Exercises week 1

Klaas Isaac Bijlsma s2394480

David Vroom s2309939

November 20, 2017

Exercise 1

Attain some familiarity with the way functions are selected from namespaces

We used the following code,

Exercise 2

Learn why streams can be used to determine the truth values of conditions, but not to assign values to bool variables.

Note: The code given in the exercise is incomplete, and therefore won't compile even without the intended mistake. So first of all we state the following code as a starting point:

header.ih

```
1 #include <iostream>
2 #include <string>
3
4 using namespace std;
5
6 bool promptGet(istream &in, string &str);
7 void process(string const &str);
```

main.cc

```
1 #include "header.ih"
2
3 int main()
4 {
5     string str;
6     while (promptGet(cin, str))
7         process(str);
8 }
```

process.cc

```
1 #include "header.ih"
2
3 void process(string const &str)
4 {
5     cout << "processed: " << str << '\n';
6 }
```

promptget.cc

```
1 #include "header.ih"
2
3 bool promptGet(istream &in, string &str)
4 {
5     cout << "Enter a line or ^D\n";     // ^D signals end-of-input
6
7     return getline(in, str);
8 }
```

1.

This code doesn't work, because `getline(in, str)` cannot be returned as a bool in `promptGet`. This is because the class `istream` defines `explicit operator bool() const`. This allows the compiler to only perform a conversion to a bool when this is explicitly required (as in a while statement), but not implicitly (as in the return statement above).

2.

By changing `promptGet`'s body in the following way, the code does compile:

promptget.cc

```
1 #include "header.ih"
2
3 bool promptGet(istream &in, string &str)
4 {
5     cout << "Enter a line or ^D\n";     // ^D signals end-of-input
6
7     return static_cast<bool>(getline(in, str));
8 }
```

3.

By changing `promptGet` (and the declaration in the internal header) in the following way, the code does compile:

promptget.cc

```
1 #include "header.ih"
2
3 istream &promptGet(istream &in, string &str)
4 {
5     cout << "Enter a line or ^D\n";     // ^D signals end-of-input
6
7     return getline(in, str);
8 }
```

Exercise 3

Learn to implement index operators

The Matrix class that is used here, is derived from the solutions of excercise 64. We used the following code,

matrix/matrix.h

```
#ifndef INCLUDED_MATRIX_
#define INCLUDED_MATRIX_

#include <iosfwd>
#include <initializer_list>

class Matrix
{
    size_t d_nRows = 0;
    size_t d_nCols = 0;
    double *d_data = 0;                     // in fact R x C matrix

        // exercise 5
        // ==========
    size_t d_idxColStart = 0;
    size_t d_idxRowStart = 0;
    size_t d_nColEnd = d_nCols;
    size_t d_nRowEnd = d_nRows;

    std::istream &(Matrix::*d_extractMode)(
        std::istream &in, Matrix const &matrix) const = &Matrix::extractRows;

    public:
        typedef std::initializer_list<std::initializer_list<double>> IniList;

        Matrix() = default;
        Matrix(size_t nRows, size_t nCols);         // 1
        Matrix(Matrix const &other);                // 2
        Matrix(Matrix &&tmp);                       // 3
        explicit Matrix(IniList inilist);           // 4

        ~Matrix();
```

```
33
34          Matrix &operator=(Matrix const &rhs);
35          Matrix &operator=(Matrix &&tmp);
36
37          size_t nRows() const;
38          size_t nCols() const;
39          size_t size() const;            // nRows * nCols
40
41          static Matrix identity(size_t dim);
42
43          Matrix &tr();                   // transpose (must be square)
44          Matrix transpose() const;       // any dim.
45
46          void swap(Matrix &other);
47
48              // exercise 3
49              // ==========
50          double *operator[](size_t index);
51          double const *operator[](size_t index) const;
52
53              // exercise 4
54              // ==========
55          friend Matrix operator+(Matrix const &lhs, Matrix const &rhs);
56          friend Matrix operator+(Matrix &&lhs, Matrix const &rhs);
57          Matrix &operator+=(Matrix const &other) &;      // 1
58          Matrix operator+=(Matrix const &other) &&;      // 2
59
60              // exercise 5
61              // ==========
62          friend std::ostream &operator<<(
63              std::ostream &out, Matrix const &matrix);
64          friend std::istream &operator>>(
65              std::istream &in, Matrix const &matrix);
66
67          enum Mode
68          {
69              BY_ROWS,
70              BY_COLS
71          };
72
73          Matrix &operator()(
```

```
74              size_t nRows, size_t nCols, Mode byCols = BY_ROWS);     // 1
75          Matrix &operator()(
76              Mode byCols, size_t idxStart = 0, size_t remLines = 0); // 2
77          Matrix &operator()(
78              Mode byCols, size_t idxRowStart, size_t nSubRows,
79              size_t idxColStart, size_t nSubCols);                   // 3
80
81      private:
82          double &el(size_t row, size_t col) const;
83          void transpose(double *dest) const;
84
85              // exercise 3
86              // ==========               // private backdoor
87          double *operatorIndex(size_t index) const;
88
89              // exercise 4
90              // ==========
91          void add(Matrix const &rhs);
92
93              // exercise 5
94              // ==========
95          std::istream &extractRows(
96              std::istream &in, Matrix const &matrix) const;
97          std::istream &extractCols(
98              std::istream &in, Matrix const &matrix) const;
99  };
100
101 inline size_t Matrix::nCols() const
102 {
103     return d_nCols;
104 }
105
106 inline size_t Matrix::nRows() const
107 {
108     return d_nRows;
109 }
110
111 inline size_t Matrix::size() const
112 {
113     return d_nRows * d_nCols;
114 }
```

```
115
116 inline double &Matrix::el(size_t row, size_t col) const
117 {
118     return d_data[row * d_nCols + col];
119 }
120
121     // exercise 3
122     // ==========
123 inline double *Matrix::operatorIndex(size_t index) const
124 {
125     return d_data + index * d_nCols;
126 }
127
128 inline double *Matrix::operator[](size_t index)
129 {
130     return operatorIndex(index);
131 }
132
133 inline double const *Matrix::operator[](size_t index) const
134 {
135     return operatorIndex(index);
136 }
137
138
139 #endif
```

