Exercises week 6 - STL and Generic Algorithms

Klaas Isaac Bijlsma s2394480

David Vroom s2309939

January 12, 2018

Exercise 42

Learn to extract lines using generic algorithms into a container holding ***`string`*** *objects, although* ***`operator>>()`*** *extracts strings*

If we overloaded the extraction operator for a `std::string`, we would extract individual words. This is because as soon as a whitespace character is encountered, the 'word' is stored in a `std::string`. Therefore we made a class `Derived`, which inherits from `std::string`, and overloaded the extraction operator for this class. We used the following code:

main.cc

```
1  #include <iostream>
2  #include <iterator>
3  #include <string>
4  #include <vector>
5  #include <algorithm>
6
7  using namespace std;
8
9  class Derived: public string
10 {};
11
12 istream &operator>>(istream &istr, Derived &str)
13 {
14     return getline(istr, str);
15 }
```

```
16
17 int main()
18 {
19     vector<string> vs;
20
21     copy(istream_iterator<Derived>(cin), istream_iterator<Derived>(),
22          back_inserter(vs));
23 }
```

Exercise 43

Learn to use promotion with generic algorithms and predefined function objects when manipulating basic data types.

By only using features from the STL, we made a program that sorts the command line's arguments twice, once ascending, once descending without storing them in a `vector`. We used the following code:

main.cc

```
1  #include <iostream>
2  #include <algorithm>
3  #include <functional>
4
5  using namespace std;
6
7  int main(int argc, char **argv)
8  {
9      sort(argv + 1, argv + argc, greater<string>());
10     copy(argv + 1, argv + argc, ostream_iterator<string>(cout));
11     cout << '\n';
12
13     sort(argv + 1, argv + argc, less<string>());
14     copy(argv + 1, argv + argc, ostream_iterator<string>(cout));
15     cout << '\n';
16 }
```

Exercise 44

Learn to recognize a situation where lambda functions may be used

The output of our program produced from the given txt file is given below the code.

vstring/vstring.h

```
1  #ifndef EX44_VSTRING_H
2  #define EX44_VSTRING_H
3
4  #include <vector>
5  #include <string>
6  #include <map>
7  #include <istream>
8
9  class Vstring: public std::vector<std::string>
10 {
11     public:
12         typedef std::map<char, size_t> Charmap;
13
14         explicit Vstring(std::istream &in);
15
16         size_t count(Charmap &cmap, bool (*accept)(char, Charmap &));
17
18     private:
19         static size_t countChar(std::string const &str, Charmap &cmap,
20                                 bool (*accept)(char, Charmap &));
21 };
22
23 #endif
```

vstring/vstring.ih

```
1  #include "vstring.h"
2  #include <algorithm>
3  #include <iterator>
4
5  using namespace std;
```

vstring/count.cc

```
#include "vstring.ih"

size_t Vstring::count(Charmap &cmap, bool (*accept)(char, Charmap &))
{
    size_t count = 0;
    for_each(
        begin(), end(),
        [&, accept](string &str)
        {
            count += countChar(str, cmap, accept);
        }
    );
    return count;
}
```

vstring/countchar.cc

```
#include "vstring.ih"

size_t Vstring::countChar(string const &str, Charmap &cmap,
                          bool (*accept)(char, Charmap &))
{
    return count_if(
        str.begin(), str.end(),
        [&, accept](char ch)
        {
            return accept(ch, cmap);
        }
    );
}
```

main.ih

```
#include <iostream>
#include <fstream>
#include "vstring/vstring.h"
#include <algorithm>

