Week 5

Exercises 45 & 47

DANGEROUS

45, 47: 9

../45–47/main.ih

```
// Main file: internal header

#include "strings/strings.h"

#include <string>
#include <iostream>

extern const char **environ;

using namespace std;
```

../45–47/main.cc

```
// Main file
// This is just an example main file to demonstrate the workings of the Strings class
// The constructors will also work for other NTBSs, but environ and argc/argv
// are convenient examples to use.

#include "main.ih"

int main(int argc, char const **argv)
{
  Strings objectA = Strings(cin);      // Create Strings using cin
  Strings objectB = Strings(environ); // Create Strings using environ
  Strings objectC = Strings(argc, argv);  // Create Strings based on argc, argv

  Strings::stringsSwap(objectA, objectB); // Swap environ and istream Strings

  // objectA.printStrings(); // Print what is now environ Strings
  // objectB.printStrings(); // Print what is now istream Strings
  // objectC.printStrings(); // Print the unchanged objectC
  // These are for testing purposes
}
```

TC

../45–47/strings/strings.h

```
#ifndef INCLUDED_STRINGS_
#define INCLUDED_STRINGS_

#include <cstddef>
#include <string>
// #include <ioforward>

class Strings
{
  size_t d_size = 0;       // Number of elements in d_str
  std::string *d_str = 0; // Stored strings

  public:
    Strings(size_t numStrings, char const **strings); // argc, argv constructor
    Strings(char const **strings);                    // environ constructor
    Strings(std::istream &input);                     // istream constructor
    Strings();                                        // default constructor

    void printStrings() const;                        // Just for testing

    // 46
    size_t size() const;
    // std::string* data(); // Not implemented
```

TC, make it default

```
    // std::string* at(size_t index, bool) const; // Not implemented
    // std::string* at(size_t index); // Not implemented

    // 47
    static void stringsSwap(Strings &objectA, Strings &objectB);

  private:
    void add(char const *novelString);          // Add char array to d_str
};

#endif
```

Ok, but why not a member? (instead of a static member)

../45–47/strings/strings.ih

```
#include "strings.h"
#include <iostream>
//#define CERR std::cerr << __FILE__": "

using namespace std;
```

../45–47/strings/addChar.cc

```
#include "strings.ih"

void Strings::add(char const *novelString)
{
  std::string *temporary = new string[d_size + 1];
  // Create a pointer temporary that points towards a newly allocated
  // piece of memory in which an array of
  // d_size + 1 initialised strings are held

  for (size_t index = 0; index != d_size; ++index)
    temporary[index] = d_str[index];
  // Transfer over the current array of strings to temporary

  temporary[d_size] = novelString;
  // Add the new element to the end of temporary

  delete[] d_str;
  // Delete/deallocate the memory currently pointed at by d_str

  d_str = temporary;
  // Point d_str to the memory pointed at by temporary

  ++d_size;
  // Increment d_size
}
```

../45–47/strings/c_argcargv.cc

TC: you know the size. No need to continuously resize

```
#include "strings.ih"

Strings::Strings(size_t numStrings, char const **strings)
{
  std::cout << "Argc / argv constructor called. \n";

  for (size_t index = 0; index != numStrings; ++index)
    add(strings[index]);
  // For NTBSs 0 to numStrings within strings, pass them to the add function
}
```

next page

../45–47/strings/c_default.cc

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

```

Strings::Strings()
{
  std::cout << "Default constructor called. \n";
};
```

|→ clutter. omit

../45–47/strings/c_environ.cc

```
#include "strings.ih"

Strings::Strings(char const **strings)
{
  std::cout << "environ constructor called. \n";

  for (size_t index = 0; strings[index] != 0; ++index)
    add(strings[index]);
  // For NTBSs 0 to when a null char is encountered, pass them to the add function
};
```

../45–47/strings/c_istream.cc

```
#include "strings.ih"

Strings::Strings(std::istream &input)
{
  std::cout << "istream constructor called. \n"
            << "Enter an empty line (enter/return) to hault input. \n";

  std::string newEntry; // Define string newEntry
  while (getline(input, newEntry))  // Loop while getline works, setting
  {                                 // newEntry to the new line
    if (newEntry.empty()) // If getline creates an empty string
      break;  // Break out of the whie loop (happens when enter/return is pressed)

    add(newEntry.c_str());  // Call the add using the newly entered string.
    // Note that the string is converted to a NTBS to work with the add
    // function. Alternatively another add function could be written.
  }
}
```

why can't lines be empty?

../45–47/strings/stringsSwap.cc

```
#include "strings.ih"

void Strings::stringsSwap(Strings &objectA, Strings &objectB)
{
  Strings temporary = objectA;
  // First, a Strings object temporary is created using an implicit (i.e. non-user
  // defined) / trivial copy constructor. In other words, temporary is constructed
  // based on a constant reference to objectA and temporary is now a copy of objectA
  // (in a new location in memory). Strings temporary(objectA); would do the same.
  objectA = objectB;
  // This a default class assignment. Now, both objectA and objectB point to the same
  // memory, which must be remedied.
  objectB = temporary;
  // This assigns objectB to the same memory as temporary.
  // Since temporary is not destroyed, this solution works fine, but really an
  // overloaded assignment operator and copy constructor should be written.
};
```

TC: access the members