Exercise 4

Learn to implement and spot opportunities for overloaded operators

The header is shown in exercise 3, the implementations of the added functions are shown below:

matrix/add.cc

```
1  #include "matrix.ih"
2
3  void Matrix::add(Matrix const &rhs)
4  {
5      if (rhs.d_nCols != d_nCols or rhs.d_nRows != d_nRows)
6      {
7          cerr << "Warning: Matrices have differnt size, "
8                  "so cannot be added!\n";
9          exit(1);
10     }
11
12     for (size_t idx = size(); idx--; )
13         d_data[idx] += rhs.d_data[idx];
14 }
```

matrix/operatoradd.cc

```
1  #include "matrix.ih"
2
3  Matrix operator+(Matrix const &lhs, Matrix const &rhs)
4  {
5      Matrix tmp{ lhs };
6      tmp.add(rhs);
7      return tmp;
8  }
```

matrix/operatoradd2.cc

```
1  #include "matrix.ih"
2
```

```
3  Matrix operator+(Matrix &&lhs, Matrix const &rhs)
4  {
5      lhs.add(rhs);
6      return move(lhs);
7  }
```

matrix/operatorcompadd1.cc

```
1  #include "matrix.ih"
2
3  Matrix &Matrix::operator+=(Matrix const &other) &
4  {
5      Matrix tmp{ *this };
6      tmp.add(other);
7      swap(tmp);
8      return *this;
9  }
```

matrix/operatorcompadd2.cc

```
1  #include "matrix.ih"
2
3  Matrix Matrix::operator+=(Matrix const &other) &&
4  {
5      add(other);
6      return move(*this);
7  }
```

Exercise 5

Learn to insert/extract objects of your own class

We used the following code,

matrix/extractcols.cc

```
1  #include "matrix.ih"
2
3  std::istream &Matrix::extractCols(
4      std::istream &in, Matrix const &matrix) const
5  {
6      for (size_t colIdx = matrix.d_idxColStart;
7           colIdx != matrix.d_nColEnd;
8           ++colIdx)
9          for (size_t rowIdx = matrix.d_idxRowStart;
10              rowIdx != matrix.d_nRowEnd;
11              ++rowIdx)
12             in >> matrix.el(rowIdx, colIdx);
13     return in;
14 }
```

matrix/extractrows.cc

```
1  #include "matrix.ih"
2
3  std::istream &Matrix::extractRows(
4      std::istream &in, Matrix const &matrix) const
5  {
6      for (size_t rowIdx = matrix.d_idxRowStart;
7           rowIdx != matrix.d_nRowEnd;
8           ++rowIdx)
9          for (size_t colIdx = matrix.d_idxColStart;
10              colIdx != matrix.d_nColEnd;
11              ++colIdx)
12             in >> matrix.el(rowIdx, colIdx);
13     return in;
14 }
```

