Week 2

Exercise 10

../10–6/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  string test[2][2] = {
    {"hello", "bye"},
    {"nono", "yes"}
  };

  Matrix<2, 2, string> b = move(test);

  cout << b;

}
```

../10–6/matrix/matrix.h

```
#ifndef INCLUDED_MATRIX_
#define INCLUDED_MATRIX_

#include <algorithm>
#include <iostream>


template<size_t Rows, size_t Columns, typename DataType>
class Matrix
{
      typedef Matrix<1, Columns, DataType>    MatrixRow;
      MatrixRow d_matrix[Rows];

  public:
    Matrix();
    Matrix(DataType matrix[Rows][Columns]);

    MatrixRow &operator[](size_t idx);
    MatrixRow const &operator[](size_t idx) const;

    void print();

    Matrix &operator=(Matrix &&) = default;

  private:

    template<size_t Rows2, size_t Columns2, typename LhsType, typename RhsType>
    friend bool operator==(Matrix<Rows2, Columns2,  LhsType> const &lhs,
                           Matrix<Rows2, Columns2,  RhsType> const &rhs);

    template<size_t Rows2, size_t Columns2, typename LhsType, typename RhsType>
    friend bool operator!=(Matrix<Rows2, Columns2,  LhsType> const &lhs,
                           Matrix<Rows2, Columns2,  RhsType> const &rhs);

    template<size_t Rows2, size_t Columns2, typename LhsType, typename RhsType>
    friend Matrix<Rows2, Columns2,  LhsType> operator+(
                          Matrix<Rows2, Columns2,  LhsType> const &lhs,
                          Matrix<Rows2, Columns2,  RhsType> const &rhs);

    template<size_t Rows2, size_t Columns2, typename LhsType, typename RhsType>
    friend Matrix<Rows2, Columns2,  LhsType> operator+=(
                          Matrix<Rows2, Columns2,  LhsType>  &lhs,
```

```
43                          Matrix<Rows2, Columns2,  RhsType>  &rhs);
44
45      template<size_t Rows2, size_t Columns2, typename LhsType, typename RhsType>
46      friend Matrix<Rows2, Columns2, LhsType> operator+(
47                          Matrix<Rows2, Columns2, LhsType> const &lhs,
48                          RhsType const &rhs);
49
50      template<size_t Rows2, size_t Columns2, typename LhsType, typename RhsType>
51      friend Matrix<Rows2, Columns2, LhsType> operator+(
52                          LhsType const &lhs,
53                          Matrix<Rows2, Columns2,  RhsType> const &rhs);
54
55      template<size_t Rows2, size_t Columns2, typename RhsType>
56      friend std::ostream &operator<<(
57                          std::ostream &out,
58                          Matrix<Rows2, Columns2,  RhsType> const &rhs);
59
60      template<size_t Rows2, size_t Columns2, typename RhsType>
61      friend std::istream &operator>>(
62                          std::istream &in,
63                          Matrix<Rows2, Columns2, RhsType> &rhs);
64
65  };
66
67  //MATRIXROW
68  template <size_t Columns, typename DataType>  // no default allowed
69  class Matrix<1, Columns, DataType>
70  //=
71  {
72  //ROWDATA
73      DataType d_column[Columns];
74  //=
75      public:
76          Matrix();
77
78          template <size_t Rows>
79          Matrix(Matrix<Rows, Columns, DataType> const &matrix);
80
81          DataType &operator[](size_t idx);
82          DataType const &operator[](size_t idx) const;
83  };
84
85  //OPERATORS
86  template<size_t Rows, size_t Columns, typename LhsType, typename RhsType>
87  bool operator==(Matrix<Rows, Columns,  LhsType> const &lhs,
88                  Matrix<Rows, Columns,  RhsType> const &rhs)
89  {
90    for (size_t row = 0; row < Rows; ++row)
91      for (size_t col = 0; col < Columns; ++col)
92        if(lhs.d_matrix[row][col] != rhs.d_matrix[row][col])
93          return false;
94    return true;
95  }
96
97  template<size_t Rows, size_t Columns, typename LhsType, typename RhsType>
98  bool operator!=(Matrix<Rows, Columns,  LhsType> const &lhs,
99                  Matrix<Rows, Columns,  RhsType> const &rhs)
100 {
101   for (size_t row = 0; row < Rows; ++row)
102     for (size_t col = 0; col < Columns; ++col)
103       if(lhs.d_matrix[row][col] != rhs.d_matrix[row][col])
104         return true;
105   return false;
106 }
107
108 template<size_t Rows, size_t Columns, typename LhsType, typename RhsType>
```

