Week 8

Exercise 67

../67.4/strings/strings.h

```cpp
#ifndef INCLUDED_STRINGS_
#define INCLUDED_STRINGS_

#include <iosfwd>

class Strings
{
    size_t d_size = 0;
    size_t d_capacity = 1;
    std::string **d_str;            // now a double *

    public:
        Strings();

        Strings(int argc, char *argv[]);
        Strings(char **environLike);

        Strings(Strings const &other); //copy const
        Strings(Strings &&tmp); //move const

        ~Strings();

        size_t size() const;
        size_t capacity() const;
        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 resize(size_t newSize);
        void reserve(size_t newCapacity);

        void printstring();

        Strings &operator=(Strings const &other); //copy assignment operator
        Strings &operator=(Strings &&tmp); //move assignment operator

    private:
        std::string &safeAt(size_t idx) const;      // private backdoor
        std::string **storageArea();                // to store the next str.
        void destroy();
        std::string **enlarged();                   // to d_capacity
        std::string **rawPointers(size_t nPointers);

        void swap(Strings &other);
};

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

inline size_t Strings::capacity() const     // potentially dangerous practice:
{                                           // inline accessors
    return d_capacity;
}

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

inline std::string &Strings::at(size_t idx)
{
    return safeAt(idx);
}


#endif
```

../67.4/strings/strings.ih

```
#include "strings.h"
#include <string>
#include <cstring>

#include <iostream>

using namespace std;
```

../67.4/strings/copyconstructor.cc

```
#include "strings.ih"

Strings::Strings(Strings const &other)
:
  d_size(other.d_size),
  d_capacity(other.d_capacity),
  d_str(new string*[other.d_size])

{
  for (size_t idx = 0; idx < d_size; ++idx)
    d_str[idx] = new string(*other.d_str[idx]);
}
```

../67.4/strings/copyoperator.cc

```
#include "strings.ih"

Strings &Strings::operator=(Strings const &rvalue)
{
  d_size = rvalue.d_size;
  d_capacity = rvalue.d_capacity;
  d_str = new string*[rvalue.d_size];

  for (size_t idx = 0; idx < d_size; ++idx)
  {
    d_str[idx] = new string(*rvalue.d_str[idx]);
  }

  return *this;
}
```

../67.4/strings/destructor.cc

```
#include "strings.ih"

Strings::~Strings()
{
  for (string **end = d_str + d_size; end-- != d_str; )
  delete *end;
  delete[] d_str;
}
```

../67.4/strings/moveconstructor.cc

```
#include "strings.ih"

Strings::Strings(Strings &&tmp)
:
  d_size(tmp.d_size),
  d_capacity(tmp.d_capacity),
  d_str(tmp.d_str)

{
  tmp.d_str = 0;
  tmp.d_size = 0;
}
```

../67.4/strings/moveoperator.cc

```
#include "strings.ih"

Strings &Strings::operator=(Strings &&tmp)
{
  swap(tmp);
  return *this;
}
```

../67.4/strings/swap.cc

```
#include "strings.ih"

void Strings::swap(Strings &other)
{
  char bytes[sizeof(Strings)];
  memcpy(bytes, this, sizeof(Strings));
  memcpy(this, &other, sizeof(Strings));
  memcpy(&other, bytes, sizeof(Strings));
}
```

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;
}

inline std::string const *Strings::data() const
```

```
{
    return d_str;
}

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

inline std::string &Strings::at(size_t idx)
{
    return safeAt(idx);
}


#endif
```

../68/strings/strings.ih

```
#include "strings.h"

#include <istream>
#include <string>

#include <iostream>

using namespace std;

extern char **environ;
```

../68/strings/add.cc

```
#include "strings.ih"
#include "iostream"
void Strings::add(string const &next)
{

    string *tmp;
    if (d_copy)
      tmp = enlargebyCopy();          // make room for the next string,
    else
      tmp = enlargebyMove();

                                        // tmp is the new string *

    tmp[d_size] = next;                 // store next

    destroy();                          // return old memory

    d_str = tmp;                        // update d_str and d_size

    ++d_size;

}
```

../68/strings/destructor.cc

```
#include "strings.ih"

Strings::~Strings()
{

    destroy();

