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

Rasmus Diederichsen

14. Januar 2015

Aufgabe 11.1 STL-Funktionen

1.

Die Lösung hierzu findet sich in den Slides.

src/Summer.hpp

```cpp
/**
 * @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

```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, "␣"));
  cout << endl;

   return 0;
}
```

Aufgabe 11.2 Threads in C++

src/rendering/Bullet.cpp

```
/**
 * Bullet.cpp
 *
 */

#include "Bullet.hpp"
#include <iostream>
#include <functional>
using namespace std;

namespace asteroids
{

   const int Bullet::m_lifetime = 9000;
   const float Bullet::m_bulletSpeed = 1.;

   Bullet::Bullet(Vertex<float> fighterPosition, Vertex<float>
       m_fighterAxis)
   {
      m_alive = false;
      this->m_fighterAxis = m_fighterAxis * -1; // xAxis points
          into wrong direction
      this->m_position = fighterPosition;
   }

   Bullet::~Bullet()
   {
      if (m_thread)
      {
         m_thread->join(); // what happens if I delete before
             termination? Probably mayhem.
         delete m_thread;
      }
      cout << "bullet␣destroyed" << endl;
   }

   bool Bullet::isAlive()
   {
      return m_alive;
   }

   void Bullet::stop()
   {
      if (m_thread) m_thread->join();
      m_alive = false;
   }

   void Bullet::start()
   {
      m_thread = new std::thread(&Bullet::run, this);
      m_alive = true;
   }

   void Bullet::run()
   {
      int i = 0;
      while (i++ < m_lifetime)
      {
         m_position += m_fighterAxis * m_bulletSpeed;
         std::this_thread::sleep_for(std::chrono::milliseconds(1));
      }
      m_alive = false;
   }

   void Bullet::render()
   {
      // Compute transformation matrix
      computeMatrix();
      // Push old transformation of the OpenGL matrix stack and
      // start rendering the bullet in according to the
      // internal transformation matrix
      glPushMatrix();
      glMultMatrixf(m_transformation);
      glutSolidSphere(10,16,16);
      // Pop transformation matrix of this object
      // to restore the previous state of the OpenGL
      // matrix stack
      glPopMatrix();
   }

} // 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

```
/**
 * Fighter.cpp
 *
 */
#include <iostream>


#include "Fighter.hpp"

namespace asteroids
{

void Fighter::shoot()
{
   Bullet* b = new Bullet(m_position, m_xAxis);
   b->start();
   m_bullets.push_back(b);
}


void Fighter::render()
{
    // Render the fighter
    TexturedMesh::render();

    std::vector<Bullet*>::iterator begin = m_bullets.begin();
    while (begin != m_bullets.end())
    {
       if ((*begin)->isAlive())
       {
          (*begin)->render();
          begin++;
       }
       else
       {
          /* delete *begin; */
          begin = m_bullets.erase(begin);
       }
    }
}

} // namespace asteroids
```