7. Übungsblatt - C++

Henrik Gerdes, Manuel Eversmeyer
D10

12. Dezember 2018

```cpp
/**
 * @file ListTest.cpp
 * @author your name (you@domain.com)
 * @brief
 * @version 0.1
 * @date 2018-12-06
 *
 * @copyright Copyright (c) 2018
 *
 */

#include "List.hpp"
#include "Vector.hpp"


/**
 * @brief
 *
 * @param x
 */
void print_ints(int& x)
{
    std::cout << "Zahl in Liste: " << x << std::endl;
}

int main(int argc, char** argv)
{
    ///Creates a new List
    asteroids::List<int> IntList;

    ///Put some numbers in
    for(int i = 0; i < 10; i++)
    {
        IntList.insert(i);
    }


    void (*pointer)(int&);
    pointer = &print_ints;

    ///Do something with every emlement
    IntList.for_each(pointer);


```

```
45      //Test for VectorGen
46      // asteroids::Vector<float,3> test1(4.0,5.4,7.3);
47      // asteroids::Vector<float,3> test3(4.0,5.4,7.3);
48      // asteroids::Vector<int,2> test4(4,5);
49      // asteroids::Vector<int,2> test2(4,5);
50      // test2 + test4;
51      // test1 += test3;
52      // test1.printVector();
53
54      // std::cout << test1 * test3 << std::endl;
55
56
57
58      return 0;
59  }
```

ListTest.cpp

```
1   /*
2    * List.hpp
3    *
4    *  @date 02.12.2018
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
13  #ifndef LIST_H
14  #define LIST_H
15
16  namespace asteroids
17  {
18
19  /**
20   * @brief   A simple generic list class
21   */
22  template<typename T> class List
23  {
24     public:
25
26        /**
27         * @brief Constructs an empty list.
28         */
29        List<T>();
30        /**
31         * @brief  Destructor. Frees the generated nodes.
32         */
33        ~List<T>();
34        /**
35         * @brief Inserts an item into the list, i.e. a new node
36         *        constaining @ref item is created.
37         * @param item To be inserted
38         *
39         */
```

```
40        void insert(T item);
41
42        /**
43         * @brief Iterates over all items in the list and calls
44         * the given function @ref do_something(...) for
45         * every item stored in the list.
46         *
47         * @param do_something Function pointer to apply to all elements.
48         */
49        void for_each(void (*do_something)(T& item));
50
51    private:
52
53        /**
54         * @brief Struct to represent an inner node of the list.
55         */
56        struct Node {
57           Node* next;
58           T data;
59        };
60
61        // Root of the list
62        Node* m_list;
63 };
64
65 }
66
67 #include "List.tcc"
68 #endif
69 /* end of include guard: LIST_H */
```

List.hpp

```
1  /*
2   * List.tcc
3   *
4   *  @date 02.12.2018
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 #include <iostream>
13
14 namespace asteroids
15 {
16
17 template<typename T> List<T>::List()
18 {
19    m_list = NULL;
20 }
21 template<typename T> List<T>::~List()
22 {
23    if (m_list->next != NULL)
24    {
```

```
25          delete m_list->next;
26      }
27      delete m_list;
28  }
29  template<typename T> void List<T>::insert(T item)
30  {
31      if (m_list == NULL)
32      {
33          m_list = new Node;
34          m_list->data = item;
35          m_list->next = NULL;
36      }
37      else
38      {
39          Node* m = new Node;
40          m->data = item;
41          m->next = m_list;
42          m_list = m;
43      }
44  }
45  template<typename T> void List<T>::for_each(void (*do_something)(T& item))
46  {
47      Node* tmp = m_list;
48      while (tmp != NULL)
49      {
50          do_something(tmp->data);
51          tmp = tmp->next;
52      }
53  }
54
55  } // namespace asteroids
```

