Week 5

Exercise 36

../36/main.ih

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

#include <iostream>
#include <string>
#include <set>

using namespace std;
```

../36/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  string inputString;                // Strings extracted from cin
  multiset<string> sortedStrings;    // Multiset orders with repeats

  cout << "Please enter delimited words to be sorted, end input with ^D \n";

  while(cin >> inputString)          // Input
    sortedStrings.insert(sortedStrings.begin(),inputString);

  cout << "\nSorted input: \n";
                                     // Output
  for (const auto &word: sortedStrings)
        std::cout << word << ' ';
}
```

Exercise 37

../37/main.ih

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

#include <iostream>
#include <string>
#include <set>

using namespace std;
```

../37/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  string inputString;                // Strings extracted from cin
  multiset<string> sortedStrings;    // Multiset orders with repeats

  cout << "Please enter delimited words to be sorted, end input with ^D \n";

  while(cin >> inputString)          // Input
    sortedStrings.insert(sortedStrings.begin(),inputString);

  cout << "\nSorted input: \n"
       << "String \t\tCount \n";
                                     // Output
  for (const auto &word: sortedStrings)
        std::cout << word << "\t\t" << sortedStrings.count(word) << '\n';
}
```

Exercise 38

../38/strings/strings.h

```
#ifndef INCLUDED_STRINGS_
#define INCLUDED_STRINGS_

#include <string>     // Actually need string here instead of iosfwd
#include <vector>     // Strings container
#include <stdexcept>  // For throwing out of range exception
#include <iostream>   // Just for testing

class Strings
{
  std::vector<std::string> d_vStrings;  // New container for strings

  public:
    Strings() = default;               // No need for another constructor
    ~Strings() = default;              // or destructor

    Strings(int argc, char *argv[]);  // Argc/argv constructor
    Strings(char **environLike);      // Environ constructor

    size_t size() const;
    size_t capacity() const;
    std::string const &at(size_t idx) const;  // Only const at

    void add(std::string const &next);        // Adding
    // Not private since it can be used by user as well

    void resize(size_t newSize);
    void reserve(size_t newCapacity);

    void operator+=(std::string const &next);         // Operators
    std::string const &operator[](size_t idx) const;

    void print() const;                          // Just for testing
};

#endif

inline void Strings::operator+=(std::string const &next)
{
  add(next);
}

inline std::string const &Strings::at(size_t idx) const
{
  return d_vStrings.at(idx);  // Already throws if out of range
}

// The following inline implementations were outside the scope of the assignment

inline size_t Strings::size() const
{
  return d_vStrings.size();
}

inline size_t Strings::capacity() const
{
  return d_vStrings.capacity();
}

inline void Strings::resize(size_t newSize)
{
  d_vStrings.resize(newSize);
}

inline void Strings::reserve(size_t newCapacity)
{
  d_vStrings.reserve(newCapacity);
}

inline std::string const &Strings::operator[](size_t idx) const
{
  return at(idx);
}

inline void Strings::print() const          // Testing
{
  for (auto idx = d_vStrings.begin(); idx != d_vStrings.end(); ++idx)
    std::cout << *idx << '\n';
}
```

../38/strings/strings.ih

```
#include "strings.h"

using namespace std;
```

../38/strings/add.cc

```
#include "strings.ih"

void Strings::add(string const &next)
{
  d_vStrings.push_back(next);
}
```

../38/strings/c_stringsArgcArgv.cc

```
#include "strings.ih"

Strings::Strings(int argc, char *argv[])
{
  for (size_t idx = 0, end = argc; idx != end; ++idx)
    add(argv[idx]);
}
```

../38/strings/c_stringsEnv.cc

```
#include "strings.ih"

Strings::Strings(char **environLike)
{
  while (*environLike)
    add(*environLike++);
}
```

Exercise 39

../39/strings/strings.h

```cpp
#ifndef INCLUDED_STRINGS_
#define INCLUDED_STRINGS_

#include <string>     // Actually need string here instead of iosfwd
#include <vector>     // Strings container
#include <stdexcept>  // For throwing out of range exception
#include <iostream>   // Just for testing

class Strings
{
  std::vector<std::string*> d_vStrings;  // New container for strings

  public:
    Strings() = default;              // No need for another constructor
    ~Strings();                       // But a destructor is needed

    Strings(int argc, char *argv[]);  // Argc/argv constructor
    Strings(char **environLike);      // Environ constructor

    Strings(const Strings &ogStrings);  // Needs a novel copy constructor

    size_t size() const;
    size_t capacity() const;
    std::string const &at(size_t idx) const;  // Only const at

    void add(std::string const &next);        // Adding
    // Not private since it can be used by user as well

    void resize(size_t newSize);
    void reserve(size_t newCapacity);

    void operator+=(std::string const &next);         // Operators
    std::string const &operator[](size_t idx) const;

    void print() const;                         // Just for testing
};

inline void Strings::operator+=(std::string const &next)
{
  add(next);
}

inline std::string const &Strings::operator[](size_t idx) const
{
  return at(idx);
}

// The following inline implementations were outside of the scope of the assignment

inline size_t Strings::size() const
{
  return d_vStrings.size();
}

inline size_t Strings::capacity() const
{
  return d_vStrings.capacity();
}

inline void Strings::resize(size_t newSize)
{
  d_vStrings.resize(newSize);
}

inline void Strings::reserve(size_t newCapacity)
{
  d_vStrings.reserve(newCapacity);
}

inline void Strings::print() const          // Testing
{
  for (auto idx = d_vStrings.begin(); idx != d_vStrings.end(); ++idx)
    std::cout << **idx << '\n';
}

#endif
```

