GED Practical Course



computer graphics & visualization

Assignment 01

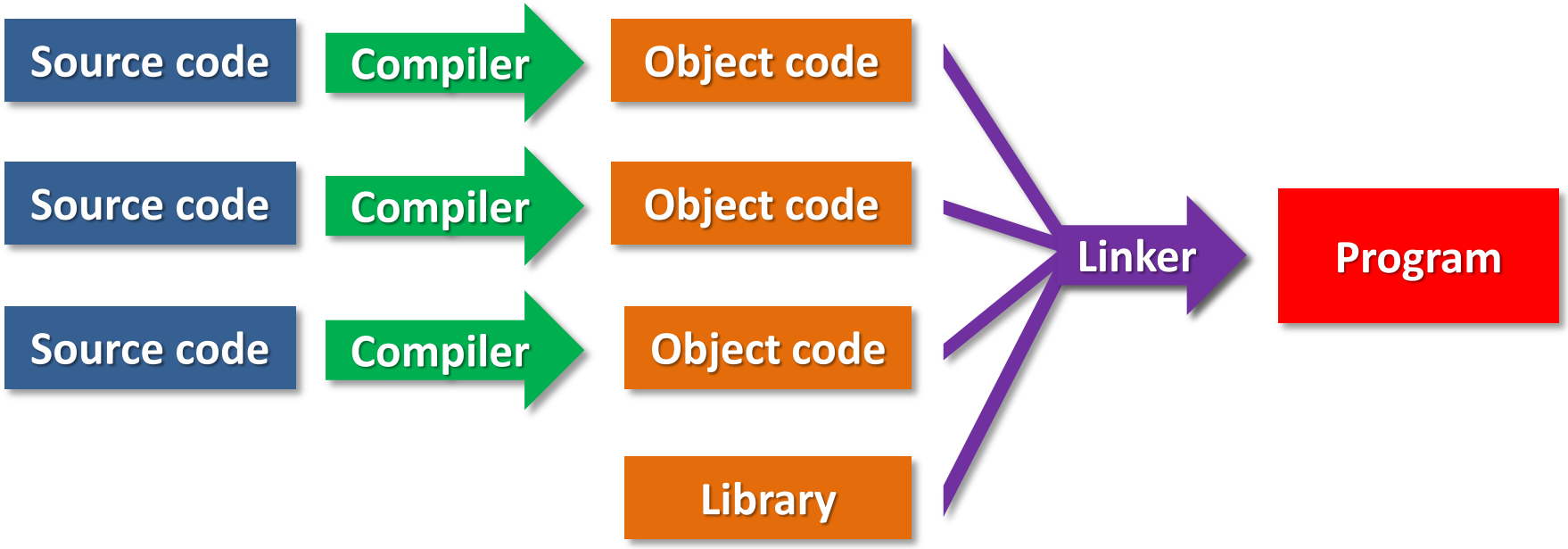
- This week:
  - C++ Introduction
  - Assignment
    1. Hello World
    2. 2D Arrays
    3. Using `std::vector`
    4. File I/O with streams

C++: Compiler

- Java
  - Everything is compiled into `.class` files
  - Information about other class files is automatically generated at compile time
  - `.class`-Files can be directly executed on the JVM
- C++
  - Classes and methodes are compiled into `.obj` files
  - Information about other included files must be declared explicitly
  - `.obj` files are linked together into final executable program

C++

- Source code is compiled into object code (*.obj)
- Object code and libraries (*.lib) are linked together into an executable program (*.exe)



C++: Memory management

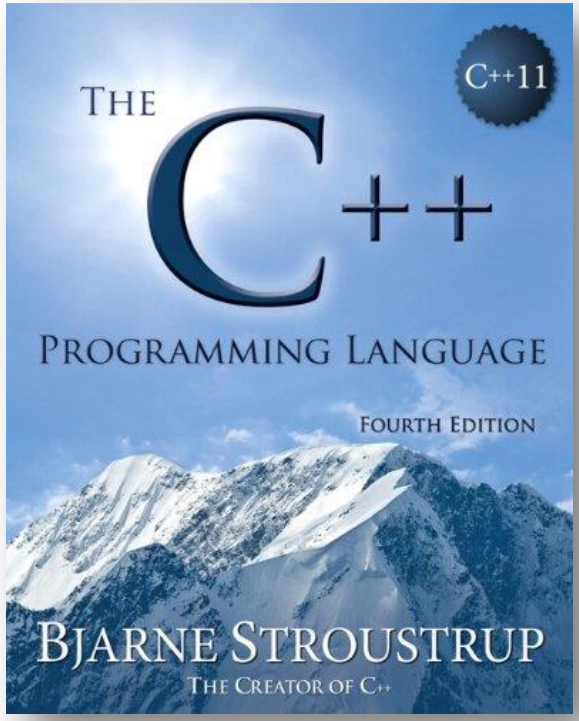
- Memory management in C++ is explicit
  - Allocating with `new`
  - Free memory with `delete`
  - One `delete` for every `new`!
- Memory addresses are stored with pointers
  - `int* pData = new int; …; delete pData;`
  - `pData` points to an element of type `int`
    - Compiler knows type and size of data
  - `int* myArray = new int[10]; …; delete[] myArray;`