List.tcc

```
1   /**
2    *   @file Vector.hpp
3    *
4    *   @date 07.12.2018
5    *   @author Henrik Gerdes
6    *
7    */
8
9
10  #ifndef __Vector_HPP__
11  #define __Vector_HPP__
12
13  #include <iostream>
14  #include <cmath>
15
16  namespace asteroids
17  {
18
19
20  template<typename T, int L> class Vector
21  {
22      public:
23      Vector<T,L>();
24
```

```
25      /**
26       * @brief Construct a new Vector object
27       *
28       * @param x
29       * @param y
30       * @param z
31       */
32      Vector<T,L>(T x, T y ,T z = 0);
33
34      /**
35       * @brief Destroy the Vector object
36       */
37      ~Vector<T,L>();
38
39      /**
40       * @brief   Normalize a Vector
41       */
42      void normalize();
43
44      /**
45       * @brief   Assings a Vector to this object.
46       *          Performs a copy of vec.
47       *
48       * @param vec Vector to assign
49       */
50      void operator= (const Vector& vec);
51
52      /**
53       * @brief   Defines the vector addition
54       * @param vec vector
55       * @return vector
56       */
57      Vector operator+ (const Vector& vec) const;
58
59      /**
60       * @brief   Defines the vector subtraction
61       * @param vec vector
62       * @return vector
63       */
64      Vector operator- (const Vector& vec) const;
65
66      /**
67       * @brief   Construcs the scalar multiplication
68       * @param scale scalar
69       * @return vector
70       */
71      Vector operator* (const T& scale) const;
72
73      /**
74       * @brief   Defines the vector multiplication
75       * @param vec vector
76       * @return result (as T)
77       */
78      T  operator* (const Vector& vec) const;
79
80      /**
81       * @brief   Defines the access to a Vector value
82       * @param index wanted value
```

```
83          * @return vectorentry (as T)
84          */
85         T operator[] (const int& index) const;
86 
87         /**
88          * @brief   Defines the access to a Vector value
89          * @param index wanted value
90          * @return vectorentry (as T)
91          */
92         T& operator[] (const int& index);
93 
94         /**
95          * @brief   Defines the fast notation of vector addition
96          * @param v vector
97          */
98         void operator+= (const Vector& v);
99 
100        /**
101         * @brief   Defines the fast notation of vector subtraction
102         * @param v vector
103         */
104        void operator-= (const Vector& v);
105 
106        /**
107         * @brief Prints the Vector to stdout
108         */
109        void printVector();
110 
111        private:
112        ///To store the Vector-Values
113        T m[L];
114 
115 };
116 
117 typedef Vector<float,3> Vector3f;
118 typedef Vector<int,2> Vector2i;
119 
120 
121 }//asteroids
122 #include "Vector.tcc"
123 #endif
```

