Exercises week 1

Klaas Isaac Bijlsma
s2394480

David Vroom
s2309939

November 19, 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

```
1  #ifndef INCLUDED_MATRIX_
2  #define INCLUDED_MATRIX_
3
4  #include <iosfwd>
5  #include <initializer_list>
6
7  class Matrix
8  {
9      size_t d_nRows = 0;
10     size_t d_nCols = 0;
11     double *d_data = 0;                      // in fact R x C matrix
12
13         // exercise 5
14         // ==========
15     std::istream &(Matrix::*d_extractMode)(
16             std::istream &in, Matrix const &matrix) const = &Matrix::extract
17
18     public:
19         typedef std::initializer_list<
20             std::initializer_list<double>> IniList;
21
22         Matrix() = default;
23         Matrix(size_t nRows, size_t nCols);          // 1
24         Matrix(Matrix const &other);                 // 2
25         Matrix(Matrix &&tmp);                        // 3
26         Matrix(IniList inilist);                     // 4
27
28         ~Matrix();
29
30         Matrix &operator=(Matrix const &rhs);
31         Matrix &operator=(Matrix &&tmp);
32
```

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

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

```
115     return d_data + index * d_nCols;
116 }
117
118 inline double *Matrix::operator[](size_t index)
119 {
120     return operatorIndex(index);
121 }
122
123 inline double *Matrix::operator[](size_t index) const
124 {
125     return operatorIndex(index);
126 }
127
128     // exercise 4
129     // ==========
130 Matrix operator+(Matrix const &lhs, Matrix const &rhs);     // 1
131 Matrix operator+(Matrix &&lhs, Matrix const &rhs);          // 2
132
133     // exercise 5
134     // ==========
135 std::ostream &operator<<(std::ostream &out, Matrix const &matrix);
136 std::istream &operator>>(std::istream &in, Matrix const &matrix);
137
138 #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

Exercise 6

Exercise 7

Exercise 8

Exercise 9

Exercise 10