```
109 Matrix<Rows, Columns,  LhsType> operator+(
110   Matrix<Rows, Columns,  LhsType> const &lhs,
111   Matrix<Rows, Columns,  RhsType> const &rhs)
112 {
113   Matrix<Rows, Columns, LhsType> tmp;
114   for (size_t row = 0; row < Rows; ++row)
115     for (size_t col = 0; col < Columns; ++col)
116       tmp.d_matrix[row][col] = lhs.d_matrix[row][col] + rhs.d_matrix[row][col];
117   return tmp;
118 }
119
120 template<size_t Rows, size_t Columns, typename LhsType, typename RhsType>
121 Matrix<Rows, Columns,  LhsType> operator+=(
122   Matrix<Rows, Columns,  LhsType> &lhs,
123   Matrix<Rows, Columns,  RhsType> &rhs)
124 {
125   for (size_t row = 0; row < Rows; ++row)
126     for (size_t col = 0; col < Columns; ++col)
127       lhs.d_matrix[row][col] += rhs.d_matrix[row][col];
128   return lhs;
129 }
130
131 template<size_t Rows, size_t Columns, typename LhsType, typename RhsType>
132 Matrix<Rows, Columns, LhsType> operator+(
133                 Matrix<Rows, Columns,  LhsType> const &lhs,
134                 RhsType const &rhs)
135 {
136   Matrix<Rows, Columns, LhsType> tmp;
137   for (size_t row = 0; row < Rows; ++row)
138     for (size_t col = 0; col < Columns; ++col)
139       tmp.d_matrix[row][col] = lhs.d_matrix[row][col] + rhs;
140   return tmp;
141 }
142
143 template<size_t Rows, size_t Columns, typename LhsType, typename RhsType>
144 Matrix<Rows, Columns, LhsType> operator+(
145                 LhsType const &lhs,
146                 Matrix<Rows, Columns,  RhsType> const &rhs)
147 {
148   Matrix<Rows, Columns, LhsType> tmp;
149   for (size_t row = 0; row < Rows; ++row)
150     for (size_t col = 0; col < Columns; ++col)
151       tmp.d_matrix[row][col] = lhs + rhs.d_matrix[row][col];
152   return tmp;
153 }
154
155 template<size_t Rows, size_t Columns, typename RhsType>
156 std::ostream &operator<<(
157                 std::ostream &out,
158                 Matrix<Rows, Columns,  RhsType> const &rhs)
159 {
160   for (size_t row = 0; row < Rows; ++row)
161   {
162     for (size_t col = 0; col < Columns; ++col)
163       out << rhs.d_matrix[row][col] << '\t';
164     out << '\n';
165   }
166   return out;
167 }
168
169 template<size_t Rows, size_t Columns, typename RhsType>
170 std::istream &operator>>(
171                 std::istream &in,
172                 Matrix<Rows, Columns, RhsType> &rhs)
173 {
174   for (size_t row = 0; row < Rows; ++row)
```