Vector.hpp

```
1  /**
2   * @file Vector.tcc
3   * @author Henrik Gerdes
4   * @brief   Template-Class for a Vector with Dimension between 2 and 3.
5   *          Types must implement *  + -/ and =
6   *
7   * @version 0.1
8   * @date 2018-12-07
9   *
10  * @copyright Copyright (c) 2018
11  *
12  */
13 
14 #include "Vector.hpp"
```

```
15
16 namespace asteroids
17 {
18     template<typename T, int L>Vector<T,L>::Vector()
19     {
20         static_assert(L<4 && L>1);
21         for(int i = 0; i < L; i++)
22         {
23             m[i] = 0;
24         }
25     }
26
27     template<typename T, int L>Vector<T,L>::Vector(T x, T y , T z)
28     {
29         static_assert(L<4 && L>1);
30
31         m[0] = x;
32         m[1] = y;
33
34         if(L == 3)
35         {
36             m[2] = z;
37         }
38     }
39
40     template<typename T, int L> void Vector<T,L>::normalize()
41     {
42         // Normalize the Vector3f
43         T mag2 = 0;
44         //For every Vector-Dimension
45         for(int i = 0; i < L; i++)
46         {
47             mag2 += m[i] * m[i];
48         }
49
50         //Sholud do the samle as the old class
51         if (fabs(mag2 - 1.0f) > 0.00001)
52         {
53             float mag = sqrt(mag2);
54             for(int i = 0; i < L; i++)
55             {
56                 m[i] /=mag;
57             }
58         }
59     }
60
61     template<typename T, int L> void Vector<T,L>::operator=(const Vector&
           vec)
62     {
63         for(int i = 0; i < L; i++)
64         {
65             m[i] = vec[i];
66         }
67     }
68
69     template<typename T, int L> Vector<T,L> Vector<T,L>::operator+(const
           Vector& vec) const
70     {
```

```
71          T tmp[3] = {0};
72          for(int i = 0; i < L; i++)
73          {
74              tmp[i] = m[i] + vec[i];
75          }
76          return Vector<T,L>(tmp[0], tmp[1], tmp[2]);
77
78      }
79
80      template<typename T, int L> void Vector<T,L>::operator+=(const Vector&
            vec)
81      {
82          *this = *this + vec;
83      }
84
85      template<typename T, int L> Vector<T,L> Vector<T,L>::operator-(const
            Vector& vec) const
86      {
87          T tmp[3] = {0};
88          for(int i = 0; i < L; i++)
89          {
90              tmp[i] = this[i] - vec[i];
91          }
92          return Vector<T,L>(tmp[0], tmp[1], tmp[2]);
93
94      }
95
96      template<typename T, int L> void Vector<T,L>::operator-=(const Vector&
            vec)
97      {
98          *this = *this - vec;
99      }
100
101     template<typename T, int L> Vector<T,L> Vector<T,L>::operator*(const T&
            scale) const
102     {
103         return Vector<T,L>(m[0] * scale, m[1] * scale, m[2] * scale);
104     }
105
106     template<typename T, int L> T Vector<T,L>::operator*(const Vector& vec)
            const
107     {
108         T res = 0;
109         for(int i = 0; i < L; i++)
110         {
111             res += m[i]*vec[i];
112         }
113         return res;
114     }
115
116     template<typename T, int L> T Vector<T,L>::operator[](const int& index)
            const
117     {
118         if(index >= L || index < 0)
119         {
120             throw std::invalid_argument("Außerhalb der Vector-Dimension");
121         }
122         else
```

```
123         {
124             return m[index];
125         }
126     }
127 
128     template<typename T, int L> T& Vector<T,L>::operator[](const int& index
        )
129     {
130         if(index >= L || index < 0)
131         {
132             throw std::invalid_argument("Außerhalb der Vector-Dimension");
133         }
134         else
135         {
136             return m[index];
137         }
138     }
139 
140     template<typename T, int L> void Vector<T,L>::printVector()
141     {
142         std::cout << "Vector-Daten: " << std::flush;
143         for(int i = 0; i < L; i++)
144         {
145             std::cout << m[i] << " " << std::flush;
146         }
147         std::cout << std::endl;
148     }
149 
150     template<typename T, int L>Vector<T,L>::~Vector()
151     {
152         //Do nothing
153     }
154 
155 
156 }
```

