9. Übungsblatt - C++

Henrik Gerdes, Manuel Eversmeyer
D10

7. Januar 2019

```
1. Es sollen alle geraden Zahlen aus dem Vektor entfernt werden. Dazu
wird ein Interator und remove_if benutzt


2. Bind erwartet einen Funktionspointer oder Funktionsobjekt als ersten
Parameter, dieser ist bei den inneren beiden bind Aufrufe mit den
Funktionen equal_to und modulus gegeben.
Als weitere Parameter erwartet bind eine Liste an beibig vielen Parametern.
Diese werden an das Funktionsobjekt übergeben. Diese Parameter können
konkrete Werte wie 0 und 2 sein oder Platzhalter wie _1.
Das dritte bind soll alle Elemente im Vector an modulus mit 2 binden.
Das zweite bind soll dann prüfen ob die mit modulus verarbeiteten Elemente
    = 0 sind.
Die verarbeiteten Elemente soll das zweite bind vom ersten bind bekommen.
Dieses gibt dann eine Rückgabe, die ein gültiges Argument für remove_if ist
    .

Bsp:
->   bind(equal_to<int>(),_1,0)        //Liefert True wenn für den Platzhalter
     _1 0 eingesetzt wird.
->   bind(modulus<int>(),_1,2)         //Für alle Paltzhaltwer wird %2
    berechnet

3. Der return-value von 'remove_if' der in der Variable 'it' gespeichert
    wird und das neue Ende des Vektors angibt muss für die Ausgabe in der
    vorletzten Zeile genutzt werden:

vector<int> v = {1, 4, 2, 8, 5, 7};

copy(v.begin(), v.end(), ostream_iterator<int>(cout, " "));
cout << endl;

auto it = remove_if(v.begin(), v.end(),bind(bind(equal_to<int>(),_1, 0),
    bind(modulus<int>(), _1, 2)));

copy(v.begin(), it, ostream_iterator<int>(cout, " "));
cout << endl;
```

Aufgabe_1.txt

```
#########################################
cmake_minimum_required(VERSION 3.0.2)

#########################################
# The name of out project
#########################################
project(CPP18_ASTEROIDS)
project(CPP18_FUNCTIONALS)

//Like old CMake
...

set( CMAKE_CXX_STANDARD 11)
set( CMAKE_CXX_STANDARD_REQUIRED ON)

//Like old CMake
...

set(FUNCTIONALS_SOURCES
    Functionals.cpp)

//Like old CMake
...


add_executable(asteroids ${ASTEROID_SOURCES} ${C3DSREADER_SOURCES})
add_executable(functionals ${FUNCTIONALS_SOURCES})
```

CMakeLists–Kopie.txt

```
#include <vector>
#include <functional>
#include <algorithm>
#include <iostream>
#include <iterator>


int main(int argc, char const *argv[])
{
    using namespace std::placeholders;

    std::vector<int> v = {1, 4, 2, 8, 5, 7};

    copy(v.begin(), v.end(), std::ostream_iterator<int>(std::cout, " "));
    std::cout << std::endl;

    auto it = remove_if(v.begin(), v.end(),
        std::bind(std::bind(std::equal_to<int>(), _1, 0),
            bind(std::modulus<int>(), _1, 2)));

    copy(v.begin(), it, std::ostream_iterator<int>(std::cout, " "));
    std::cout << std::endl;


    return 0;
}
```

Functionals.cpp

```cpp
#ifndef SHARED_ARRAY_HPP
#define SHARED_ARRAY_HPP

#include <memory>

namespace asteroids
{

template<typename T> class shared_array : public std::shared_ptr<T>
{
    public:
    /**
     * @brief Construct a new shared array object
     *
     */
    shared_array();

    /**
     * @brief Construct a new shared array object
     *
     * @param arr
     */
    shared_array(T* arr);

    operator T()
    {
        return T();
    }

    /**
     * @brief   Indexed element (reading) access.
     */
    T& operator[](const int index);


    /**
     * @brief   Indexed element (reading) access.
     */
    T operator[](const int index) const;

    // a lamda array deleter:
    struct Del {
        void operator()(T* p)
        {
            delete[] (p);
        }
    };
};

}; //asteroids

#include "shared_array.tcc"
#endif
```

util/shared_array.hpp

```cpp
#include "shared_array.hpp"

namespace asteroids
{

template<class T>
shared_array<T>::shared_array()
{

}

template<class T>
shared_array<T>::shared_array(T* arr):std::shared_ptr<T>(arr, [](T* p){})
{

}

template<class T>
T shared_array<T>::operator[](const int index) const
{
    return this->get()[index];
}

template<class T>
T& shared_array<T>::operator[](const int index)
{
    return this->get()[index];
}


}; //asteroids
```

