

Week 5

Exercise 36

../36/main.ih

```
1  #define ERR(msg) printf("%s : %d", (msg), __LINE__)
2
3  #include <iostream>
4  #include <string>
5  #include <set>
6
7  using namespace std;
```

../36/main.cc

```
1  #include "main.ih"
2
3  int main(int argc, char const **argv)
4  {
5      string inputString;           // Strings extracted from cin
6      multiset<string> sortedStrings; // Multiset orders with repeats
7
8      cout << "Please enter delimited words to be sorted, end input with ^D \n";
9
10     while(cin >> inputString)      // Input
11         sortedStrings.insert(sortedStrings.begin(), inputString);
12
13     cout << "\nSorted input: \n";
14
15     for (const auto &word: sortedStrings) // Output
16         std::cout << word << ' ';
17 }
```

Exercise 37

../37/main.ih

```
1 #define ERR(msg) printf("%s : %d", (msg), __LINE__)
2
3 #include <iostream>
4 #include <string>
5 #include <set>
6
7 using namespace std;
```

../37/main.cc

```
1 #include "main.ih"
2
3 int main(int argc, char const **argv)
4 {
5     string inputString;           // Strings extracted from cin
6     multiset<string> sortedStrings; // Multiset orders with repeats
7
8     cout << "Please enter delimited words to be sorted, end input with ^D \n";
9
10    while(cin >> inputString)      // Input
11        sortedStrings.insert(sortedStrings.begin(), inputString);
12
13    cout << "\nSorted input: \n"
14         << "String \t\tCount \n";
15
16    for (const auto &word: sortedStrings) // Output
17        std::cout << word << "\t\t" << sortedStrings.count(word) << '\n';
18 }
```

Exercise 38

../38/strings/strings.h

```
1  #ifndef INCLUDED_STRINGS_
2  #define INCLUDED_STRINGS_
3
4  #include <string>      // Actually need string here instead of iosfwd
5  #include <vector>      // Strings container
6  #include <stdexcept>   // For throwing out of range exception
7  #include <iostream>    // Just for testing
8
9  class Strings
10 {
11     std::vector<std::string> d_vStrings;  // New container for strings
12
13 public:
14     Strings() = default;                  // No need for another constructor
15     ~Strings() = default;                 // or destructor
16
17     Strings(int argc, char *argv[]);      // Argc/argv constructor
18     Strings(char **environLike);          // Environ constructor
19
20     size_t size() const;
21     size_t capacity() const;
22     std::string const &at(size_t idx) const; // Only const at
23
24     void add(std::string const &next);      // Adding
25     // Not private since it can be used by user as well
26
27     void resize(size_t newSize);
28     void reserve(size_t newCapacity);
29
30     void operator+=(std::string const &next); // Operators
31     std::string const &operator[](size_t idx) const;
32
33     void print() const;                    // Just for testing
34 };
35
36 #endif
37
38 inline void Strings::operator+=(std::string const &next)
39 {
40     add(next);
41 }
42
43 inline std::string const &Strings::at(size_t idx) const
44 {
45     return d_vStrings.at(idx); // Already throws if out of range
46 }
47
48 // The following inline implementations were outside the scope of the assignment
49
50 inline size_t Strings::size() const
51 {
52     return d_vStrings.size();
53 }
54
55 inline size_t Strings::capacity() const
56 {
57     return d_vStrings.capacity();
58 }
59
60 inline void Strings::resize(size_t newSize)
61 {
62     d_vStrings.resize(newSize);
```

```
63 }
64
65 inline void Strings::reserve(size_t newCapacity)
66 {
67     d_vStrings.reserve(newCapacity);
68 }
69
70 inline std::string const &Strings::operator[](size_t idx) const
71 {
72     return at(idx);
73 }
74
75 inline void Strings::print() const           // Testing
76 {
77     for (auto idx = d_vStrings.begin(); idx != d_vStrings.end(); ++idx)
78         std::cout << *idx << '\n';
79 }
```

../38/strings/strings.ih

```
1 #include "strings.h"
2
3 using namespace std;
```

../38/strings/add.cc

```
1 #include "strings.ih"
2
3 void Strings::add(string const &next)
4 {
5     d_vStrings.push_back(next);
6 }
```

../38/strings/c_stringsArgcArgv.cc

```
1 #include "strings.ih"
2
3 Strings::Strings(int argc, char *argv[])
4 {
5     for (size_t idx = 0, end = argc; idx != end; ++idx)
6         add(argv[idx]);
7 }
```

../38/strings/c_stringsEnv.cc

```
1 #include "strings.ih"
2
3 Strings::Strings(char **environLike)
4 {
5     while (*environLike)
6         add(*environLike++);
7 }
```