Vector.tcc

```cpp
/*
 *  MainWindow.cpp
 *
 *  Created on: Nov. 04 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 "MainWindow.hpp"
#include "Circle.hpp"
#include "Rectangle.hpp"
#include "Sphere.hpp"
#include "List.hpp"

#include <iostream>

namespace asteroids
{

MainWindow::MainWindow(
    const std::string& title,
    const std::string& plyname, int w, int h)
    : m_camera(Vector3f(0.0f, 0.0f, -700.0f), 0.05f, 5.0f)
{
    // Save width and height
    m_height = h;
    m_width = w;

    // Setup window
    m_sdlWindow = SDL_CreateWindow(
        "SDL Main Window",
        SDL_WINDOWPOS_CENTERED, SDL_WINDOWPOS_CENTERED,
        m_width, m_height, SDL_WINDOW_OPENGL);

    if(!m_sdlWindow)
    {
        std::cout << "MainWindow: Unable to create SDL window" << std::endl;
    }

    m_sdlGlcontext = SDL_GL_CreateContext(m_sdlWindow);

    if(!m_sdlGlcontext)
    {
        std::cout << "MainWindow: Unable to creade SDL GL context" << std::endl;
    }

    if(m_sdlWindow && m_sdlGlcontext)
    {
        // Set our OpenGL version.
        // SDL_GL_CONTEXT_CORE gives us only the newer version, deprecated functions are disabled
```

```
54          SDL_GL_SetAttribute(SDL_GL_CONTEXT_PROFILE_MASK,
                SDL_GL_CONTEXT_PROFILE_CORE);
55
56          // 3.2 is part of the modern versions of OpenGL,
57          // but most video cards whould be able to run it
58          SDL_GL_SetAttribute(SDL_GL_CONTEXT_MAJOR_VERSION, 3);
59          SDL_GL_SetAttribute(SDL_GL_CONTEXT_MINOR_VERSION, 2);
60
61          // Turn on double buffering with a 24bit Z buffer.
62          // You may need to change this to 16 or 32 for your system
63          SDL_GL_SetAttribute(SDL_GL_DOUBLEBUFFER, 1);
64
65          // This makes our buffer swap syncronized with the monitor's
                vertical refresh
66          SDL_GL_SetSwapInterval(1);
67
68  #ifndef __APPLE__
69          glewExperimental = GL_TRUE;
70          glewInit();
71  #endif
72          SDL_GL_SwapWindow(m_sdlWindow);
73
74          // Init OpenGL projection matrix
75          glClearColor(0.0, 0.0, 0.0, 1.0);
76          float ratio = m_width * 1.0 / m_height;
77          glMatrixMode(GL_PROJECTION);
78          glLoadIdentity();
79          glViewport(0, 0, m_width, m_height);
80          gluPerspective(45, ratio, 1, 10000);
81
82          // Enter model view mode
83          glMatrixMode(GL_MODELVIEW);
84      }
85
86      // Load model
87      m_mesh = new TriangleMesh(plyname);
88  }
89
90  int MainWindow::width()
91  {
92      return m_width;
93  }
94
95  int MainWindow::height()
96  {
97      return m_height;
98  }
99
100 void MainWindow::render(Renderable*& obj)
101 {
102     obj->render();
103 }
104
105 void MainWindow::execute()
106 {
107     int x = m_width / 2;
108     int y = m_height / 2;
109     int w = 200;
```

```
110     int h = 100;
111
112     List<Renderable*> renderOBJ;
113
114     Circle circle(this, Vector2i(m_width / 2,m_height / 2), 100, 20);
115     circle.setColor(1.0, 0.0, 0.0);
116     Rectangle rect(this, Vector2i(x - w / 2, y - h / 2), Vector2i(w, h));
117     rect.setColor(0.0, 1.0, 2.0);
118     Sphere sphere(Vector3f(0, 0, 0), 10);
119
120     ///List of renderable shapes
121     renderOBJ.insert(&circle);
122     renderOBJ.insert(&rect);
123     renderOBJ.insert(&sphere);
124     renderOBJ.insert(m_mesh);
125
126     if(m_mesh && m_sdlWindow && m_sdlGlcontext)
127     {
128         bool loop = true;
129         const Uint8* keyStates;
130
131         while (loop)
132         {
133             // Clear background
134             glClear(GL_COLOR_BUFFER_BIT );
135
136             // Apply camera, also loads indentity matrix
137             m_camera.apply();
138
139             // Markers for mouse buttons
140             bool r_pressed = false;
141             bool l_pressed = false;
142
143             // Handle events
144             SDL_Event event;
145             while (SDL_PollEvent(&event))
146             {
147                 switch(event.type)
148                 {
149                     // Window was closed, exit main loop
150                     case SDL_QUIT:
151                         loop = false;
152                         break;
153                     //Handle mouse motion
154                     case SDL_MOUSEMOTION:
155
