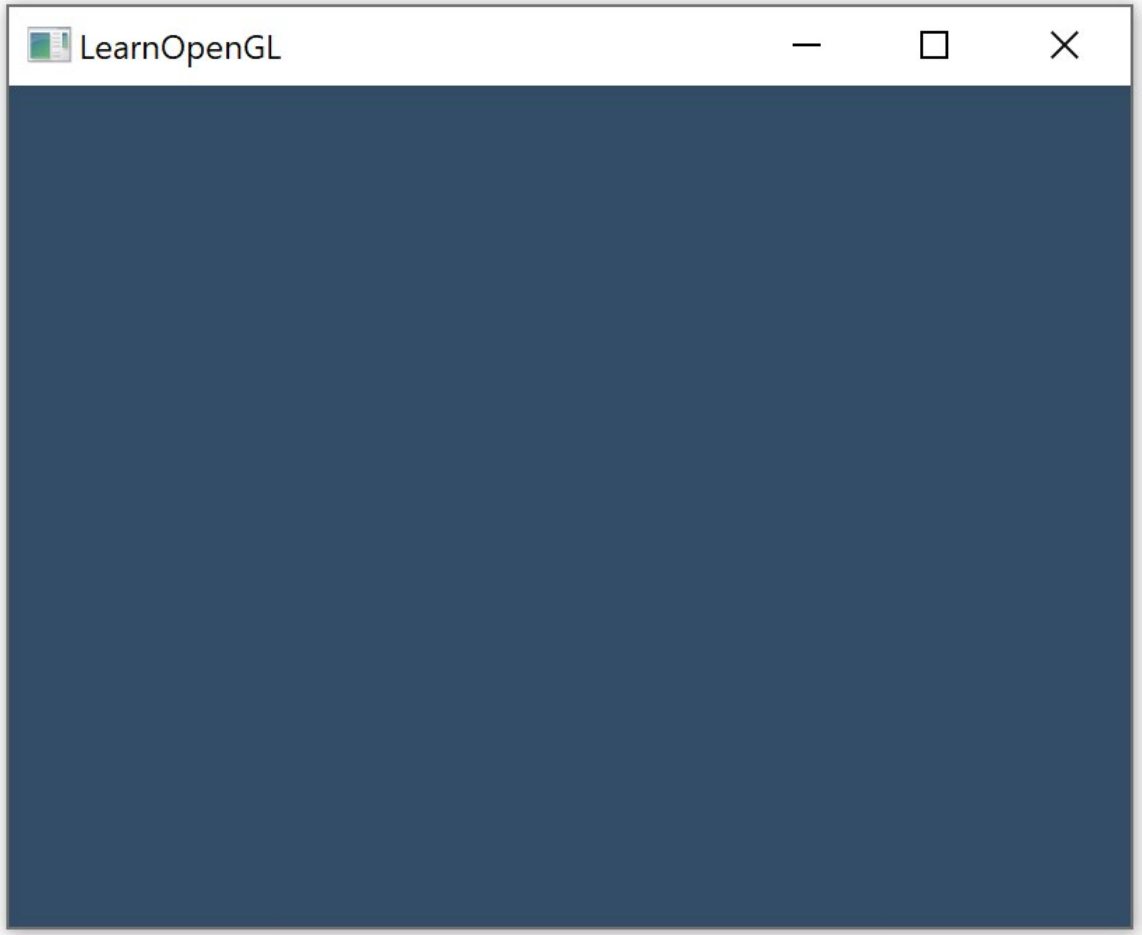
Hello Window – Rendering

- All the rendering commands in the render loop, since we want to execute all the rendering commands each iteration of the loop:

```
while (!glfwWindowShouldClose(window))
    {
        // input
        processInput(window);

        // render
        glClearColor(0.2f, 0.3f, 0.4f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT); // color, depth, stencil are possible

        // check and call events and swap the buffers
        glfwSwapBuffers(window);
        glfwPollEvents();
    }
```

Hello Window

We compile the source and…



Next Time

- … we want finally draw something!

Questions???