Project 3: String vector class

Due: Tue, Apr 12

In this assignment, you will implement a `String_vector` class: a vector containing strings. Note that this assignment builds on your `String` class implementation. In this project, the only executable will be `test_string_vector` and `test_string`, which is essentially the same as it was in Project 2.

Getting started

Clone the project stub into the `Projects` subdirectory of SVN working directory.

```
cd ~/ID/Projects
svn export https://dev.cs.uakron.edu/svn/cs210sp16/shared/Projects/String_vector/
```

Be sure to `svn add` the newly exported `String_vector` directory and then commit.

Investigate the contents of that directory. There's a lot there. You should have the following files:

- `CMakeLists.txt` — The build system for your homework. Read this file carefully.
- `string_vector.hpp` and `string_vector.cpp` — You will be implementing the `String_vector` class in these files.
- `string.hpp` and `string.cpp` — Empty files to be replaced by your implementation from Homework 2.
- `memory.hpp` and `memory.cpp` — Files containing helper functions for implementing `String_vector`.
- `test_string_vector.cpp` — A small test suite for your `String_vector` class. You must not modify this file.
- `test_string.cpp` — A small test suite for your `String` class. You must not modify this file.
- `test.hpp` and `test.cpp` — Tools provided to support testing. Read these files. You must not modify these file, but you may find them interesting to read.

You can try building the project, but it won't work. The test suite and the additional program are written in terms of functions that *you* must provide.

Start by copying your `String` implementation over `string.hpp` and `string.cpp`.

Class requirements

Implementing `String_vector` requires you to provide a number of member functions. Most of are relatively straightforward and have been discussed in class. Minimally, your class must satisfy the following requirements:

```
// Default construction; the default value shall be the empty string.
String_vector v0;

// Initialization by a list of values. This is provided for you. You do
// not need to implement it.
String_vector v1 {"a", "b", "c"};

// Copy construction. After construction, the declared variable (v2) shall
// be equal to the original (v1).
String_vector v2 = v1;

// Copy assignment. After assignment, the assigned object on the left (v1)
// shall be equal to the value on the right (v2).
v1 = v2;

// A member function to determine if a vector is empty. Returns true
// or false. This member function shall not modify its object.
bool b = v1.empty()

// A member function function that returns the number of elements in
// a vector. Returns a std::size_t value. This member function shall
// not modify its object.
std::size_t n = s1.size();

// A member function function that returns the total capacity allocated
// to the vector. Returns a std::size_t value. This member function shall
// not modify its object.
std::size_t n = s1.capacity();

// Element access. Support reading and writing of elements in the vector
// using the subscript operator. Both operators take a std::size_t argument n,
// where n < size(). You must assert that the index is in bounds.
String const& s = v1[0];
v1[0] = "a";
v1[-1]; // asserts

// A member function that returns a pointer to the underlying array.
// This member function shall not modify its object.
String const* p = v1.data();

// A member function that reserves enough capacity to store n strings.
// If n is less than or equal to the current capacity, the function
// shall have no effect.
String_vector v;  // Capacity is 0
v1.reserve(20);   // Capacity is at least 20
v1.reserve(0);    // Capacity is still at least 20.

// A member function that adjusts the size of a vector to n. If n is
// greater than size(), then n - size() default-constructed objects
// are inserted into the vector. Otherwise, size() - n objects are
// destroyed. After resizing, size() == n.
String_vector v; // Size is 0.
v.resize(10);    // Size is 10, all strings are empty
v.resize(5);     // Size is 5, all strings still empty

// A member function that makes the vector empty. This does not release
// spare capacity.
v.clear();    // Size is 0, capacity may be greater.

// Iterators. The begin() and end() functions return iterators to the vector.
// Vector iterators are simply pointers into the vector's array of objects.
// Non-const iterators can be used to modify the contents of the vector.
String_vector v;        // A mutable string vector
String_vector::iterator iter = v.begin();
String_vector::iterator end = v.end();

// Const iterators. The as above except that const iterators cannot modify
// the underlying object.
String_vector const cv; // A constant string vector
String_vector::const_iterator end = cv.begin();
String_vector::const_iterator end = cv.end();

// Equality comparisons. Two strings compare equal when they have the
// same sequence of characters. Hint: see std::copy.
bool b1 = (s1 == s2);
bool b2 = (s1 != s2);

// Ordering. One string compares less than another when it lexicographically
// precedes it. Hint: see std::lexicographical_compare.
bool b1 = (s1 < s2);
bool b2 = (s1 > s2);
bool b3 = (s1 <= s2);
bool b4 = (s1 >= s2);
```

Naturally, you will also be required to provide a destructor that releases or deletes any memory allocated to the string in a constructor or assignment operator.

