### Week 8

### Exercise 67

../67.4/strings/strings.h

```
1 #ifndef INCLUDED_STRINGS_
   #define INCLUDED_STRINGS_
2
3
4
   #include <iosfwd>
5
6
   class Strings
7
8
        size_t d_size = 0;
        size_t d_capacity = 1;
9
                                        // now a double *
10
       std::string **d_str;
11
12
       public:
13
           Strings();
14
15
            Strings(int argc, char *argv[]);
            Strings(char **environLike);
16
17
18
            Strings(Strings const &other); //copy const
19
            Strings(Strings &&tmp); //move const
20
21
            ~Strings();
22
23
            size_t size() const;
24
            size_t capacity() const;
            std::string const &at(size_t idx) const;
                                                         // for const-objects
25
26
           std::string &at(size_t idx);
                                                          // for non-const objects
27
28
           void add(std::string const &next);
                                                          // add another element
29
30
           void resize(size_t newSize);
31
           void reserve(size_t newCapacity);
32
33
           void printstring();
34
35
            Strings & operator = (Strings const & other); //copy assignment operator
36
            Strings & operator = (Strings && tmp); //move assignment operator
37
38
       private:
39
           std::string &safeAt(size_t idx) const;
                                                          // private backdoor
40
            std::string **storageArea();
                                                          // to store the next str.
           void destroy();
41
42
           std::string **enlarged();
                                                          // to d_capacity
           std::string **rawPointers(size_t nPointers);
43
44
45
           void swap(Strings &other);
46
   };
47
   inline size_t Strings::size() const
                                                 // potentially dangerous practice:
48
49
                                                  // inline accessors
50
       return d_size;
51
   }
52
   inline size_t Strings::capacity() const
                                                 // potentially dangerous practice:
53
                                                  // inline accessors
54
   {
55
       return d_capacity;
56
   }
57
58
   inline std::string const &Strings::at(size_t idx) const
59
   {
60
       return safeAt(idx);
```

```
}
61
62
63
   inline std::string &Strings::at(size_t idx)
64
   {
65
        return safeAt(idx);
66
   }
67
68
69
   #endif
                                        ../67.4/strings/strings.ih
   #include "strings.h"
 1
   #include <string>
 2
  #include <cstring>
 3
 4
   #include <iostream>
 5
 6
 7
   using namespace std;
                                    ../67.4/strings/copyconstructor.cc
1
   #include "strings.ih"
 2
 3
   Strings::Strings(Strings const &other)
 4
 5
      d_size(other.d_size),
 6
      d_capacity(other.d_capacity),
 7
      d_str(new string*[other.d_size])
 8
 9
   {
10
      for (size_t idx = 0; idx < d_size; ++idx)</pre>
11
        d_str[idx] = new string(*other.d_str[idx]);
12
                                     ../67.4/strings/copyoperator.cc
 1
   #include "strings.ih"
 2
 3
   Strings &Strings::operator=(Strings const &rvalue)
 4
 5
      d_size = rvalue.d_size;
      d_capacity = rvalue.d_capacity;
 6
 7
      d_str = new string*[rvalue.d_size];
 8
 9
      for (size_t idx = 0; idx < d_size; ++idx)</pre>
10
        d_str[idx] = new string(*rvalue.d_str[idx]);
11
      }
12
13
14
      return *this;
15
   }
                                       ../67.4/strings/destructor.cc
   #include "strings.ih"
 1
 2
 3
   Strings:: Strings()
 4
      for (string **end = d_str + d_size; end-- != d_str; )
 5
 6
      delete *end;
 7
      delete[] d_str;
 8
   }
```

# ../67.4/strings/moveconstructor.cc

```
#include "strings.ih"
1
2
3
   Strings::Strings(Strings &&tmp)
4
5
     d_size(tmp.d_size),
6
     d_capacity(tmp.d_capacity),
7
     d_str(tmp.d_str)
8
9
   {
10
     tmp.d_str = 0;
11
     tmp.d_size = 0;
12
                                    ../67.4/strings/moveoperator.cc
   #include "strings.ih"
1
2
3
   Strings &Strings::operator=(Strings &&tmp)
4
     swap(tmp);
5
6
     return *this;
7
                                        ../67.4/strings/swap.cc
   #include "strings.ih"
1
2
3
   void Strings::swap(Strings &other)
4
   {
5
     char bytes[sizeof(Strings)];
     memcpy(bytes, this, sizeof(Strings));
6
7
     memcpy(this, &other, sizeof(Strings));
8
     memcpy(&other, bytes, sizeof(Strings));
   }
9
```