```
175   {
176     for (size_t col = 0; col < Columns; ++col)
177       in >> rhs.d_matrix[row][col];
178   }
179   return in;
180 }
181
182 //CONSTRUCTORS
183 template <size_t Rows, size_t Columns, typename DataType>
184 Matrix<Rows, Columns, DataType>::Matrix()
185 {
186     std::fill(d_matrix, d_matrix + Rows, MatrixRow());
187 }
188
189 template <size_t Rows, size_t Columns, typename DataType>
190 Matrix<Rows, Columns, DataType>::Matrix(DataType matrix[Rows][Columns])
191 {
192   for (size_t row = 0; row < Rows; ++row)
193     for (size_t col = 0; col < Columns; ++col)
194       d_matrix[row][col] = matrix[row][col];
195 }
196
197
198 //ROWCONS1
199 template <size_t Columns, typename DataType>
200 Matrix<1, Columns, DataType>::Matrix()
201 {
202     std::fill(d_column, d_column + Columns, DataType());
203 }
204 //=
205 //ROWCONS2
206 template <size_t Columns, typename DataType>
207 template <size_t Rows>
208 Matrix<1, Columns, DataType>::Matrix(
209                         Matrix<Rows, Columns, DataType> const &matrix)
210 {
211     std::fill(d_column, d_column + Columns, DataType());
212
213     for (size_t col = 0; col < Columns; col++)
214         for (size_t row = 0; row < Rows; row++)
215             d_column[col] += matrix[row][col];
216 }
217
218 //ROWOPERATORINDEX
219 template <size_t Columns, typename DataType>
220 DataType &Matrix<1, Columns, DataType>::operator[](size_t idx)
221 {
222     return d_column[idx];
223 }
224
225 template <size_t Columns, typename DataType>
226 DataType const &Matrix<1, Columns, DataType>::operator[](size_t idx) const
227 {
228     return d_column[idx];
229 }
230
231 //OPERATORINDEX
232 template <size_t Rows, size_t Columns, typename DataType>
233 Matrix<1, Columns, DataType>
234 &Matrix<Rows, Columns, DataType>::operator[](size_t idx)
235 {
236     return d_matrix[idx];
237 }
238
239
240 template <size_t Rows, size_t Columns, typename DataType>
```

```
241 Matrix<1, Columns, DataType>
242 const &Matrix<Rows, Columns, DataType>::operator[](size_t idx) const
243 {
244     return d_matrix[idx];
245 }
246
247 //printing
248 template <size_t Rows, size_t Columns, typename DataType>
249 void Matrix<Rows, Columns, DataType>::print()
250 {
251   for (size_t row = 0; row < Rows; ++row)
252   {
253     for (size_t col = 0; col < Columns; ++col)
254       std::cout << d_matrix[row][col] << '\t';
255     std::cout << '\n';
256   }
257 }
258
259 #endif
```

Exercise 11

../11/semaphore/semaphore.h

```
1  #ifndef INCLUDED_SEMAPHORE_
2  #define INCLUDED_SEMAPHORE_
3
4  #include <mutex>
5  #include <condition_variable>
6  #include <utility>
7  #include "chrono"
8
9  class Semaphore
10 {
11   size_t d_nAvailable;
12   std::mutex d_Mutex;
13   std::condition_variable d_condition;
14
15
16   public:
17     Semaphore(size_t nAvailable);
18
19     void notify();
20     void notify_all();
21
22     size_t size() const;
23
24     bool check(size_t &nAvailable);
25
26     void wait() //for use by producer
27     {
28       std::unique_lock<std::mutex> lk(d_Mutex);
29
30       while(d_nAvailable == 0)
31         d_condition.wait(lk);
32
33       --d_nAvailable;
34     }
35
36     template<typename Params> //for use by clients
37     bool wait(Params &&params)
38     {
39       std::unique_lock<std::mutex> lk(d_Mutex);
40
41       while(d_nAvailable == 0)
42         d_condition.wait(lk);
43
44       bool result = params();
45
46       if(result == false)
47         return false;
48       if(result == true && d_nAvailable != 0)
49         return true;
50
51       --d_nAvailable;
52     }
53
54   private:
55 };
56
57 #endif
```