```

```
6  using namespace std;
7
8  void display(Vstring::Charmap &cmap);
9  bool vowels(char c, Vstring::Charmap &cmap);
```

display.cc

```
1  #include "main.ih"
2
3  void display(Vstring::Charmap &cmap)
4  {
5      for_each(
6          cmap.begin(), cmap.end(),
7          [](auto const &value)
8          {
9              cout << value.first << ": " << value.second << '\n';
10         }
11     );
12 }
```

vowels.cc

```
1  #include "main.ih"
2
3  bool vowels(char c, Vstring::Charmap &cmap)
4  {
5      if (string("aeiuoAEIUO").find(c) != string::npos)
6      {
7          ++cmap[c];
8          return true;
9      }
10     return false;
11 }
```

main.cc

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

```
3  int main()
4  {
5      ifstream is("text.txt");
6      Vstring vstring(is);
7      Vstring::Charmap vmap;
8
9      cout << "Vowels: " << vstring.count(vmap, vowels) << '\n';
10
11     display(vmap);
12 }
```

Output

```
Vowels: 819
A: 7
E: 2
I: 8
O: 1
U: 3
a: 192
e: 230
i: 143
o: 148
u: 85
```

Exercise 45

Learn to use generic algorithms to remove elements from a vector

We used the following code:

main.cc

```
1  #include <fstream>
2  #include <vector>
3  #include <string>
4  #include <algorithm>
5  #include <iterator>
6  #include <iostream>
7
8  using namespace std;
9
10 int main(int argc, char **argv)
11 {
12                          // construct ifstream with 1st given filename
13     ifstream inputFile(argv[1]);
14     vector<string> data;  // construct empty vector to store the words
15                          // read all words from 1st file into data
16     copy(istream_iterator<string>(inputFile),
17         istream_iterator<string>(), back_inserter(data));
18                          // close ifstream object to prepare for new file
19
20     inputFile.close();
21                          // associate inputFile with 2nd given file
22     inputFile.open(argv[2]);
23     vector<string> data2; // construct 2nd empty vector
24                          // read all words from 2nd file into data2
25     copy(istream_iterator<string>(inputFile),
26         istream_iterator<string>(), back_inserter(data2));
27
28                          // reorder data s.t. "extra"(s) are at the end
29     auto last = remove(data.begin(), data.end(), string("extra"));
30                          // erase those words from data
31     data.erase(last, data.end());
32
33                          // add all words in data2 to data
```

```
34      data.insert(data.end(), data2.begin(), data2.end());
35                              // sort data, needed for GA unique
36      sort(data.begin(), data.end());
37                              // reorder data s.t. duplicate words are at the end
38      last = unique(data.begin(), data.end());
39                              // erase those words
40      data.erase(last, data.end());
41                              // shed excess capacity
42      vector<string>(data).swap(data);
43
44                              // print words
45      for (string &elem: data)
46          cout << elem << '\n';
47  }
```

Exercise 46

Learn to distinguish two frequently used generic algorithms

In general, the generic algorithm `copy` is used to copy a range to a destination, and the `for_each` generic algorithm passes each element from a range to a funtion or function object. So, with `copy` you can move all elements of an existing range with respect to each other. This is not possible with `for_each`. This is shown in the following code:

copy.cc

```
1  #include <vector>
2  #include <iostream>
3
4  using namespace std;
5
6  int main()
7  {
8      vector<int> vi{-2, -1, 0, 1, 2};
9      copy(vi.begin() + 2, vi.end(), vi.begin());
10
11     for (int elem: vi)
12         cout << elem << ' ';
13     cout << '\n';
14 }
```

.

On the other hand, `copy` can not manipualte the individual elements of the range, something which `for_each` can, as illustrated in the following code:

foreach.cc

```
1  #include <vector>
2  #include <algorithm>
3  #include <iostream>
4
5  using namespace std;
6
7  int main()
8  {
9      vector<int> vi{1,3, -2, 5};
```

```
10      for_each(vi.begin(), vi.end(),
11          [](int &val)
12          {
13              val *= 2;
14          }
15      );
16
17      for (int elem: vi)
18          cout << elem << ' ';
19      cout << '\n';
20  }
```

.