#### Exercise 68

../68/strings/strings.h

```
1 #ifndef INCLUDED_STRINGS_
   #define INCLUDED_STRINGS_
2
3
4
   #include <iosfwd>
5
6
   class Strings
7
   {
8
        size_t d_size;
9
        std::string *d_str;
10
        bool d_copy;
11
        size_t d_nIterate;
12
13
        public:
            struct POD
14
            {
15
16
                size_t
                             size;
17
                std::string *str;
            };
18
19
20
            Strings();
21
            Strings(int argc, char *argv[]);
22
            Strings(char *environLike[]);
23
            Strings(std::istream &in);
24
            Strings(Strings &&tmp);
25
            Strings(size_t nIterate, bool copy);
26
27
            ~Strings();
28
29
            void swap(Strings &other);
30
            size_t size() const;
31
32
            std::string const *data() const;
            POD release();
33
34
                                                          // for const-objects
35
            std::string const &at(size_t idx) const;
                                                           // for non-const objects
36
            std::string &at(size_t idx);
37
38
            void add(std::string const &next);
                                                           // add another element
39
40
            void iterate(char **environLike);
41
42
            void printstring();
43
        private:
            void fill(char *ntbs[]);
44
                                                           // fill prepared d_str
45
            std::string &safeAt(size_t idx) const;
46
                                                          // private backdoor
            std::string *enlargebyCopy();
47
48
            std::string *enlargebyMove();
49
            void destroy();
50
51
52
            static size_t count(char *environLike[]);    // # elements in env.like
53
54
   };
55
56
   inline size_t Strings::size() const
                                                  // potentially dangerous practice:
57
58
                                                  // inline accessors
   {
59
        return d_size;
60
   }
61
   inline std::string const *Strings::data() const
```

```
{
63
64
        return d_str;
65
   }
66
   inline std::string const &Strings::at(size_t idx) const
67
68
   {
69
        return safeAt(idx);
70
71
   inline std::string &Strings::at(size_t idx)
72
73
74
        return safeAt(idx);
75
   }
76
77
78
   #endif
                                         ../68/strings/strings.ih
   #include "strings.h"
 2
 3
   #include <istream>
 4
   #include <string>
 5
 6
   #include <iostream>
 7
 8
   using namespace std;
 9
10
   extern char **environ;
                                          ../68/strings/add.cc
   #include "strings.ih"
 1
   #include "iostream"
 2
 3
   void Strings::add(string const &next)
 4
 5
 6
        string *tmp;
        if (d_copy)
 7
 8
          tmp = enlargebyCopy();
                                           // make room for the next string,
 9
        else
10
          tmp = enlargebyMove();
11
12
                                                // tmp is the new string *
13
        tmp[d_size] = next;
14
                                                // store next
15
16
        destroy();
                                                // return old memory
17
18
        d_str = tmp;
                                                // update d_str and d_size
19
20
        ++d_size;
21
   }
22
                                       ../68/strings/destructor.cc
   #include "strings.ih"
 1
 2
   Strings::~Strings()
 3
 4
   {
 5
 6
        destroy();
 7
 8
   }
```

### ../68/strings/enlargeByCopy.cc

```
1
   #include "strings.ih"
2
3
   string *Strings::enlargebyCopy()
4
5
        string *ret = new string[d_size + 1];
                                                 // room for an extra string
6
7
        for (size_t idx = 0; idx != d_size; ++idx) // copy existing strings
8
            ret[idx] = d_str[idx];
9
10
11
       return ret;
12
   }
                                     ../68/strings/enlargeByMove.cc
   #include "strings.ih"
1
2
3
   string *Strings::enlargebyMove()
4
5
        string *ret = new string[d_size + 1];
                                                      // room for an extra string
6
7
        for (size_t idx = 0; idx != d_size; ++idx) // copy existing strings
8
            ret[idx] = move(d_str[idx]);
9
10
11
        return ret;
12
   }
                                        ../68/strings/iterate.cc
   #include "strings.ih"
1
2
3
   void Strings::iterate(char **environLike)
4
5
6
     for (size_t idx = 0; idx < d_nIterate; ++idx)</pre>
7
8
          size_t idx2 = 0;
        //for (size_t idx2 = 0; idx2 < 80; ++idx2)
9
          while (environLike [idx2])
10
11
12
            add(environLike[idx2]);
13
            ++idx2;
14
15
     }
16
   }
                                       ../68/strings/printstring.cc
1
   #include "strings.ih"
2
3
   void Strings::printstring()
4
   {
5
        cout << d_str[0] << '\n';</pre>
        cout << d_str[d_size - 1] << '\n';</pre>
6
7
   }
                                        ../68/strings/strings6.cc
1
   #include "strings.ih"
3
   Strings::Strings(size_t nIterate, bool copy)
```

```
{
4
 5
         d_{copy} = copy;
6
         d_nIterate = nIterate;
 7
 8
         d_size = 0;
9
         d_str = 0;
10
11
    }
                                                     ../68/output.txt
    output:
 2
 3
    time tmp/bin/binary 100 copy
    CLUTTER_IM_MODULE=xim
 4
    _=tmp/bin/binary
5
6
7
    real
               0m1,706s
 8
               0m1,693s
    user
9
    sys
               0 \, \text{m0} , 0 \, 12 \, \text{s}
10
11
   time tmp/bin/binary 100 move
12
   CLUTTER_IM_MODULE=xim
13
    _=tmp/bin/binary
14
15
16
   real
               0 \, \text{m0} , 241 \, \text{s}
17
               0m0,150s
    user
18
               0 \, \text{m0}, 0 \, 9 \, 1 \, \text{s}
   sys
```