Exercise 39

../39/strings/strings.h

```
1  #ifndef INCLUDED_STRINGS_
2  #define INCLUDED_STRINGS_
3
4  #include <string>      // Actually need string here instead of iosfwd
5  #include <vector>      // Strings container
6  #include <stdexcept>   // For throwing out of range exception
7  #include <iostream>    // Just for testing
8
9  class Strings
10 {
11     std::vector<std::string*> d_vStrings; // New container for strings
12
13 public:
14     Strings() = default;                // No need for another constructor
15     ~Strings();                          // But a destructor is needed
16
17     Strings(int argc, char *argv[]);    // Argc/argv constructor
18     Strings(char **environLike);        // Environ constructor
19
20     Strings(const Strings &ogStrings);  // Needs a novel copy constructor
21
22     size_t size() const;
23     size_t capacity() const;
24     std::string const &at(size_t idx) const; // Only const at
25
26     void add(std::string const &next);    // Adding
27     // Not private since it can be used by user as well
28
29     void resize(size_t newSize);
30     void reserve(size_t newCapacity);
31
32     void operator+=(std::string const &next); // Operators
33     std::string const &operator[](size_t idx) const;
34
35     void print() const;                    // Just for testing
36 };
37
38 inline void Strings::operator+=(std::string const &next)
39 {
40     add(next);
41 }
42
43 inline std::string const &Strings::operator[](size_t idx) const
44 {
45     return at(idx);
46 }
47
48 // The following inline implementations were outside of the scope of the assignment
49
50 inline size_t Strings::size() const
51 {
52     return d_vStrings.size();
53 }
54
55 inline size_t Strings::capacity() const
56 {
57     return d_vStrings.capacity();
58 }
59
60 inline void Strings::resize(size_t newSize)
61 {
62     d_vStrings.resize(newSize);
```

```
63 }
64
65 inline void Strings::reserve(size_t newCapacity)
66 {
67     d_vStrings.reserve(newCapacity);
68 }
69
70 inline void Strings::print() const           // Testing
71 {
72     for (auto idx = d_vStrings.begin(); idx != d_vStrings.end(); ++idx)
73         std::cout << **idx << '\n';
74 }
75
76 #endif
```

../39/strings/strings.ih

```
1 #include "strings.h"
2
3 using namespace std;
```

../39/strings/add.cc

```
1 #include "strings.ih"
2
3 void Strings::add(string const &next)
4 {
5     string *pString = new string(next);
6     d_vStrings.push_back(pString);
7 }
```

../39/strings/at.cc

```
1 #include "strings.ih"
2
3 string const &Strings::at(size_t idx) const
4 {
5     if ( idx > d_vStrings.size() || idx < 0 )
6         throw std::out_of_range( "idx out of range \n" );
7
8     return *d_vStrings[idx];
9 }
10 // A separate at() function is necessary here because of the indirection
```

../39/strings/c_stringsArgcArgv.cc

```
1 #include "strings.ih"
2
3 Strings::Strings(int argc, char *argv[])
4 {
5     for (size_t idx = 0, end = argc; idx != end; ++idx)
6         add(argv[idx]);
7 }
```

../39/strings/c_stringsCopy.cc

```
1 #include "strings.ih"
2
3 Strings::Strings(const Strings &ogStrings)
4 {
5     for (auto elem: ogStrings.d_vStrings)
6         add(*elem);
7 }
```

../39/strings/c_stringsEnv.cc

```
1  #include "strings.ih"
2
3  Strings::Strings(char **environLike)
4  {
5      while (*environLike)
6          add(*environLike++);
7  }
```

Exercise 40

Note that the files pertaining to the first part of this question (i.e. the conceptual/exploratory part) are under `../40/testing/`, and the latter where these concepts are implemented in a class are under `../40/class/`.

`../40/testing/main.ih`

```
1 #define ERR(msg) printf("%s : %d", (msg), __LINE__)
2
3 #include <iostream>
4 #include <string>
5 #include <vector>
6 #include <set>
7
8 using namespace std;
```

`../40/testing/main.cc`

```
1 #include "main.ih"
2
3 #include <fstream>
4
5 int main(int argc, char const **argv)
6 {
7     set<std::string> setWords;
8     char const *filename = { "example.txt" };
9     ifstream input(filename);
10
11     if ( input.is_open() )
12     {
13         std::string word;
14         while (input >> word)
15             setWords.insert(word);
16     }
17
18     vector words(setWords.begin(), setWords.end());
19
20     cout << "Size: " << words.size() << '\n'
21          << "Capacity: " << words.capacity() << '\n'
22          << "- Now adding one more word \n";
23
24     words.push_back("wsdfjasedfsdf");
25
26     cout << "Size: " << words.size() << '\n'
27          << "Capacity: " << words.capacity() << '\n'
28          << "- Now shedding capacity \n";
29
30     words = vector(words);
31
32     cout << "Size: " << words.size() << '\n'
33          << "Capacity: " << words.capacity() << '\n';
34
35     //for (auto idx = words.begin(); idx != words.end(); ++idx)
36     //    cout << *idx << ' ';
37 }
```