Define the `String_vector` class and declare all associated functions in `string_vector.hpp`. Define all of those functions in `string_vector.cpp`.

Building and testing

Build your program in the usual way using CMake and make. You can run the test suite by typing in your build directory:

```
make test
```

This will either pass or fail. If you want to see specifically what failed, you can run the test program directly.

```
./test_string_vector
```

Errors will be printed as they are encountered.

Program Sanitizers

Recent compilers provide runtime analysis via a set of plugins called "sanitizers". GCC 5 provides support for two:

- The address sanitizer inserts runtime checks for invalid memory accesses. These will cause your program to trap (crash) when you access memory in an invalid way.
- The undefined behavior sanitizer inserts a number of runtime checks that causes your program to trap (crash) when you invoke some forms of undefined behavior.

It is *very* useful to have these enabled when testing your program. It is easy to write code that does invalid things, but continues to run. Those are, quite simply, the hardest problems to solve.

To enable sanitizers, make sure that you have GCC-5 installed on your system. Here are some links on how to do that:

- Ubuntu: `sudo apt-get install g++-5`
- Mac OS X: `brew install g++5` (If I'm remember correctly)

To use a different compiler, you can set the `CXX` environment variable before running CMake. For example:

```
mkdir build
export CXX=g++-5
cmake ..
```

Then proceed as normal. If you've already run `cmake`, I would just remove your build directory and start over. You can also change your compiler using `ccmake`.

Run your program in the normal way.

Note that running with sanitizers may cause the test suite to fail instead of trapping gracefully. The test suite for this program is designed to cause failures. If you miss an assertion, the test environment will attempt to detect that, but it will continue running. If the sanitizers are enabled and you miss an assertion, your program will die.

I strongly recommend that you build and debug your project using these features. They will help immensely.

Going the extra mile

Implement move semantics and positional insertion/erasure for vectors.

```
String_vector v { ... };

// Move constructor. v2 gets the internal representation of v1, and v1 is
// restored to its default state. No memory is allocated or destroyed.
String_vector v2 = std::move(v1);

// Move assignment. v2 gets the internal representation of v1, and v1 is
// restored to its default state. No memory is allocated or destroyed.
v2 = std::move(v1);

// Let i be an iterator referring to an object in v0.
String_vector::itrerator i = ..;

// A member function to insert a string before the iterator i. This
// Returns an iterator to the inserted element. Note that insertion
// at begin() is valid.
String_vector::iterator j = v1.insert(i, "abc");

// A member function to erase the element referred to by an iterator. This
// Returns an iterator past the erased element.
String_vector::iterator j = v1.insert(i, "abc");
```

Submission

Homework is submitted in two ways: 1. Committing it to your SVN repository 2. Submitting a printout on the day the project is due.

To generate the printout, simply type:

```
make printout
```

in your build directory. This will generate a code listing in `printout.pdf` with the `string.hpp` and `string.cpp`. Open the PDF (using Chrome, Firefox, or a PDF viewer) and print it (two-sided if possible). This is easily done from a lab computer. **Staple multiple sheets together.**

DO NOT FORGET TO BRING YOUR PRINTOUT TO CLASS.

Grading basis

If your homework is not in subversion OR you did not submit a printout, you will get a 0 on your assignment. You must submit **both** to receive a grade.

The total is out of 100 points.

- 50 You submitted something shows you put minimal effort into doing the work
- 65 You submitted code that does not compile
- 75 You submitted code that compiles, but crashes or has errors when executed
- 90 You submitted a program that compiles and runs with no errors
- 100 You've gone the extra mile (see above), and your source code is well documented and consistently styled