Einführung in C++ – Übung 9
Testatgruppe A (Isaak)

Rasmus Diederichsen

8. Dezember 2014

Aufgabe 9.1 Exception-Handling

Fehlerabfragen machen meines Erachtens nur beim Aufruf von `Vertex::normalize()` Sinn, und auch da nur beschränkt. Alle Arrayzugriffe durch den `[]`-operator geschehen durch Literale, weshalb hier ebenfalls kein `try`-Block nötig ist.

src/math/Vertex.cpp

```
#include "Vertex.hpp"
#include <stdio.h>
#include <string>
#include "exceptions/DivisionByZeroException.hpp"
#include "exceptions/OutOfBoundsException.hpp"

namespace asteroids {

Vertex::Vertex()
{
        // Default values
        x = y = z = 0.0;
}


Vertex::Vertex(float _x, float _y, float _z)
{
        // Set the given values
        x = _x;
        y = _y;
        z = _z;
}

void Vertex::normalize()
{
        // Normalize the vector
        float mag2 = x * x + y * y + z * z;
        if (fabs(mag2 - 1.0f) > TOLERANCE)
        {
                float mag = sqrt(mag2);
      // to_string is c++ 11
```

```
32          if (mag == .0f)
33              throw DivisionByZeroException("Vector to normalise has 0
                    length.");
34          x /= mag;
35          y /= mag;
36          z /= mag;
37      }
38  }
39
40  Vertex Vertex::operator+(const Vertex vec) const
41  {
42      // Add value to value
43      float tx = x + vec.x;
44      float ty = y + vec.y;
45      float tz = z + vec.z;
46      return Vertex(tx, ty, tz);
47  }
48
49  Vertex Vertex::operator-(const Vertex vec) const
50  {
51      // Subtract value from value
52      float tx = x - vec.x;
53      float ty = y - vec.y;
54      float tz = z - vec.z;
55      return Vertex(tx, ty, tz);
56  }
57
58
59  float Vertex::operator[](const int &index) const
60  {
61
62      // Get the wanted value
63      if(index == 0)
64      {
65          return x;
66      }
67
68      if(index == 1)
69      {
70          return y;
71      }
72
73      if(index == 2)
74      {
75          return z;
76      }
77
78      // to_string is c++ 11
79      throw OutOfBoundsException("Wrong index " + std::to_string(index
            ));
80  }
81
82
83  float& Vertex::operator[](const int &index)
84  {
85
86      if(index == 0)
```

```
87      {
88          return x;
89      }
90
91      if(index == 1)
92      {
93          return y;
94      }
95
96      if(index == 2)
97      {
98          return z;
99      }
100
101     throw OutOfBoundsException("Wrong index " + std::to_string(index
            ));
102 }
103
104
105
106
107 float Vertex::operator*(const Vertex vec) const
108 {
109     // Calculate the result
110     return (x * vec.x + y * vec.y + z * vec.z);
111 }
112
113 Vertex Vertex::operator*(float scale) const
114 {
115     // Calculate the result
116     float tx = x * scale;
117     float ty = y * scale;
118     float tz = z * scale;
119     return Vertex(tx, ty, tz);
120 }
121
122 void Vertex::operator+=(Vertex v)
123 {
124     // Add value to value
125     x += v.x;
126     y += v.y;
127     z += v.z;
128 }
129
130 } // namespace cpp2014
```