C++ Introduction

- For a further introduction to C++ let us refer to the C++ Primer on our webpage.
- Reference
  - http://www.cplusplus.com/reference/
- Online Tutorial
  - http://www.cplusplus.com/doc/tutorial/
- Book

  Die C++-Programmiersprache / The C++ Programming Language (4th Edition)

  Bjarne Stroustrup



Assignment 01

1. Hello World

   – Print „Hello World“ on the console

   – … yeah, that‘s about it

Streams

- Check the C++ slides on streams!
  - You can “chain” the streaming operators
  - `std::cout << “test” << myVar << “\n”`
  - “\n” denotes “newline”

- There is also an extraction operator >>

```
int userInput;
std::cin >> userInput;
```

  - `std::cin` is the input of the default console
  - Can be any other stream, of course

    (filestream? *incredibly hidden hint for the last task!*)

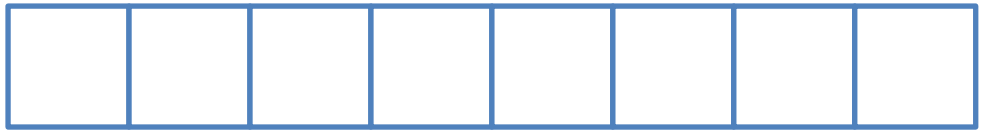
2. Smoothing Values in a 2D Array

- Create a 2D array
- Fill it with random values
- Replace every value by the average of the 3x3 surrounding values

Arrays

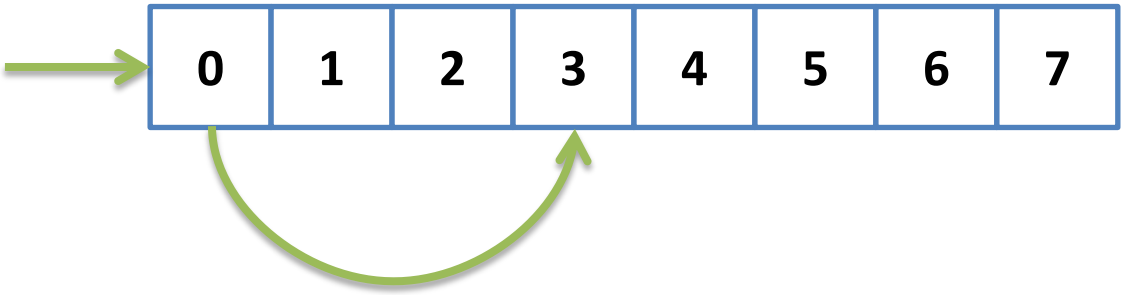
- There is no native “Array” Type in C++
  - Allocate memory for a fixed number of elements:

```
int* a = new int[8];
```



  - Address of the first element + size of each element known

  - Offset addressing: `a[3]` *means a + 3 * sizeof(datatype)*



Arrays

- 2D Array: “Flatten” the array
  - Allocate 1D array with appropriate size
  - Calculate 1D position from 2D position and width
  - You can use a preprocessor macro for that:
    - Common convention: Macros and defines are ALL_CAPS

```
// Access a 2D array of width w at position x / y
#define IDX(x, y, w) ((x) + (y) * (w))
```

  - Or, alternatively you can use an inline function:

```
inline int idx(int x, int y, int w) {return x + y * w;}
```

  - Access at position (2, 4) of an array of width 10:

```
a[IDX(2, 4, 10)] = 10;
```

  - Brackets in macros are important: Simple text replacement!
    - IDX(2, 4 + 1, 10) would fail without brackets: 2 + 4 + 1 * 10

