CS 455 Lab 11: C++ Vectors

Fall 2021 [Bono]

Goals

In this lab you're going to get some practice with basic C++ programming by writing a program with a few simple functions that use C++ vectors. The C++ vector class is very similar to Java ArrayList. If the lab looks familiar, it's because this lab has some of the same problems as your lab 4, but unlike lab 4 this lab doesn't involve implementing a class.

Since you are probably new to writing and compiling C++ code, you and your partner should trade off who is the driver for each exercise in this lab.

You are required to do this lab in the Vocareum environment, so you can make sure you know how to use C++ in Vocareum (you don't want to get surprised when you try to turn in your first C++ assignment).

Reading and reference material

- The following sections of cplusplus.com C++ Language Tutorial: Introduction, Basics of C++, and in Program Structure, the sections on Control Structures and Functions.
- From cal-linux.com: C++ Vectors (up to the part about Iterators)
- Code examples from Thu lecture on C++ Basics (available after lecture)

Background

Here are all the vector operations you will need to use for this lab, shown by example:

```
vector<int> v;       // creates an empty vector of ints
int n = v.size();    // the size of the vector (currently 0)
v.push_back(3);      // add an element to the end of the vector
v.push_back(5);      // add another element.  vector is [3, 5]
cout << v[0];      // access element at position 0 (currently 3), and prints it
v[1] = 7;            // update element at position 1.  vector is [3, 7]
v[2] = 10;    // ERROR: invalid index for this vector
```

You can see here that the C++ vector is very similar to the Java *ArrayList*. One syntactic difference is that with a vector you can access or change individual elements using the same [] syntax as you do with a Java *array* (C++ arrays also use this syntax to index elements). Also, C++ templates (the feature equivalent to Java generics) allow you to use a primitive type, such as `int`, as the template parameter.

The main difference between objects in general in C++ and Java is that the first statement above that declares the vector also creates the object and initializes it (i.e., it calls the default constructor for vector). So `v` holds the vector itself, not a reference to it. (You will get a compile error if you try to assign a "`new`" expression to it.) Put another way, C++ objects have value semantics, just like primitive types. (Although, we'll see later that the situation changes if we use pointers to objects.)

Since object variables are not object-references in C++, when we pass one as a parameter, it is passed by value, like primitive types such as `int` and `double`. I.e., the formal parameter in the called function is a copy of the object passed from the caller. (We will discuss other C++ parameter-passing modes in lecture next week.)

Exercise 1 (1 checkoff point)

Write a program called `testVec.cpp` to read an indeterminate number of integers into a vector and then print out the values in the vector. This program is going to become the test-driver for the functions we write in the following exercises. Your program must define and call two functions that do the bulk of the work:

```
vector<int> readVals();
void printVals(vector<int> v);
```

A note about syntax: The above are function headers (aka function prototypes) in C++. Stand-alone functions (i.e., ones not part of a class) in C++ are analogous to static methods in Java. As you can see, their syntax is almost identical to method headers in Java (except they do not appear inside of a class).

To be more specific, `readVals` should read values until it reaches end of file; it should not prompt for each value. Thus, you can run it with input from the keyboard, or using Unix input redirection, just like we have done with similar Java programs. `printVals` should print its values space-separated, finally ending with a newline. Your program should work for zero or more values.

Here's a reminder of the compile command to use:

```
g++ -ggdb -Wall testVec.cpp -o testVec
```

Note the `-o` option, which lets you specify the name of the executable file.

With the `-o` option, above, your executable will be called `testVec`. Note: you may get a compiler warning about comparing signed and unsigned ints; don't worry about that for today.

To get checked-off, show the TA your working `testVec` program, including the source code.

If you get a lot of compile errors, it may be easier to view them if you save the messages to a file instead. You can do that with the following command, and then look at `compile.out` in the Vocareum editor:

```
g++ -ggdb -Wall testVec.cpp -o testVec >& compile.out
```

The ">&" means that both `cout` and `cerr` go to the file. `cerr` is an output stream for error messages, and that's where g++ puts its error messages.

Exercise 2 (1 checkoff point)

Copy your completed `testVec.cpp` (i.e., from Ex. 1) to a file called `filt.cpp`. Add a function which creates a vector all of whose elements appear in the original vector, but that all satisfy some condition, for example, values that are > 0. You can choose what the filter condition will be (and name your function appropriately). Of course, if no elements satisfy the condition, then the filtered version of the vector will be empty. The original vector will be unchanged by this operation. Here's an example interface:

```
// returns a vector of values from v that are greater than 0
// these values are in the same relative order as they are in v.
vector<int> valsGT0(vector<int> v);
```

Here's an example showing what your output should look like. This also shows using input redirection with your program:

```
% filt < somevals
Vector: 3 -50 103 0 -12 103 20
Filtered vector: 3 103 103 20
Original vector: 3 -50 103 0 -12 103 20
```

NOTE: Your progam will use your already-written `readVals` and `printVals` to read and print the vector(s).

Use the following command to compile your program.

```
g++ -ggdb -Wall filt.cpp -o filt
```

Prepare various test files to test your program with. To get checked-off, show the TA your working `filt` program, including the source code and your test files.

Exercise 3 (1 checkoff point)

Copy your completed `filt.cpp` to a file called `findLast.cpp`. Add and call a new function called `findLast` which finds the last instance of a particular value in a vector.

```
/**
 * returns location of last instance of target in v or -1 if not found
 */
int findLast(vector<int> v, int target);
```

For example, if vector `v` had the values `[1, -2, 12, 7, 19, 7, 12]`, `findLast(v, 7)` would return `5`, and `findLast(v, 1)` would return `0`.

To test `findLast` try it on the vector of read-in values as well as the filtered vector your program already creates. But for the *targets* to test it on, use hardcoded data. Create test files that make sense to test this hard-coded data with (named `in1`, `in2`, etc.) To make it easier to write the tests, add a function `testFindLast` which takes the vector and target to test, and does the test and prints the test case and the result. (i.e, each test will involve one call to `testFindLast`). Here's some sample output for such a function:

```
Vector: 1 2 3 4 5 6 7 8 9 10 7 8
The last instance of 7 is at position 10
```

(it's position 10 because we are using an index counting from 0)

To try out another feature of C++, in `testFindLast` write the code so that the call to `findLast` is right in the `cout` statement (i.e., don't use a temp variable).

Compile your code so that the executable has the name `findLast`.

To get checked-off, show the TA your working `findLast` program, including the source code, and your sample input files.

Checkoff for DEN students

When you click the `Submit` button, it will be looking for and compiling (for source code) the files `in1`, `in2`, `testVec.cpp`, `filt.cpp`, and `findLast.cpp`.