util/shared_array.tcc

```cpp
/*
 *  TextureFactory.cpp
 *
 *  @date 18.11.2018
 *  @author Thomas Wiemann
 *
 *  Copyright (c) 2018 Thomas Wiemann.
 *  Restricted usage. Licensed for participants of the course "The C++
    Programming Language" only.
 *  No unauthorized distribution.
 */

#include "TextureFactory.hpp"
#include "ReadPPM.hpp"
#include "ReadTGA.hpp"
#include "ReadJPG.hpp"
#include "BitmapReader.hpp"
#include "../util/shared_array.hpp"

#include <iostream>
using std::cout;
using std::endl;

namespace asteroids
```

```
{

using uCharPtr = shared_array<unsigned char>;
using textPtr = shared_array<Texture>;
using BitmapReadPtr = shared_array<BitmapReader>;

std::map<string, Texture*> TextureFactory::m_loadedTextures;

string TextureFactory::m_basePath;

TextureFactory::TextureFactory()
{
    // TODO Auto-generated constructor stub

}

TextureFactory::~TextureFactory()
{
    // TODO Auto-generated destructor stub
}

TextureFactory& TextureFactory::instance()
{
    // Just crate one instance
    static TextureFactory instance;
    return instance;
}

void TextureFactory::setBasePath(const string& base)
{
    m_basePath = base;
}

Texture* TextureFactory::getTexture(const string& filename)
{
    // A texture object


    textPtr tex;

    string tex_filename = m_basePath + filename;

    std::map<string, Texture*>::iterator it = m_loadedTextures.find(
        tex_filename);
    if(it == m_loadedTextures.end())
    {
        // Texture data
        int width = 0;
        int height = 0;
        uCharPtr data;
        BitmapReadPtr reader;

        // Get file extension
        if(filename.substr(filename.find_last_of(".") + 1) == "ppm")
        {
            reader = new ReadPPM(tex_filename);
        }
        else if(filename.substr(filename.find_last_of(".") + 1) == "tga")
```

```
81          {
82              reader = new ReadTGA(tex_filename);
83          }
84          else if(filename.substr(filename.find_last_of(".") + 1) == "jpg")
85          {
86              reader = new ReadJPG(tex_filename);
87          }
88
89          if(reader)
90          {
91              data = reader->getPixels();
92              width = reader->getWidth();
93              height = reader->getHeight();
94          }
95
96          // Check data and create new texture if possible
97          if(data.get() != 0 && width != 0 && height != 0)
98          {
99              tex = new Texture(data.get(), width, height);
100             m_loadedTextures[tex_filename] = tex.get();
101         }
102         else
103         {
104             cout << "TextureFactory: Unable to read file " << tex_filename
                    << "." << endl;
105         }
106
107         return tex.get();
108     }
109     else
110     {
111         return m_loadedTextures[tex_filename];
112     }
113 }
114 }//asteroids
```

io/TextureFactory.cpp

```
1  /*
2   *  TriangleMesh.hpp
3   *
4   *  Created on: Nov. 04 2018
5   *      Author: Thomas Wiemann
6   *
7   *  Copyright (c) 2018 Thomas Wiemann.
8   *  Restricted usage. Licensed for participants of the course "The C++
        Programming Language" only.
9   *  No unauthorized distribution.
10  */
11
12 #ifndef __TriangleMesh_HPP__
13 #define __TriangleMesh_HPP__
14
15 #include <string>
16
17 #include "Renderable3D.hpp"
18
19 namespace asteroids
```

```
20 {
21
22
23 /**
24  * @brief A struct to represent a simple 3D TriangleMesh
25  *
26  */
27 class TriangleMesh : public Renderable3D
28 {
29 public:
30
31     using floatPtr = shared_array<float>;
32     using intPtr = shared_array<int>;
33
34     TriangleMesh();
35
36     /**
37      * @brief Construct a new TriangleMesh object from another instance
38      *
39      * @param other         Instance to clone
40      */
41     TriangleMesh(const TriangleMesh& other);
42
43     /**
44      * @brief   Contructs a triangle mesh from given buffers
45      * @param indexBuffer indexbuffer
46      * @param vertexBuffer vertexbuffer
47      * @param numFaces number of faces
48      * @param numVertices number of vertices
49      */
50     TriangleMesh(int* indexBuffer, float* vertexBuffer, float* normals, int
         numFaces, int numVertices);
51
52
53     /**
54      * @brief Prints general information (number of vertices and faces)
55      *        to stdout.
56      *
57      */
58     void printTriangleMeshInformation();
59
60     /**
61      * @brief Prints the contends of the internal buffers to stdout.
62      *
63      */
64     void printBuffers();
65
66     /**
67      * @brief Renders the triangle mesh
68      *
69      */
70     virtual void render();
71
72     /**
73      * @brief   Sets the normal buffer of the mesh
74      *
75      * @param normals The new normal buffer
76      */
```

```
77      void setNormalBuffer(float* normals) { m_normalBuffer = normals;}
78
79      /**
80       * @brief   Sets the vertex buffer of the mesh
81       *
82       * @param vertices  The new vertex buffer
83       * @param n         The number of vertices in the mesh
84       */
85      void setVertexBuffer(float* vertices, int n)
86      {
87          m_vertexBuffer = vertices;
88          m_numVertices = n;
89      }
90
91      /**
92       * @brief   Sets the index buffer of the mesh
93       *
94       * @param faces     The new index buffer
95       * @param n         The number of faces in the mesh
96       */
97
98      void setIndexBuffer(int* faces, int n)
99      {
100         m_indexBuffer = faces;
101         m_numFaces = n;
102     }
103
104     /**
105      * @brief Destroys the TriangleMesh object
106      *
107      */
108     virtual ~TriangleMesh();
109
110 protected:
111
112     /// Number of vertices
113     int          m_numVertices;
114
115     /// Number of faces
116     int          m_numFaces;
117
118     /// Vertexbuffer
119     floatPtr     m_vertexBuffer;
120
121     /// Normalbuffer
122     floatPtr     m_normalBuffer;
123
124     /// Indexbuffer
125     intPtr       m_indexBuffer;
126
127
128 };
129
130 } // namespace asteroids
131
132 #endif
```

rendering/TriangleMesh.hpp