Randomization

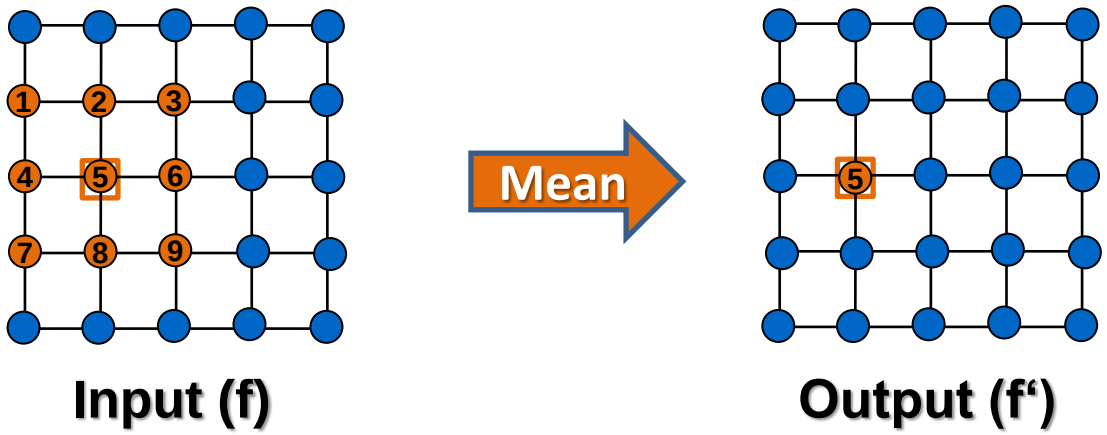
- Simple pseudo random number generator: `rand()`
  http://www.cplusplus.com/reference/cstdlib/rand/
  - `#include <cstdlib>`
  - Creates integer values in `[0, RAND_MAX]`

- Each PRNG needs an initial value („seed“)
  - Seed with `srand()`
    http://www.cplusplus.com/reference/cstdlib/srand/
  - The same seed will create the same numbers
  - Common: Seed with current time
    - `#include <ctime>`
    - `time(nullptr)` returns the current time
      http://www.cplusplus.com/reference/ctime/time/

Mean Value Filter

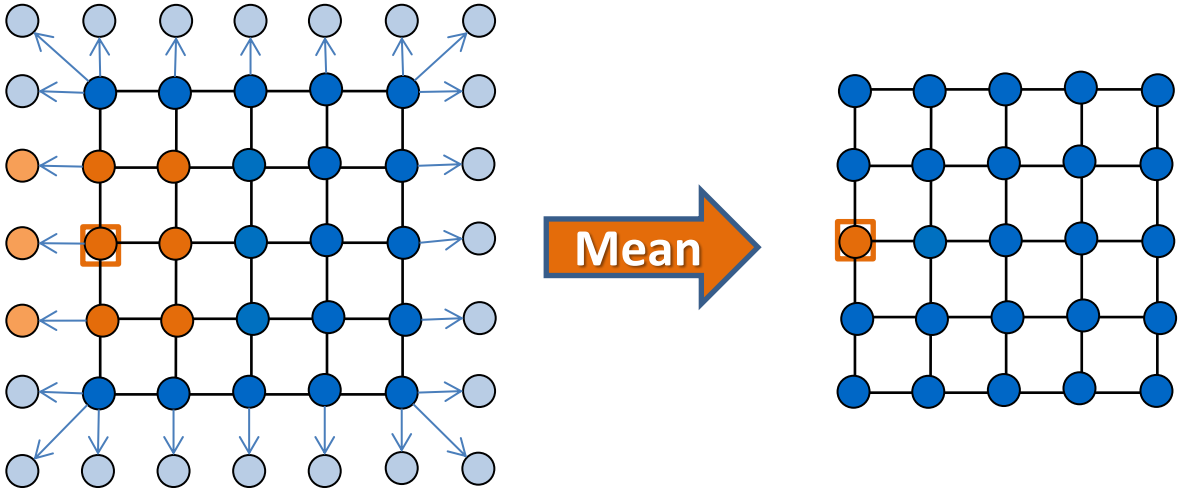
- 3x3 Mean Value Filter
  - Replace height value of each grid point with mean height value from its 3x3 neighborhood
  - Use two separate arrays for input and output, otherwise data is overwritten before it can be read
  - At boundaries, reduce filter size or extend grid (see next slide)