src/math/Quaternion.cpp

```
1  #include "Quaternion.hpp"
2  #include "exceptions/DivisionByZeroException.hpp"
3
4  namespace asteroids
5  {
6
7      Quaternion::Quaternion()
8      {
9          // Default Quaternion
10         x = 1.0;
```

```
11          y = 0.0;
12          z = 0.0;
13          w = 0.0;
14      }
15
16      Quaternion::~Quaternion()
17      {
18          // Do nothing
19      }
20
21      Quaternion::Quaternion(Vertex vec, float angle)
22      {
23          // Calculate the quaternion
24          fromAxis(vec, angle);
25      }
26
27      Quaternion::Quaternion(float _x, float _y, float _z, float
            _angle)
28      {
29          // Set the values
30          x = _x;
31          y = _y;
32          z = _z;
33          w = _angle;
34      }
35
36      Quaternion::Quaternion(float* vec, float _w)
37      {
38          // Set the values
39          x = vec[0];
40          y = vec[1];
41          z = vec[2];
42          w = _w;
43      }
44
45      void Quaternion::fromAxis(Vertex axis, float angle)
46      {
47          float sinAngle;
48          angle *= 0.5f;
49
50          // Create a copy of the given vector and normalize the new
                vector
51          Vertex vn(axis.x, axis.y, axis.z);
52          try
53          {
54              vn.normalize();
55          } catch (DivisionByZeroException &divex)
56          {
57              std::cout << divex.what() << std::endl;
58          }
59
60          // Calculate the sinus of the given angle
61          sinAngle = sin(angle);
62
63          // Get the quaternion
64          x = (vn.x * sinAngle);
65          y = (vn.y * sinAngle);
```

```
66          z = (vn.z * sinAngle);
67          w = cos(angle);
68      }
69
70      Quaternion Quaternion::getConjugate()
71      {
72          // Conjugate the given quaternion
73          return Quaternion(-x, -y, -z, w);
74      }
75
76
77      Quaternion Quaternion::operator* (const Quaternion rq)
78      {
79          // Calculate the new quaternion
80          return Quaternion(w * rq.x + x * rq.w + y * rq.z - z * rq.y,
81                  w * rq.y + y * rq.w + z * rq.x - x * rq.z,
82                  w * rq.z + z * rq.w + x * rq.y - y * rq.x,
83                  w * rq.w - x * rq.x - y * rq.y - z * rq.z);
84      }
85
86      Vertex Quaternion::operator* (Vertex vec)
87      {
88          // Copy the vector and normalize the new vector
89          Vertex vn(vec);
90          try
91          {
92              vn.normalize();
93          } catch (DivisionByZeroException &divex)
94          {
95              std::cout << divex.what() << std::endl;
96          }
97
98          // Fill the first quaternion and...
99          Quaternion vecQuat, resQuat;
100         vecQuat.x = vn.x;
101         vecQuat.y = vn.y;
102         vecQuat.z = vn.z;
103         vecQuat.w = 0.0f;
104
105         // calculate the new quaternion
106         resQuat = vecQuat * getConjugate();
107         resQuat = *this * resQuat;
108         return (Vertex(resQuat.x, resQuat.y, resQuat.z));
109     }
110
111 }
```

Aufgabe 9.2 Timestamps und Logging

Timestamp

src/time/Timestamp.hpp

```
1  /**
2   * @file Timestamp.hpp
```

```
 * @author Rasmus Diederichsen (rdiederichse@uos.de)
 * @version 08.12.2014
 */


#ifndef TIMESTAMP_H

#define TIMESTAMP_H

#include <iostream>
#include <sys/time.h>

namespace asteroids
{
   /**
    * @class Timestamp
    * @brief Represents a point in time.
    */
   class Timestamp
   {
      public:
         /**
          * @brief Constructor. Time is initialised to current
              system time.
          */
         Timestamp();

         /**
          * @brief Get current system time.
          * @return Current system time in milliseconds (from UNIX
              epoch)
          */
         unsigned long getCurrentTimeInMs() const;

         /**
          * @brief Get time elapsed since instance creation.
          * @return The time elapsed since instance creation in
              milliseconds
          * (from UNIX epoch)
          */
         unsigned long getElapsedTimeInMs() const;

         /**
          * @see Timestamp::getCurrentTimeInMs()
          */
         unsigned long getCurrentTimeInS() const;

         /**
          * @see Timestamp::getElapsedTimeInMs()
          */
         unsigned long getElapsedTimeInS() const;

         /**
          * @brief Reset the timer to current system time.
          */
         void resetTimer();

```

```
57          /**
58           * @brief Get string representation of time elapsed since
                 creation.
59           * @return The elapsed time as a string.
60           */
61          std::string getElapsedTime() const;
62
63          /**
64           * @brief Operator to print to a stream.
65           */
66          friend std::ostream& operator<<(std::ostream& os, const
                Timestamp& ts); // why is friend necessary?
67
68       private:
69          unsigned long m_startTime;
70    };
71 } /* namespace asteroids */
72
73 #endif /* end of include guard: TIMESTAMP_H */
```

