Week 6

Exercise 45

../45/main.cc

```
1  #include <algorithm>
2  #include <iostream>
3  #include <iterator>
4  #include <set>
5  #include <string>
6
7  using namespace std;
8
9  int main(int argc, char const **argv)
10 {
11   set<string> iStrings;
12
13   cout << "Please enter delimited words to be sorted, end input with ^D \n";
14
15   copy(
16        istream_iterator<string>(cin),
17        istream_iterator<string>(),
18        inserter(iStrings, iStrings.end())
19       );
20
21   // Or: set<string> iStrings((istream_iterator<string>(cin)),
22   //                          istream_iterator<string>());
23
24   copy(
25        iStrings.begin(),
26        iStrings.end(),
27        ostream_iterator<string>(cout, " ")
28       );
29 }
```

Exercise 46

../46/main.cc

```
1  #include <vector>
2  #include <algorithm>
3  #include <iostream>
4  #include <string>
5
6  using namespace std;
7
8  int main(int argc, char **argv)
9  {
10   size_t nrRvalues = stoi(argv[1]);
11   size_t maxRvalue = stoi(argv[2]);
12   size_t lookupVal = stoi(argv[3]);
13
14   vector<size_t> numbers;
15
16   for (size_t idx = 0; idx < nrRvalues; ++idx)
17     numbers.push_back(random() % (maxRvalue + 1)); //adding a random number
18                                                    //between 0 and max_rvalue
19   for (auto idx: numbers)
20     cout << idx << '\t';
21   cout << '\n';
22
23   auto it = find_if(
24                       numbers.begin(),
25                       numbers.end(),
26                       [lookupVal](const size_t & val)
27                       {
28                         if (val > lookupVal)
29                           return true;
30                         return false;
31                       }
32                      );
33
34   if (it != numbers.end())
35     cout << "The first value exceeding " << lookupVal << " is at index "
36          << distance(numbers.begin(), it) << '\n';
37   else
38     cout << "No random value exceeds " << lookupVal << '\n';
39
40 }
```

Exercise 47

../47/main.cc

```cpp
#include <algorithm>
#include <iostream>
#include <iterator>

using namespace std;

int main(int argc, char **argv)
{
  sort(
       argv, argv + argc,
       [](char *left, char *right) // Sort ascending
       {
         return *left < *right;
       }
      );
  copy(argv, argv + argc, ostream_iterator<string>(cout, " ")); // Print

  cout << '\n'; // New line

  sort(
       argv,
       argv + argc,
       [](char *left, char *right) // Sort descending
       {
         return *left > *right;
       }
      );
  copy(argv, argv + argc, ostream_iterator<string>(cout, " ")); // Print
}
```

Exercise 48

The difference between `std::copy` and `std::for_each` is centered primarily around the fact that `std::copy` leaves the original intact (unless, of course, as in exercise 47 the destination range is the same as the origin). Furthermore, the `std::for_each` algorithm applies a 'transformation' (i.e. function) to the range, while copy does not. Hence, the latter is applicable for applications where some kind of function must be applied to all elements of a container.

../48/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

using namespace std;

#include <vector>
#include <algorithm>
#include <iterator>
#include <iostream>
```

../48/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  vector<size_t> numbers {1, 2, 3, 4, 5, 6, 7};
  // Simple vector of ascending numbers

  copy(numbers.begin(), numbers.end(), ostream_iterator<size_t>(cout, "\n"));
  // Copies the vector to cout, printing them all, new-line seperated

  for_each(
           numbers.begin(),
           numbers.end(),
           [](size_t &n0)
           {
             cout << ++n0 << '\n';
           }
          );

  // The same, but now the numbers are incremented in the vector, as well as
  // printed
  // This could not be accomplished with copy
}
```

Exercise 49

../49E/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

using namespace std;

#include "student/student.h"

#include <vector>
#include <fstream>
#include <iostream>
#include <algorithm>
#include <numeric>

void read(string fileName, vector<Student> &vStudents);
string toLower(string toLowerCase);
void writeNames(vector<Student> const &vStudents);
void writeNrs(vector<Student> vStudents, vector<size_t> vIndices);

void sortByName(vector<Student> &vStudents);
void sortByNr(vector<Student> &vStudents, vector<size_t> &vIndices);
```

../49E/main.cc

```
#include "main.ih"


int main(int argc, char const **argv)
{
  vector<Student> vStudents;
  read(string(argv[1]), vStudents); //reading input from file

  sortByName(vStudents);            //sorting by name

  writeNames(vStudents);            //printing sorted vector with names

  vector<size_t> vIndices(vStudents.size());
  iota(vIndices.begin(), vIndices.end(), 0); //filling indices with values
                                             //0,1,...,n where n is the last
                                             //element

  sortByNr(vStudents, vIndices);            //sorting by student number

  writeNrs(vStudents, vIndices);            //printing student number and
                                             //grade
}
```

../49E/student/student.h

```
#ifndef INCLUDED_STUDENT_
#define INCLUDED_STUDENT_

#include <string>

class Student
{

    std::string d_name;
    std::string d_lastName;
    size_t      d_sNo;
    double      d_grade;

  public:
```

```
15      Student(std::string firstName, std::string lastName, size_t sNo, double grade);
16      size_t sNo() const;
17      std::string lastName();
18      void printName();
19      void printNr();
20
21  };
22
23  #endif
24
25  inline size_t Student::sNo() const
26  {
27    return d_sNo;
28  };
29
30  inline std::string Student::lastName()
31  {
32    return d_lastName;
33  };
```

../49E/student/student.ih