$$f'_5 = \frac{1}{9}(f_1 + f_2 + f_3 + f_4 + f_5 + f_6 + f_7 + f_8 + f_9)$$

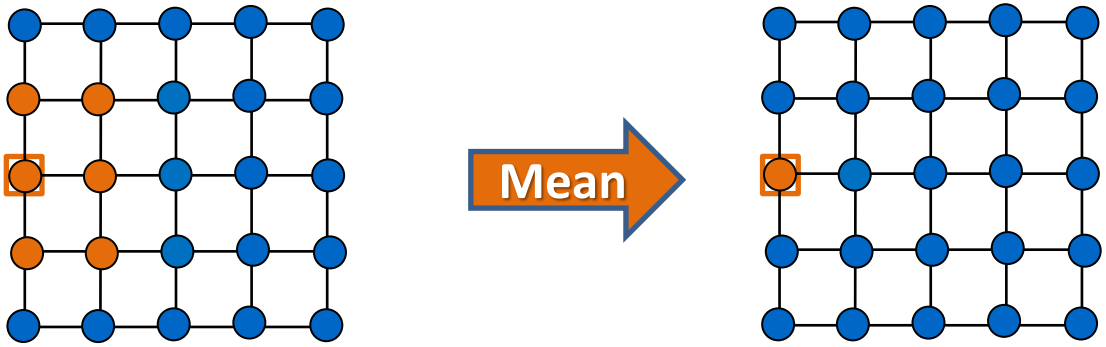


Boundary treatment

- Extend grid by cloning values at boundary



- Reduce filter size



3. Sorting a Vector of integers

- Create a vector of integers
- Fill it with values from user input
- Sort it using the `std::sort` algorithm
- Print it to the console

Vector

- `std::vector<>`: A safe array
  http://www.cplusplus.com/reference/vector/vector/
  - Templated (like all C++ std containers)
    - Think of it like a Java Generic
    - We will spare you the detailed horror of C++ templates ;-)
  - Will automatically resize when out of memory
  - Note: Resizing will copy all existing elements to new memory
    - Carefull when you push self-created classes in a vector
    - Remember the slides on pointers inside classes!

Vector

- Important methods (check the documentation!)
  - `push_back()`
  - `clear()`

- Overloads `operator[]`
  - “Look and feel” of a simple pointer array
  - Checks array bounds in DEBUG build!

```
std::vector<int> a;
a.push_back(10);

std::cout << a[0];   // Prints "10"

a[0] = 23;
std::cout << a[0];   // Prints "23"
```

std::sort

- The standard library contains many algorithms on its containers
  - `#include <algorithm>`
  - Works on iterators
    - All containers provide `.begin()` and `.end()` as well as their const Versions `.cbegin()` and `.cend()`
    - `begin()` is the first element
    - `end()` is the element **after** the last element
      - … because loops run as long as **iterator != end()**

std::sort

- `std::sort`: Sorts a container
  http://www.cplusplus.com/reference/algorithm/sort/

- Expects two iterators and (optional) a comparison function
  - Default comparison: `operator<`
  - We want to sort in **descending** order, so we need to use a custom function

  - This can be done in over 9000 ways… f*ck yeah, C++!
  - Nearly “everything” that can be interpreted as a function of two parameters returning a bool…

std::sort

- Simplest way: Define a (non-member!) function
  - `bool f(int, int)`
  - Pass it as the third parameter (without any brackets)

- Alternative: Instance of a class / struct with operator()
  - Also `bool operator()(int, int)`

- “Pro version”: C++ Lambda expressions
  - `[](const int& l, const int& r) -> bool { … }`
  - Beautifully compact, but a bit hard to read

- And many more…

4. Configuration parser

- Open a file containing key-value pairs
- Parse this file for known keys
- Store them into variables
- Output everything to the console

Config Parser

- Example game.cfg
  - Each line contains a key and an associated value
  - Values can be complex types
  - e.g. backgroundColor is a color given as R/G/B values [0.0, 1.0]

```
1  spinning 1.0
2  spinSpeed 1.0
3
4  backgroundColor 1.0 1.0 1.0
5
6  terrainWidth 64
7  terrainDepth 64
8  terrainHeight 64
9
10 terrainPath "C:\Test\test"
```

Config Parser

- Create a new class
  - Use the “add class” wizard as described in the assignment
  - It will create a .h and .cpp file already containing a class stub

- `struct Color`
  - Remember, structs work just like classes
  - Usually used for simple types with public members

  - You can declare a class / struct inside another class
    - The outer class acts like a namespace
    - Access from outside: `MyClass::MyNestedClass`
    - Only if it is declared in the public section, of course

Next week

- Due to the holiday on May 1st, there will be no lecture and practical course.
- The basic concepts of the Diamond Square algorithm were explained today and some implementation issues will be discussed on Monday, April 27th.
- The tutoring groups will take place on a regular basis.
- The slides and the assignment for next week are already published in the SVN repository “external”. **Please study them by yourself!**
- Remember, that starting from Ass. 2 your “solution” will be graded.

Questions?