src/time/Timestamp.cpp

```
1  #include "Timestamp.hpp"
2  #include <cstddef>
3  #include <stdexcept>
4  #include <sstream>
5
6  namespace asteroids
7  {
8
9     Timestamp::Timestamp()
10    {
11       resetTimer();
12    }
13    unsigned long Timestamp::getCurrentTimeInMs() const
14    {
15       struct timeval tv;
16       struct timezone tz;
17       if (gettimeofday(&tv, &tz) == -1)
18          throw std::runtime_error("Error while attempting to get
                system time.");
19       return 1000 * tv.tv_sec + (unsigned long) (tv.tv_usec / 1000)
             ;
20    }
21    unsigned long Timestamp::getElapsedTimeInMs() const
22    {
23       return getCurrentTimeInMs() - m_startTime;
24    }
25    unsigned long Timestamp::getCurrentTimeInS() const
26    {
27       return getCurrentTimeInMs() / 1000;
28    }
29    unsigned long Timestamp::getElapsedTimeInS() const
30    {
31       return getElapsedTimeInMs() / 1000;
32    }
33    void Timestamp::resetTimer()
34    {
```

```
35          m_startTime = getCurrentTimeInMs();
36      }
37      std::string Timestamp::getElapsedTime() const
38      {
39          unsigned long elapsed = getElapsedTimeInMs();
40          unsigned long hours = elapsed / (1000 * 60 * 60);
41          elapsed -= hours * 1000 * 60 * 60;
42          unsigned long minutes = elapsed / (1000 * 60);
43          elapsed -= minutes * 1000 * 60;
44          unsigned long seconds = elapsed / 1000;
45          elapsed -= seconds;
46          unsigned long milliseconds = elapsed;
47          char buffer[17];
48          buffer[16] = '\0'; // necessary?
49          sprintf(buffer, "[%02lu:%02lu:%02lu - %03lu]", hours, minutes
                , seconds, milliseconds);
50          return std::string(buffer);
51      }
52      std::ostream& operator<<(std::ostream& os, const Timestamp& ts)
53      {
54          os << ts.getElapsedTime();
55          return os;
56      }
57  } /* namespace ast */
```

Logger

src/logging/Logger.hpp

```
1   /**
2    * @file Logger.hpp
3    * @author Rasmus Diederichsen (rdiederichse@uos.de)
4    * @version 08.12.2014
5    */
6
7   #ifndef LOGGER_H
8
9   #define LOGGER_H
10
11  #include "time/Timestamp.hpp"
12  #include <ostream>
13
14  namespace asteroids
15  {
16      /**
17       * @class Logger
18       * @brief Singleton Class to log program events
19       */
20      class Logger
21      {
22          public:
23              /**
24               * @brief Get the singleton instance
25               * @return the singleton
26               */
27              static Logger& instance();
28
```

```
29          /**
30           * @brief specify destination
31           * @param filename File to which log should go.
32           */
33          void setOutputFile(std::string filename);
34
35          /**
36           * @brief reset loggin to stdout
37           */
38          void setOutputToStdout();
39
40          /**
41           * @brief Print log message.
42           * @param s Message to log.
43           */
44          Logger& operator<<(const std::string& s);
45      private:
46          static Logger* logger;
47          static Timestamp stamp;
48          std::ostream* out;
49          Logger();
50          Logger(const Logger& l);
51          void operator=(const Logger& l);
52
53  };
54  } /* namespace asteroids */
55
56  #endif /* end of include guard: LOGGER_H */
```

src/logging/Logger.cpp

```
1   #include "Logger.hpp"
2   #include <iostream>
3   #include <fstream>
4   #include <cstddef>
5
6   namespace asteroids
7   {
8      Timestamp Logger::stamp;
9      Logger* Logger::logger = NULL;
10     Logger::Logger()
11     {
12        setOutputToStdout();
13     }
14
15     Logger& Logger::operator<<(const std::string& s)
16     {
17        *out << stamp << " - " << s << std::endl;
18        return *this;
19     }
20
21     void Logger::setOutputFile(std::string filename)
22     {
23        std::ofstream file(filename.c_str());
24        out = &file;
25     }
26
27     void Logger::setOutputToStdout()
```

```
28      {
29          out = &std::cout;
30      }
31
32      Logger& Logger::instance()
33      {
34          if (logger == NULL)
35              logger = new Logger;
36          return *logger;
37      }
38  } /* namespace asteroids */
```