Einführung in C++ – Übung 11
Testatgruppe A (Isaak)

Rasmus Diederichsen

6. Januar 2015

Aufgabe 11.1 STL-Funktionen

1.

Die Lösung hierzu findet sich in den Slides.

src/Summer.hpp

```
/**
 * @file Summer.hpp
 * @author rdiederichse@uos.de
 * @brief Contains a unary_function struct definition to sum int
     vector elements.
 */
#ifndef SUMMER_H
#define SUMMER_H

#include <functional>

struct Summer : public std::unary_function<int,int>
{
   Summer() : sum(0) {}
   void operator()(int i)
   {
      sum += i;
   }
   int sum;
};

#endif /* end of include guard: SUMMER_H */
```

src/SummerTest.cpp

```
#include "Summer.hpp"
#include <vector>
#include <algorithm>
#include <iostream>

int main(int argc, const char *argv[])
{
```

```
    std::vector<int> v {1,2,3,4,5};
    Summer s = std::for_each(v.begin(), v.end(), Summer());
    std::cout << "Sum is " << s.sum << std::endl;

    return 0;
}
```

2.

src/aufg1_2.cpp

```
#include <functional> // for bind2nd, equal_to, modulus
#include <iostream>
#include <vector>
#include <iterator> // for ostream_iterator
#include <algorithm> // for remove_if
#include <ext/functional> // non-standard gnu extension for
    compose1

using namespace std;
int main(int argc, const char *argv[])
{
  vector<int> v;
  v.push_back(1);
  v.push_back(4);
  v.push_back(2);
  v.push_back(8);
  v.push_back(5);
  v.push_back(7);

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

  // This takes the binary modulus function (g), binds its second
      argument to 2,
  // binds the second argument of equal_to (f) to 0 and builds a
      unary function
  // object which given x returns f(g(x)) = equal_to(modulus(x,2)
      ,0), meaning it
  // determines whether x is even.
  vector<int>::iterator new_end =
    remove_if(v.begin(), v.end(),
              __gnu_cxx::compose1(bind2nd(equal_to<int>(), 0), //
                  namespaces were missing
                    bind2nd(modulus<int>(), 2)));

  // wrong iterator as second argument was used. Since remove_if
      alters the
  // sequence it iterates by replacing the discarded elements by
      their
  // respective next element which is accepted. As such one must
      use the
  // returned iterator which points after the last element up to
      which the
  // sequence is valid.
  copy(v.begin(), new_end, ostream_iterator<int>(cout, " "));
```

```
37    cout << endl;
38
39     return 0;
40  }
```

Aufgabe 11.2 Threads in C++

src/rendering/Bullet.cpp

```
1   /**
2    * Bullet.cpp
3    *
4    */
5
6   #include "Bullet.hpp"
7   #include <iostream>
8   #include <functional>
9   using namespace std;
10
11  namespace asteroids
12  {
13
14     const int Bullet::m_lifetime = 9000;
15     const float Bullet::m_bulletSpeed = 1.;
16
17     Bullet::Bullet(Vertex<float> fighterPosition, Vertex<float>
           m_fighterAxis)
18     {
19        m_alive = false;
20        this->m_fighterAxis = m_fighterAxis * -1; // xAxis points
              into wrong direcrion
21        this->m_position = fighterPosition;
22     }
23
24     Bullet::~Bullet()
25     {
26        if (m_thread) m_thread->join(); // what happens if I delete
              before termination? Probably mayhem.
27        delete m_thread;
28     }
29
30     bool Bullet::isAlive()
31     {
32        return m_alive;
33     }
34
35     void Bullet::stop()
36     {
37        if (m_thread) m_thread->join();
38        m_alive = false;
39     }
40
41     void Bullet::start()
42     {
43        m_thread = new std::thread(&Bullet::run, this);
44        m_alive = true;
```

```
45      }
46
47      void Bullet::run()
48      {
49          int i = 0;
50          while (i++ < m_lifetime)
51          {
52              m_position += m_fighterAxis * m_bulletSpeed;
53              std::this_thread::sleep_for(std::chrono::milliseconds(1));
54          }
55          m_alive = false;
56      }
57
58      void Bullet::render()
59      {
60          // Compute transformation matrix
61          computeMatrix();
62          // Push old transformation of the OpenGL matrix stack and
63          // start rendering the bullet in according to the
64          // internal transformation matrix
65          glPushMatrix();
66          glMultMatrixf(m_transformation);
67          glutSolidSphere(10,16,16);
68          // Pop transformation matrix of this object
69          // to restore the previous state of the OpenGL
70          // matrix stack
71          glPopMatrix();
72      }
73
74  } // namespace asreroids
```

Es ist notwendig, das `Bullet` nach jeder Bewegung kurz anzuhalten, weil sonst die Schleife praktisch instantan zu Ender wäre und man die Kugel nicht sehen würde.

src/rendering/Fighter.cpp

```
1   /**
2    * Fighter.cpp
3    *
4    */
5   #include <iostream>
6
7
8   #include "Fighter.hpp"
9
10  namespace asteroids
11  {
12
13  void Fighter::shoot()
14  {
15      Bullet* b = new Bullet(m_position, m_xAxis);
16      b->start();
17      m_bullets.push_back(b);
18  }
19
20
21  void Fighter::render()
22  {
```

```
23      // Render the fighter
24      TexturedMesh::render();
25
26      std::vector<Bullet*>::iterator begin = m_bullets.begin();
27      while (begin != m_bullets.end())
28      {
29          if ((*begin)->isAlive()) (*begin)->render();
30          else m_bullets.erase(begin);
31          begin++;
32      }
33  }
34
35  } // namespace asteroids
```