matrix/functor1.cc

```
1  #include "matrix.ih"
2
3  Matrix &Matrix::operator()(size_t nRows, size_t nCols, Mode byCols)
4  {
5      Matrix tmp{ nRows, nCols };
6      swap(tmp);
7      if (byCols)
8          d_extractMode = &Matrix::extractCols;
9      return *this;
10 }
```

matrix/functor2.cc

```
1  #include "matrix.ih"
2
3  Matrix &Matrix::operator()(Mode byCols, size_t idxStart, size_t remLines)
4  {
5      if (byCols)
6      {
7          d_extractMode = &Matrix::extractCols;
8
9          if (idxStart >= d_nCols)
10         {       // if requested submatrix lies outside matrix, do nothing
11             d_idxColStart = d_nColEnd;
12             return *this;
13         }
14         d_idxColStart = idxStart;
15                 // if a valid third argument is given
16         if (remLines and d_idxColStart + remLines < d_nCols)
17             d_nColEnd = d_idxColStart + remLines;
18     }
19     else        // extract by rows
20     {
21         if (idxStart >= d_nRows)
22         {       // if requested submatrix lies outside matrix, do nothing
23             d_idxColStart = d_nRowEnd;
24             return *this;
25         }
```

```
26          d_idxRowStart = idxStart;
27                  // if a valid third argument is given
28          if (remLines and d_idxRowStart + remLines < d_nRows)
29              d_nRowEnd = d_idxRowStart + remLines;
30      }
31
32      return *this;
33  }
```

matrix/functor3.cc

```
1   #include "matrix.ih"
2
3   Matrix &Matrix::operator()(Mode byCols,
4       size_t idxRowStart, size_t nSubRows, size_t idxColStart, size_t nSubCols)
5   {
6       if (idxRowStart >= d_nRows or idxColStart >= d_nCols)
7       {                   // if submatrix lies outside matrix then do nothing
8           d_idxRowStart = d_nRowEnd;
9           d_idxColStart = d_nColEnd;
10          return *this;
11      }
12
13      d_idxRowStart = idxRowStart; // set start values submatrix
14      d_idxColStart = idxColStart;
15
16      if (byCols)
17          d_extractMode = &Matrix::extractCols;
18
19      // set end values submatrix if within matrix
20      if (d_idxRowStart + nSubRows < d_nRows)
21          d_nRowEnd = d_idxRowStart + nSubRows;
22
23      if (d_idxColStart + nSubCols < d_nCols)
24          d_nColEnd = d_idxColStart + nSubCols;
25
26      return *this;
27  }
```

matrix/operatorextract.cc

```
#include "matrix.ih"

istream &operator>>(istream &in, Matrix const &matrix)
{
    return static_cast<istream &>(
        (matrix.*matrix.d_extractMode)(in, matrix));
}
```

matrix/operatorinsert.cc

```
#include "matrix.ih"

ostream &operator<<(ostream &out, Matrix const &matrix)
{
    for (size_t rowIdx = 0; rowIdx != matrix.d_nRows; ++rowIdx)
    {
        for (size_t colIdx = 0; colIdx != matrix.d_nCols; ++colIdx)
            out << matrix.el(rowIdx, colIdx) << " ";
        out << '\n'; // add newline after each row
    }
    return out;
}
```

Exercise 6

Exercise 7

Learn to implement and spot opportunities for overloaded operators

1.

The following two overloaded operators are added to compare two `Matrix` objects for (in)equality:

matrix/operatorequalto.cc

```
1  #include "matrix.ih"
2
3  bool operator==(Matrix const &lhs, Matrix const &rhs)
4  {
5      if (lhs.d_nCols != rhs.d_nCols or lhs.d_nRows != rhs.d_nRows)
6          return false;
7
8      for (size_t idx = lhs.size(); --idx; )
9      {
10         if (lhs.d_data[idx] != rhs.d_data[idx])
11             return false;
12     }
13     return true;
14 }
```

matrix/operatornotequalto.cc

```
1  #include "matrix.ih"
2
3  bool operator!=(Matrix const &lhs, Matrix const &rhs)
4  {
5      if (!(lhs == rhs))
6          return true;
7
8      return false;
9  }
```

Exercise 8

Exercise 9

Exercise 10