../39/strings/strings.ih

```
#include "strings.h"

using namespace std;
```

../39/strings/add.cc

```
#include "strings.ih"

void Strings::add(string const &next)
{
  string *pString = new string(next);
  d_vStrings.push_back(pString);
}
```

../39/strings/at.cc

```
#include "strings.ih"

string const &Strings::at(size_t idx) const
{
  if ( idx > d_vStrings.size() || idx < 0 )
    throw std::out_of_range( "idx out of range \n" );

  return *d_vStrings[idx];
}
// A seperate at() function is necessary here because of the indirection
```

../39/strings/c_stringsArgcArgv.cc

```
#include "strings.ih"

Strings::Strings(int argc, char *argv[])
{
  for (size_t idx = 0, end = argc; idx != end; ++idx)
    add(argv[idx]);
}
```

../39/strings/c_stringsCopy.cc

```
#include "strings.ih"

Strings::Strings(const Strings &ogStrings)
{
  for (auto elem: ogStrings.d_vStrings)
    add(*elem);
}
```

../39/strings/c_stringsEnv.cc

```
#include "strings.ih"

Strings::Strings(char **environLike)
{
  while (*environLike)
    add(*environLike++);
}
```

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

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

#include <iostream>
#include <string>
#include <vector>
#include <set>

using namespace std;
```

../40/testing/main.cc

```
#include "main.ih"

#include <fstream>

int main(int argc, char const **argv)
{
  set<std::string> setWords;
  char const *filename = { "example.txt" };
  ifstream input(filename);

  if ( input.is_open() )
  {
    std::string word;
    while (input >> word)
      setWords.insert(word);
  }

  vector words(setWords.begin(), setWords.end());

  cout << "Size: " << words.size() << '\n'
       << "Capacity: " << words.capacity() << '\n'
       << "- Now adding one more word \n";

  words.push_back("wsdfjasedfsdf");

  cout << "Size: " << words.size() << '\n'
       << "Capacity: " << words.capacity() << '\n'
       << "- Now shedding capacity \n";

  words = vector(words);

  cout << "Size: " << words.size() << '\n'
       << "Capacity: " << words.capacity() << '\n';

  //for (auto idx = words.begin(); idx != words.end(); ++idx)
  //  cout << *idx << ' ';
}
```

../40/testing/output.txt

```
Size: 125
Capacity: 125
- Now adding one more word
Size: 126
Capacity: 250
- Now shedding capacity
Size: 126
Capacity: 126
```

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

```
#ifndef INCLUDED_VECTORCLASS_
#define INCLUDED_VECTORCLASS_

#include <vector>
#include <set>
#include <string>

class VectorClass
{
  private:
    std::vector<std::string> d_vWords;

  public:
    VectorClass() = default;
    VectorClass(char const *filename);
    void swap(VectorClass &other);
    size_t size() const;
    size_t capacity() const;
    void add(std::string const &newWord);
};

#endif

inline size_t VectorClass::size() const
{
  return d_vWords.size();
}

inline size_t VectorClass::capacity() const
{
  return d_vWords.capacity();
}

inline void VectorClass::add(std::string const &newWord)
{
  d_vWords.push_back(newWord);
}
```

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

```
#include "vectorclass.h"

#include <fstream>

using namespace std;
```

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

```
#include "vectorclass.ih"

VectorClass::VectorClass(char const *filename)
{
  set<string> setWords;
  ifstream input(filename);

  if ( input.is_open() )
  {
    string word;
    while (input >> word)
      setWords.insert(word);
  }
  d_vWords = vector(setWords.begin(), setWords.end());
}
```

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

```
#include "vectorclass.ih"

void VectorClass::swap(VectorClass &other)
{
  d_vWords = vector<string>(other.d_vWords);
}
```

../40/class/main.ih

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

#include "vectorclass/vectorclass.h"
#include <iostream>

using namespace std;
```

../40/class/main.cc

```
#include "main.ih"

int main(int argc, char const **argv)
{
  char const *filename = { "example.txt" };
  VectorClass myVectorClass(filename);

  cout << "Size: "     << myVectorClass.size() << '\n'
       << "Capacity: " << myVectorClass.capacity() << '\n'
       << "- Now adding one word \n";

  myVectorClass.add( "sjdfsdf" );

  cout << "Size: "     << myVectorClass.size() << '\n'
       << "Capacity: " << myVectorClass.capacity() << '\n'
       << "- Now shedding capacity using swap()" << '\n';

  myVectorClass.swap(myVectorClass);

  cout << "Size: "     << myVectorClass.size() << '\n'
       << "Capacity: " << myVectorClass.capacity() << '\n';
}
```

../40/class/output.txt

```
Size: 125
Capacity: 125
- Now adding one word
Size: 126
Capacity: 250
- Now shedding capacity using swap()
Size: 126
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

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

#include <unordered_map>
#include <string>
#include <iostream> // For testing: print()
#include <fstream>  // For testing: fill()

using namespace std;

// These are for my own benefit:
void fill(unordered_multimap<string, string> &container);
void print(unordered_multimap<string, string> const &container);
```

../41/main.cc

```cpp
#include "main.ih"

int main(int argc, char **argv)
{
  unordered_multimap<string, string> container;

  fill(container);  // These are implemented for my own benefit, did not hand
  print(container); // them in since their implementation was not required

  size_t nUniqueKeys = 0;

  for (size_t idx = 0; idx != container.bucket_count(); ++idx)
    if ( container.bucket_size(idx) != 0 )
      ++nUniqueKeys;
  // Vector creates buckets to store pairs with unique and equal (hashed)
  // keys together, but some are empty, so this 'filters' those out

  cout << "There are " << nUniqueKeys << " unique keys in the container\n";
}
```