```
1  #include "student.h"
2
3  #include <iostream>
4  #include <iomanip>
5
6  using namespace std;
```

../49E/student/c_studentInfo.cc

```
1  #include "student.ih"
2
3  Student::Student(string firstName, string lastName, size_t sNo, double grade)
4    : d_name(firstName + ' ' + lastName), d_lastName(lastName), d_sNo(sNo), d_grade(
       grade)
5  {
6  }
```

../49E/student/printname.cc

```
1  #include "student.ih"
2
3
4  void Student::printName()
5  {
6    cout << left << setw(25) << d_name << setw(25) << d_lastName << setw(25)
7         << d_sNo << setw(25) << d_grade << '\n';
8  }
```

../49E/student/printnr.cc

```
1  #include "student.ih"
2
3  void Student::printNr()
4  {
5    cout << d_sNo << '\t'
6         << d_grade << '\n';
7  }
```

../49E/read.cc

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

```
3  void read(string fileName, vector<Student> &vStudents)
4  {
5    ifstream textFile(fileName);
6
7    while (true)
8    {
9      string firstName;
10     string lastName;
11     string sNo;
12     string grade;
13
14     getline(textFile, firstName, '\t');
15
16     if (textFile.eof())
17       break;
18
19     getline(textFile, lastName, '\t');
20     getline(textFile, sNo, '\t');
21     getline(textFile, grade);
22
23     vStudents.push_back(Student(firstName, lastName, stoi(sNo), stod(grade)));
24   }
25 }
```

../49E/sortbyname.cc

```
1  #include "main.ih"
2
3  void sortByName(vector<Student> &vStudents)
4  {
5    sort(
6         vStudents.begin(),
7         vStudents.end(),
8         [](Student left, Student right)
9         {
10          return toLower(left.lastName()) < toLower(right.lastName());
11        }
12       );
13 }
```

../49E/sortbynr.cc

```
1  #include "main.ih"
2
3  void sortByNr(vector<Student> &vStudents, vector<size_t> &vIndices)
4  {
5    sort(
6         vIndices.begin(),
7         vIndices.end(),
8         [vStudents](int left, int right)
9         {
10          return vStudents[left].sNo() < vStudents[right].sNo();
11        }
12       );
13 }
```

../49E/strCaseCmp.cc

```
1  #include "main.ih"
2
3  string toLower(string toLowerCase)
4  {
5    transform(toLowerCase.begin(), toLowerCase.end(), toLowerCase.begin(),
6     ::toupper);
7    return toLowerCase;
8  }
```

../49E/writeNames.cc

```
1  #include "main.ih"
2
3  void writeNames(vector<Student> const &vStudents)
4  {
5    for(auto idx: vStudents)
6    {
7      idx.printName();
8    }
9    cout << '\n';
10 }
```

../49E/writeNrs.cc

```
1  #include "main.ih"
2
3  void writeNrs(vector<Student> vStudents, vector<size_t> vIndices)
4  {
5
6    for (size_t idx = 0; idx < vIndices.size(); ++idx)
7      vStudents[vIndices[idx]].printNr();
8
9    cout << '\n';
10 }
```

Exercise 50

../50–2/main.ih

```
#include <iostream>
#include <string>
#include <vector>
#include <iterator>
#include "line/line.h"

using namespace std;
```

../50–2/main.cc

```
#include "main.ih"

void operator>>(istream &istr, vector<string> &dest)
{
  std::copy(std::istream_iterator<Line>(istr), std::istream_iterator<Line>(),
   inserter(dest, dest.begin()));
}

int main(int argc, char **argv)
{
  vector<string> vs;

  cin >> vs;

  for (auto it: vs)
    cout << it << '\n';
}

/*
If we had used to use istream &operator>>(std::istream &istr, std::string &str)
rather than std::istream &operator>>(std::istream & is, Line &line)
(in line.h) then we would have extracted individual words rather than lines.
This is due to how istream_iterator<string> iterating works. It would use the
extraction operator to a string which is coded this way. By instead using
a istream_iterator<Line> we can create our own extraction operator which
returns lines instead of words.
*/
```

../50–2/line/line.h

```
#ifndef INCLUDED_LINE_
#define INCLUDED_LINE_

class Line : public std::string
{
    friend std::istream &operator>>(std::istream &is, Line &line)
    {
      return std::getline(is, line);
    }
};

#endif
```

Exercise 52

../52/main.ih

```
#define ERR(msg) printf("%s : %d", (msg), __LINE__)

#include <vector>
#include <string>
#include <algorithm>
#include <iostream>
#include <fstream>
#include <iterator>

using namespace std;
```

../52/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  ifstream textFile(argv[1]); // Open file1

  vector<string> data(
                      (istream_iterator<string>(textFile)),
                      istream_iterator<string>()
                     );
  // Construct vector data using istream iterator that goes through file1

  textFile.close(); // Close file1
  textFile.clear(); // Clear flags
  textFile.open(argv[2]); // Open file2

  vector<string> data2(
                       (istream_iterator<string>(textFile)),
                       istream_iterator<string>()
                      );
  // Construct vector data using istream iterator that goes through file2

  remove_if(
            data.begin(),
            data.end(),
            [](string findMe)
            {
              return findMe == "extra";
            }
           );
  // Go through vector, remove instances of 'extra'

  data.insert( data.end(), data2.begin(), data2.end() );
  // Insert data2 at the end of data

  data.erase( unique( data.begin(), data.end() ), data.end() );
  // Erase all non unique entries in data

  vector<string>(data).swap(data);
  // Shrink to fit

  for (auto el: data)
    cout << el << '\n';
  // Print entries, line-seperated

  cout << data.size() << '\t' << data.capacity();
  // Checking space
}
```