### Exercise 69

Below is the output from when this program is executed. As can be seen, the move constructor is never called.

```
1 Constructor called
2 \quad {\tt Constructor\ called}
3 Copy constructor called
4 Assignment operator called
5 Destroyer called
6 Destructor called
  Destroyer called
7
8 Destructor called
9 Destroyer called
10 Destructor called
11 Destroyer called
                                           ../69/main.ih
1
   #include "demo/demo.h"
2
3
   using namespace std;
                                           ../69/main.cc
   #include "main.ih"
1
2
3
   int main(int argc, char **argv)
4
   {
5
     Demo myDemo;
6
     Demo demo2 = myDemo.factory(); // Copy elision and thus no move constructor
7
     Demo demo3(demo2);
                                       // Copy constructor
8
     demo3 = demo2;
                                       // Assigment operator
9
   }
                                         ../69/demo/demo.h
   #ifndef INCLUDED_DEMO_
1
2
   #define INCLUDED_DEMO_
3
4
   #include <iostream>
5
   #include <string>
6
7
   class Demo
8
9
     std::string **d_info = 0;
10
     size_t d_capacity = 0;
11
12
     public:
13
       Demo();
                                                       // Constructor
       Demo(Demo const &toBeCopied);
14
                                                      // Copy constructor
15
       Demo &operator=(Demo const &toBeAssigned);
                                                      // Assigment operator
16
       Demo(Demo &&temporary);
                                                      // Move constructor
17
       ~Demo();
                                                      // Destructor
18
       Demo factory();
                                                      // Factory function
19
20
     private:
21
       void destroy();
22
       void enlarge(size_t newSize);
23
   };
24
25
   #endif
26
27
   inline void Demo::destroy()
```

```
28
29
      std::cout << "Destroyer called \n";</pre>
30
      for (size_t idx = 0; idx != d_capacity; ++idx)
31
        delete d_info[idx];
32
   }
33
34
   inline Demo::Demo()
35
36
      d_info ( 0 )
37
38
      std::cout << "Constructor called \n";</pre>
39
   }
40
   inline Demo::Demo(Demo &&temporary)
41
42
43
      d_info( temporary.d_info ),
44
      d_capacity( temporary.d_capacity )
45
      std::cout << "Move constructor called \n";</pre>
46
47
      temporary.d_capacity = 0;
48
      temporary.d_info = 0;
49
   }
50
   inline Demo::Demo(Demo const &toBeCopied)
51
52
      d_info( new std::string *[toBeCopied.d_capacity] ),
53
54
      d_capacity( toBeCopied.d_capacity )
55
   {
      std::cout << "Copy constructor called \n";</pre>
56
57
      for (size_t idx = 0; idx != d_capacity; ++idx)
58
        d_info[idx] = new std::string(*toBeCopied.d_info[idx]);
59
60
   inline Demo &Demo::operator=(Demo const &toBeAssigned)
61
62
      std::cout << "Assignment operator called \n";</pre>
63
64
      destroy();
65
      delete[] d_info;
66
      d_capacity = toBeAssigned.d_capacity;
67
      d_info = new std::string *[d_capacity];
68
69
      for (size_t idx = 0; idx != d_capacity; ++idx)
70
        d_info[idx] = new std::string(*toBeAssigned.d_info[idx]);
71
72
      return *this;
73
   }
74
75
   inline Demo::~Demo()
76
   {
77
      std::cout << "Destructor called \n";</pre>
78
      destroy();
79
      delete[] d_info;
80
81
   inline void Demo::enlarge(size_t newSize)
82
83
84
      std::string **newDB = new std::string*[newSize];
      for (size_t idx = 0; idx != newSize; ++idx)
85
86
        newDB[idx] = move(d_info[idx]);
87
      delete[] d_info;
      d_info = newDB;
88
89
      d_capacity = newSize;
90
   }
```

# ../69/demo/demo.ih

# Exercise 70

 $../70/70.\mathrm{txt}$ 

```
1
2
  _____
3
                 constructors
                              assignment ops.
4
  define:
5
               default copy move
                               copy move
6
7
  no constructor:
8
  default cons:
9
  copy cons (CC):
              -
- MC
10 move cons (MC):
  other cons:
11
12
13 no assignment:
14 copy assignmnt (C=):
15 move assignmnt (M=):
                    M=
                                  M =
16 other assignmnt:
17 -----
```