156                             // Check if left button is pressed
157                             if(event.motion.state & SDL_BUTTON_LMASK)
158                             {
159                                 l_pressed = true;
160                             }
161
162                             // Check if right button is pressed
163                             if(event.motion.state & SDL_BUTTON_RMASK)
164                             {
165                                 r_pressed = true;
166                             }
167
```

```
                    // Handle motion for pressed L button while R is not
                    // pressed
                    if(l_pressed & !r_pressed)
                    {
                        if(event.motion.xrel > -3)
                        {
                            m_camera.turn(Camera::LEFT);
                        }
                        if(event.motion.xrel < 3)
                        {
                            m_camera.turn(Camera::RIGHT);
                        }
                        if(event.motion.yrel > 3)
                        {
                            m_camera.turn(Camera::UP);
                        }
                        if(event.motion.yrel < -3)
                        {
                            m_camera.turn(Camera::DOWN);
                        }
                    }

                    // Handle motion for pressed R button while L is not
                    // pressed
                    if(r_pressed & !l_pressed)
                    {
                        if(event.motion.xrel > 3)
                        {
                            m_camera.move(Camera::RIGHT);
                        }
                        if(event.motion.xrel < -3)
                        {
                            m_camera.move(Camera::LEFT);
                        } if(event.motion.yrel > 3)
                        {
                            m_camera.move(Camera::FORWARD);
                        }
                        if(event.motion.yrel < -3)
                        {
                            m_camera.move(Camera::BACKWARD);
                        }
                    }
                break;
            default:
                break;
        }

        // Get keyboard states and handle model movement
        keyStates = SDL_GetKeyboardState(NULL);

        if(keyStates[SDL_SCANCODE_UP])
        {
            m_mesh->rotate(TriangleMesh::YAW, 0.05);
        }
        if(keyStates[SDL_SCANCODE_DOWN])
        {
```

```
224                         m_mesh->rotate(TriangleMesh::YAW, -0.05);
225                     }
226                     if(keyStates[SDL_SCANCODE_LEFT])
227                     {
228                         m_mesh->rotate(TriangleMesh::ROLL, 0.05);
229                     }
230                     if(keyStates[SDL_SCANCODE_RIGHT])
231                     {
232                         m_mesh->rotate(TriangleMesh::ROLL, -0.05);
233                     }
234                     if(keyStates[SDL_SCANCODE_W])
235                     {
236                         m_mesh->move(TriangleMesh::ACCEL, 3);
237                     }
238                     if(keyStates[SDL_SCANCODE_S])
239                     {
240                         m_mesh->move(TriangleMesh::ACCEL, -3);
241                     }
242                     if(keyStates[SDL_SCANCODE_A])
243                     {
244                         m_mesh->move(TriangleMesh::STRAFE, 3);
245                     }
246                     if(keyStates[SDL_SCANCODE_D])
247                     {
248                         m_mesh->move(TriangleMesh::STRAFE, -3);
249                     }
250                 }
251 
252                 renderOBJ.for_each(render);
253 
254 
255                 // Bring up back buffer
256                 SDL_GL_SwapWindow(m_sdlWindow);
257             }
258         }
259 }
260 
261 MainWindow::~MainWindow()
262 {
263     // Delete model
264     if(m_mesh)
265     {
266         delete m_mesh;
267     }
268 
269     // Cleanup SDL stuff
270     SDL_GL_DeleteContext(m_sdlGlcontext);
271 
272     // Destroy our window
273     SDL_DestroyWindow(m_sdlWindow);
274 
275     // Shutdown SDL 2
276     SDL_Quit();
277 }
278 
279 } // namespace asteroids
```

MainWindow.cpp