Week 8

Exercise 68

../68/strings/strings.h

```
#ifndef INCLUDED_STRINGS_
#define INCLUDED_STRINGS_

#include <iosfwd>

class Strings
{
    size_t d_size;
    std::string *d_str;
    bool d_copy;
    size_t d_nIterate;

    public:
        struct POD
        {
            size_t      size;
            std::string *str;
        };

        Strings();
        Strings(int argc, char *argv[]);
        Strings(char *environLike[]);
        Strings(std::istream &in);
        Strings(Strings &&tmp);
        Strings(size_t nIterate, bool copy);

        ~Strings();

        void swap(Strings &other);

        size_t size() const;
        std::string const *data() const;
        POD release();

        std::string const &at(size_t idx) const;    // for const-objects
        std::string &at(size_t idx);                // for non-const objects

        void add(std::string const &next);          // add another element

        void iterate(char **environLike);

        void printstring();
    private:
        void fill(char *ntbs[]);                    // fill prepared d_str

        std::string &safeAt(size_t idx) const;      // private backdoor
        std::string *enlargebyCopy();
        std::string *enlargebyMove();
        void destroy();



        static size_t count(char *environLike[]);   // # elements in env.like

};

inline size_t Strings::size() const         // potentially dangerous practice:
{                                           // inline accessors
    return d_size;
}
```

```
61
62  inline std::string const *Strings::data() const
63  {
64      return d_str;
65  }
66
67  inline std::string const &Strings::at(size_t idx) const
68  {
69      return safeAt(idx);
70  }
71
72  inline std::string &Strings::at(size_t idx)
73  {
74      return safeAt(idx);
75  }
76
77
78  #endif
```

../68/strings/strings.ih

```
1  #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

```
1  #include "strings.ih"
2  #include "iostream"
3  void Strings::add(string const &next)
4  {
5
6      string *tmp;
7      if (d_copy)
8        tmp = enlargebyCopy();          // make room for the next string,
9      else
10       tmp = enlargebyMove();
11
12                                       // tmp is the new string *
13
14     tmp[d_size] = next;               // 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

```
1  #include "strings.ih"
2
3  Strings::~Strings()
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

```
1  #include "strings.ih"
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

```
1  #include "strings.ih"
2
3  void Strings::iterate(char **environLike)
4  {
5
6    for (size_t idx = 0; idx < d_nIterate; ++idx)
7      {
8        size_t idx2 = 0;
9      //for (size_t idx2 = 0; idx2 < 80; ++idx2)
10       while(environLike[idx2])
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';
6      cout << d_str[d_size - 1] << '\n';
7  }
```

../68/strings/strings6.cc

```
1  #include "strings.ih"
```

```
2
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

```
1  output:
2
3  time tmp/bin/binary 100 copy
4  CLUTTER_IM_MODULE=xim
5  _=tmp/bin/binary
6
7  real    0m1,706s
8  user    0m1,693s
9  sys     0m0,012s
10
11
12 time tmp/bin/binary 100 move
13 CLUTTER_IM_MODULE=xim
14 _=tmp/bin/binary
15
16 real    0m0,241s
17 user    0m0,150s
18 sys     0m0,091s
```

Exercise 69

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

```
Constructor called
Constructor called
Copy constructor called
Assignment operator called
Destroyer called
Destructor called
Destroyer called
Destructor called
Destroyer called
Destructor called
Destroyer called
```

../69/main.ih

```
#include "demo/demo.h"

using namespace std;
```

../69/main.cc

```
#include "main.ih"

int main(int argc, char **argv)
{
  Demo myDemo;
  Demo demo2 = myDemo.factory();  // Copy elision and thus no move constructor
  Demo demo3(demo2);              // Copy constructor
  demo3 = demo2;                  // Assigment operator
}
```

../69/demo/demo.h

```
#ifndef INCLUDED_DEMO_
#define INCLUDED_DEMO_

#include <iostream>
#include <string>

class Demo
{
  std::string **d_info = 0;
  size_t d_capacity = 0;

  public:
    Demo();                                         // Constructor
    Demo(Demo const &toBeCopied);                   // Copy constructor
    Demo &operator=(Demo const &toBeAssigned);      // Assigment operator
    Demo(Demo &&temporary);                         // Move constructor
    ~Demo();                                        // Destructor
    Demo factory();                                 // Factory function

  private:
    void destroy();
    void enlarge(size_t newSize);
};

#endif

inline void Demo::destroy()
```

```
28 {
29   std::cout << "Destroyer called \n";
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";
39 }
40
41 inline Demo::Demo(Demo &&temporary)
42 :
43   d_info( temporary.d_info ),
44   d_capacity( temporary.d_capacity )
45 {
46   std::cout << "Move constructor called \n";
47   temporary.d_capacity = 0;
48   temporary.d_info = 0;
49 }
50
51 inline Demo::Demo(Demo const &toBeCopied)
52 :
53   d_info( new std::string *[toBeCopied.d_capacity] ),
54   d_capacity( toBeCopied.d_capacity )
55 {
56   std::cout << "Copy constructor called \n";
57   for (size_t idx = 0; idx != d_capacity; ++idx)
58     d_info[idx] = new std::string(*toBeCopied.d_info[idx]);
59 }
60
61 inline Demo &Demo::operator=(Demo const &toBeAssigned)
62 {
63   std::cout << "Assignment operator called \n";
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";
78   destroy();
79   delete[] d_info;
80 }
81
82 inline void Demo::enlarge(size_t newSize)
83 {
84   std::string **newDB = new std::string*[newSize];
85   for (size_t idx = 0; idx != newSize; ++idx)
86     newDB[idx] = move(d_info[idx]);
87   delete[] d_info;
88   d_info = newDB;
89   d_capacity = newSize;
90 }
```

../69/demo/demo.ih

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

../69/demo/factory.cc

```
1 #include "demo.ih"
2
3 Demo Demo::factory()
4 {
5   return (Demo());  // Provoking copy elision
6 }
```