

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

```
1  #include "Vertex.hpp"
2  #include <stdio.h>
3  #include <string>
4  #include "exceptions/DivisionByZeroException.hpp"
5  #include "exceptions/OutOfBoundsException.hpp"
6
7  namespace asteroids {
8
9  Vertex::Vertex()
10 {
11     // Default values
12     x = y = z = 0.0;
13 }
14
15
16 Vertex::Vertex(float _x, float _y, float _z)
17 {
18     // Set the given values
19     x = _x;
20     y = _y;
21     z = _z;
22 }
23
24 void Vertex::normalize()
25 {
26     // Normalize the vector
27     float mag2 = x * x + y * y + z * z;
28     if (fabs(mag2 - 1.0f) > TOLERANCE)
29     {
30         float mag = sqrt(mag2);
31         // to_string is c++ 11
```

```

32         if (mag == .0f)
33             throw DivisionByZeroException("Vector to normalise has 0
34                                     length.");
35         x /= mag;
36         y /= mag;
37         z /= mag;
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     float Vertex::operator[](const int &index) const
59     {
60
61         // Get the wanted value
62         if(index == 0)
63         {
64             return x;
65         }
66
67         if(index == 1)
68         {
69             return y;
70         }
71
72         if(index == 2)
73         {
74             return z;
75         }
76
77         // to_string is c++ 11
78         throw OutOfBoundsException("Wrong index" + std::to_string(index
79                                     ));
80     }
81
82     float& Vertex::operator[](const int &index)
83     {
84
85         if(index == 0)
86

```

```

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

```

```

3  * @author Rasmus Diederichsen (rdiederichse@uos.de)
4  * @version 08.12.2014
5  */
6
7
8  #ifndef TIMESTAMP_H
9
10 #define TIMESTAMP_H
11
12 #include <iostream>
13 #include <sys/time.h>
14
15 namespace asteroids
16 {
17     /**
18      * @class Timestamp
19      * @brief Represents a point in time.
20      */
21     class Timestamp
22     {
23     public:
24         /**
25          * @brief Constructor. Time is initialised to current
26          *          system time.
27          */
28         Timestamp();
29
30         /**
31          * @brief Get current system time.
32          * @return Current system time in milliseconds (from UNIX
33          *          epoch)
34          */
35         unsigned long getCurrentTimeInMs() const;
36
37         /**
38          * @brief Get time elapsed since instance creation.
39          * @return The time elapsed since instance creation in
40          *          milliseconds
41          *          (from UNIX epoch)
42          */
43         unsigned long getElapsedTimeInMs() const;
44
45         /**
46          * @see Timestamp::getCurrentTimeInMs()
47          */
48         unsigned long getCurrentTimeInS() const;
49
50         /**
51          * @see Timestamp::getElapsedTimeInMs()
52          */
53         unsigned long getElapsedTimeInS() const;
54
55         /**
56          * @brief Reset the timer to current system time.
57          */
58         void resetTimer();
59

```

```

57     /**
58      * @brief Get string representation of time elapsed since
59      *        creation.
60      * @return The elapsed time as a string.
61      */
62     std::string getElapsedTime() const;
63
64     /**
65      * @brief Operator to print to a stream.
66      */
67     friend std::ostream& operator<<(std::ostream& os, const
68         Timestamp& ts); // why is friend necessary?
69
70     private:
71         unsigned long m_startTime;
72     };
73 } /* namespace asteroids */
74
75 #endif /* end of include guard: TIMESTAMP_H */

```

src/time/Timestamp.cpp

```

1  #include "Timestamp.hpp"
2  #include <cstdint>
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
19                 system time.");
20         return 1000 * tv.tv_sec + (unsigned long) (tv.tv_usec / 1000)
21             ;
22     }
23     unsigned long Timestamp::getElapsedTimeInMs() const
24     {
25         return getCurrentTimeInMs() - m_startTime;
26     }
27     unsigned long Timestamp::getCurrentTimeInS() const
28     {
29         return getCurrentTimeInMs() / 1000;
30     }
31     unsigned long Timestamp::getElapsedTimeInS() const
32     {
33         return getElapsedTimeInMs() / 1000;
34     }
35     void Timestamp::resetTimer()
36     {

```

```

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
50             , seconds, milliseconds);
51     return std::string(buffer);
52 }
53 std::ostream& operator<<(std::ostream& os, const Timestamp& ts)
54 {
55     os << ts.getElapsedTime();
56     return os;
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 login 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
46      /**
47       * @brief Desctructor
48       */
49      ~Logger();
50      private:
51      static Logger* logger;
52      static Timestamp stamp;
53      std::ostream* out;
54      Logger();
55      Logger(const Logger& l);
56      void operator=(const Logger& l);
57
58  };
59  } /* namespace asteroids */
60
61  #endif /* end of include guard: LOGGER_H */

```

src/logging/Logger.cpp

```

1  #include "Logger.hpp"
2  #include <iostream>
3  #include <fstream>
4  #include <cstdint>
5  #include <typeinfo>
6
7  namespace asteroids
8  {
9      Timestamp Logger::stamp;
10     Logger* Logger::logger = NULL;
11     Logger::Logger()
12     {
13         setOutputToStdout();
14     }
15
16     Logger& Logger::operator<<(const std::string& s)
17     {
18         *out << stamp;
19         *out << "□-□";
20         *out << s << std::endl;
21         return *this;
22     }

```

```

23
24     void Logger::setOutputFile(std::string filename)
25     {
26         out = new std::ofstream(filename.c_str());
27     }
28
29     void Logger::setOutputToStdout()
30     {
31         out = &std::cout;
32     }
33
34     Logger& Logger::instance()
35     {
36         if (logger == NULL)
37             logger = new Logger;
38         return *logger;
39     }
40
41     Logger::~Logger()
42     {
43         if (typeid(*out) == typeid(std::ofstream))
44         {
45             ((std::ofstream*)out)->close();
46             delete out;
47         }
48     }
49 } /* namespace asteroids */

```