`../40/testing/output.txt`

```
1 Size: 125
2 Capacity: 125
3 - Now adding one more word
4 Size: 126
5 Capacity: 250
6 - Now shedding capacity
7 Size: 126
8 Capacity: 126
```


../40/class/vectorclass/vectorclass.h

```
1  #ifndef INCLUDED_VECTORCLASS_
2  #define INCLUDED_VECTORCLASS_
3
4  #include <vector>
5  #include <set>
6  #include <string>
7
8  class VectorClass
9  {
10     private:
11         std::vector<std::string> d_vWords;
12
13     public:
14         VectorClass() = default;
15         VectorClass(char const *filename);
16         void swap(VectorClass &other);
17         size_t size() const;
18         size_t capacity() const;
19         void add(std::string const &newWord);
20 };
21
22 #endif
23
24 inline size_t VectorClass::size() const
25 {
26     return d_vWords.size();
27 }
28
29 inline size_t VectorClass::capacity() const
30 {
31     return d_vWords.capacity();
32 }
33
34 inline void VectorClass::add(std::string const &newWord)
35 {
36     d_vWords.push_back(newWord);
37 }
```

../40/class/vectorclass/vectorclass.ih

```
1  #include "vectorclass.h"
2
3  #include <fstream>
4
5  using namespace std;
```

../40/class/vectorclass/c_vectorclassFile.cc

```
1  #include "vectorclass.ih"
2
3  VectorClass::VectorClass(char const *filename)
4  {
5      set<string> setWords;
6      ifstream input(filename);
7
8      if ( input.is_open() )
9      {
10         string word;
11         while (input >> word)
12             setWords.insert(word);
13     }
14     d_vWords = vector(setWords.begin(), setWords.end());
15 }
```

../40/class/vectorclass/swap.cc

```
1  #include "vectorclass.ih"
2
3  void VectorClass::swap(VectorClass &other)
4  {
5      d_vWords = vector<string>(other.d_vWords);
6  }
```

../40/class/main.ih

```
1  #define ERR(msg) printf("%s : %d", (msg), __LINE__)
2
3  #include "vectorclass/vectorclass.h"
4  #include <iostream>
5
6  using namespace std;
```

../40/class/main.cc

```
1  #include "main.ih"
2
3  int main(int argc, char const **argv)
4  {
5      char const *filename = { "example.txt" };
6      VectorClass myVectorClass(filename);
7
8      cout << "Size: "          << myVectorClass.size() << '\n'
9           << "Capacity: " << myVectorClass.capacity() << '\n'
10          << "- Now adding one word \n";
11
12      myVectorClass.add( "sjdfsdf" );
13
14      cout << "Size: "          << myVectorClass.size() << '\n'
15           << "Capacity: " << myVectorClass.capacity() << '\n'
16          << "- Now shedding capacity using swap()" << '\n';
17
18      myVectorClass.swap(myVectorClass);
19
20      cout << "Size: "          << myVectorClass.size() << '\n'
21           << "Capacity: " << myVectorClass.capacity() << '\n';
22  }
```

../40/class/output.txt

```
1  Size: 125
2  Capacity: 125
3  - Now adding one word
4  Size: 126
5  Capacity: 250
6  - Now shedding capacity using swap()
7  Size: 126
8  Capacity: 126
```

Here, I used the following approach to shed the excess capacity of the `vector` in the class. Using the `swap` function, the data member `d_vWords` is replaced with an anonymous new `vector` constructed directly using the original `d_vWords`. In this process, the size and capacity of the anonymous (and new `d_vWords`) are immediately set appropriately. `shrink_to_fit` should not be used because, as stated, it is merely a request to the compiler to shed capacity. It is therefore not always executed, even though it seems to constitute an explicit command. Furthermore, in a class environment, it makes more sense to incorporate a full 'clean up' of (all) its data allocation, as it were, and to ensure that these instructions are actually executed.

Exercise 41

../41/main.ih

```
1  #define ERR(msg) printf("%s : %d", (msg), __LINE__)
2
3  #include <unordered_map>
4  #include <string>
5  #include <iostream> // For testing: print()
6  #include <fstream>  // For testing: fill()
7
8  using namespace std;
9
10 // These are for my own benefit:
11 void fill(unordered_multimap<string, string> &container);
12 void print(unordered_multimap<string, string> const &container);
```

../41/main.cc

```
1  #include "main.ih"
2
3  int main(int argc, char **argv)
4  {
5      unordered_multimap<string, string> container;
6
7      fill(container); // These are implemented for my own benefit, did not hand
8      print(container); // them in since their implementation was not required
9
10     size_t nUniqueKeys = 0;
11
12     for (size_t idx = 0; idx != container.bucket_count(); ++idx)
13         if ( container.bucket_size(idx) != 0 )
14             ++nUniqueKeys;
15     // Vector creates buckets to store pairs with unique and equal (hashed)
16     // keys together, but some are empty, so this 'filters' those out
17
18     cout << "There are " << nUniqueKeys << " unique keys in the container\n";
19 }
```