}
```

../68/strings/enlargeByCopy.cc

```
#include "strings.ih"

string *Strings::enlargebyCopy()
{
    string *ret = new string[d_size + 1];        // room for an extra string

    for (size_t idx = 0; idx != d_size; ++idx)  // copy existing strings
        ret[idx] = d_str[idx];


    return ret;
}
```

../68/strings/enlargeByMove.cc

```
#include "strings.ih"

string *Strings::enlargebyMove()
{
    string *ret = new string[d_size + 1];        // room for an extra string

    for (size_t idx = 0; idx != d_size; ++idx)  // copy existing strings
        ret[idx] = move(d_str[idx]);


    return ret;
}
```

../68/strings/iterate.cc

```
#include "strings.ih"

void Strings::iterate(char **environLike)
{

  for (size_t idx = 0; idx < d_nIterate; ++idx)
    {
      size_t idx2 = 0;
    //for (size_t idx2 = 0; idx2 < 80; ++idx2)
      while(environLike[idx2])
      {
        add(environLike[idx2]);
        ++idx2;
      }
  }
}
```

../68/strings/printstring.cc

```
#include "strings.ih"

void Strings::printstring()
{
    cout << d_str[0] << '\n';
    cout << d_str[d_size - 1] << '\n';
}
```

../68/strings/strings6.cc

```
#include "strings.ih"

Strings::Strings(size_t nIterate, bool copy)
```

```
{
    d_copy = copy;
    d_nIterate = nIterate;

    d_size = 0;
    d_str = 0;

}
```

../68/output.txt

```
output:

time tmp/bin/binary 100 copy
CLUTTER_IM_MODULE=xim
_=tmp/bin/binary

real    0m1,706s
user    0m1,693s
sys     0m0,012s


time tmp/bin/binary 100 move
CLUTTER_IM_MODULE=xim
_=tmp/bin/binary

real    0m0,241s
user    0m0,150s
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

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

using namespace std;
```

../69/main.cc

```cpp
#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

```cpp
#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()
```

```
{
  std::cout << "Destroyer called \n";
  for (size_t idx = 0; idx != d_capacity; ++idx)
    delete d_info[idx];
}

inline Demo::Demo()
:
  d_info ( 0 )
{
  std::cout << "Constructor called \n";
}

inline Demo::Demo(Demo &&temporary)
:
  d_info( temporary.d_info ),
  d_capacity( temporary.d_capacity )
{
  std::cout << "Move constructor called \n";
  temporary.d_capacity = 0;
  temporary.d_info = 0;
}

inline Demo::Demo(Demo const &toBeCopied)
:
  d_info( new std::string *[toBeCopied.d_capacity] ),
  d_capacity( toBeCopied.d_capacity )
{
  std::cout << "Copy constructor called \n";
  for (size_t idx = 0; idx != d_capacity; ++idx)
    d_info[idx] = new std::string(*toBeCopied.d_info[idx]);
}

inline Demo &Demo::operator=(Demo const &toBeAssigned)
{
  std::cout << "Assignment operator called \n";
  destroy();
  delete[] d_info;
  d_capacity = toBeAssigned.d_capacity;
  d_info = new std::string *[d_capacity];

  for (size_t idx = 0; idx != d_capacity; ++idx)
    d_info[idx] = new std::string(*toBeAssigned.d_info[idx]);

  return *this;
}

inline Demo::~Demo()
{
  std::cout << "Destructor called \n";
  destroy();
  delete[] d_info;
}

inline void Demo::enlarge(size_t newSize)
{
  std::string **newDB = new std::string*[newSize];
  for (size_t idx = 0; idx != newSize; ++idx)
    newDB[idx] = move(d_info[idx]);
  delete[] d_info;
  d_info = newDB;
  d_capacity = newSize;
}
```

../69/demo/demo.ih

```cpp
#include "demo.h"

using namespace std;
```

../69/demo/factory.cc

```cpp
#include "demo.ih"

Demo Demo::factory()
{
  return (Demo());  // Provoking copy elision
}
```

Exercise 70

../70/70.txt

```

-----------------------------------------------------------
                        constructors        assignment ops.
                     ------------------     ---------------
define:              default copy move        copy    move
-----------------------------------------------------------
no constructor:
default cons:
copy cons (CC):         -
move cons (MC):         -    MC
other cons:

no assignment:
copy assignmnt (C=):
move assignmnt (M=):         M=                  M=
other assignmnt:
-------------------------------------------------------------
```