Lab #15: Vectors

Maria Bosco

Due Date: December 7th, 2016

It's holidays, and some of the Teaching Assistants and Professors decide to get together for a holiday extravaganza. Seeing as your star student, you decide to have them over. With the assistance of a newly-learned topic (vectors), your job is to write a C++ program to invite the guests, welcome them to your home, exchange gifts, and kick them out.

1 Preliminaries

Fundamentally, a vector is a C++ array. Meaning everything you did with an array to this point, you can do with a vector. After including the vector header (`#include <vector>`) and specifying a type (`vector<type>` , where `type` can be primitives like `char` or `double` or even classes like `string`). We see they're not much different from arrays.

```
1     vector<int> vector = { 1, 2, 3, 4, 5, 6 };
2     int array[6] = { 1, 2, 3, 4, 5, 6 };
3
4     /* Both print 2 */
5     cout << "Vector: " << vector[1] << " Array: " << array
      [1];
6
7     /* answer = 42 */
8     int answer = vector[5]*array[5] + vector[5];
9
10    /* walking off array */
11    answer = vector[42];
12    answer = array[42];
```

Note that we never specified a size; this is not a mistake. Vectors are **dynamic**, meaning as you use them, they grow larger. No more guessing

how large your array might have to be! So something like this is perfectly legal.

```
1      vector<double> someDoubles = { 1.1, 2.2, 3.3, 4.4, 5.5 };
2
3      someDoubles.push_back(6.6);
4      someDoubles.push_back(7.7);
5
6      for (int i = 0; i < rand(); i++) {
7          someDoubles.push_back(i + i/10.0);
8      }
```

We started with `someDoubles` being of size 5 and growing up to sizes greater than 10 000. Below I will list some of the more important parts of vectors. This won't be comprehensize; for more information you can check documenation here, here, or here.

1.1 Accessing Elements

As we have already seen, accessing an element in a vector can be done with the accessor (the `[]` operator). Alternatively, you can use the `at(i)` method[1] instead.

In addition, there are two methods that might come in handy: `front()` and `back()`. They return the front element and the back element, respectively.

```
1      vector<string> assistants = { "Ian", "Illya", "Grant" };
2
3      cout << "The front assistant is " << assistants.front()
       << "\n"; // Ian
4      cout << "The back assistant is " << assistants.back() <<
       "\n"; // Grant
5      cout << "The middle assistant is " << assistants.at(1) <<
       "\n"; // Illya
```

1.2 Adding Elements

We have already seen `push_back(element)`, and for the purposes of this lab, this will suffice.

[1] Method means member function; it's more object-oriented way of saying it.

```
vector<string> assistants;

assistants.push_back("Illya");
assistants.push_back("Grant");
assistants.push_back("Ian");

for (int i = 0; i < 3; i++) {
    // prints Illya Grant Ian
    cout << assistants.at(i) << " ";
}
```

1.3 Removing Elements

There are only two ways we are going to talk about removing elements: `pop_back()` and `erase()`. `pop_back()` is relatively straightforward; it deletes the last element (exact opposite of `push_back()`).

```
vector<string> assistants = { "Illya", "Ian", "Grant" };

/* Removes Grant from assistants. */
assistants.pop_back();

/* Array now contains [Illya, Ian] */
```

This is useful, but it doesn't allow for arbitrary deletion. To do this, we use `erase`; however, there is a catch. In true C++ fashion, you cannot say `erase(index)` or `erase(element)`. To delete *by index*, the syntax is `erase(vector.begin() + index)` **where vector is the name of your vector variable.**

```
vector<string> assistants = { "Illya", "Ian", "Grant" };

/* Erase Grant */
assistants.erase(assistants.begin() + 2);

/* Erase Ian */
assistants.erase(assistants.begin() + 1);

/* Erase Illya */
assistants.erase(assistants.begin());
```

Note the order is important! If we did this in reverse, we would get very different results.

```
vector<string> assistants = { "Illya", "Ian", "Grant" };

/* Erase Illya */
assistants.erase(assistants.begin());

/* Erase Grant */
assistants.erase(assistants.begin() + 1);

/* segfault! There's only one element, Ian */
assistants.erase(assistants.begin() + 2);
```

Last, we can clear all the contents of the array with `clear`.

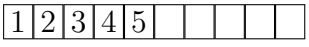
```
vector<string> assistants = { "Illya", "Ian", "Grant" };
string newAssistant;
/* Erase Grant */
assistants.erase(assistants.begin() + 2);

for (int i = 0; i < 1000; i++) {
    /* Assistants size is 1002 */
    newAssistant = getInferiorTA();
    assistants.push_back(newAssistant);
}

/* Assistant size is now 0 */
assistants.clear();
```

For deleting all occurrences, that's a more advanced topic. You can read about it here.

1.4 Size and Max Size

In addition, there are two methods that are quite useful: `size()` and `max_size()`. So, if you had the following vector 

, calling `size()` would return 5, but `max_size()` would return 10. For the purposes of this lab, you should not need `max_size()`.

2 The Assignment

For this assignment, you will be required to implement four functions:

1. Invite Guests

2. Welcome Guests
3. Exchange Presents
4. Kick Out

Any maintenance or helper functions you might want/need are up to you.

2.1 Invite Guests

Invite guests should **take no input and return a vector of type string**.

First, print out a string to signify you are taking in guests, and have `q` stop the input. Do not stop until you have input two guests. Simply take in strings (you can assume no spaces, using `cout`) that signify the guest's names.

Because the postal service is.. super fast.. on the holidays, one person might not get the invitation in time. At random, decide if the invitation is lost[2] and print out that the person's invitation (`cout` the name) was lost, but the other x number of people got their invitation (`cout` the number of people that got the invitation). If the invitation didn't get lost, just output something like "Everyone got the invitation!".

Return the guests's name vector.

2.2 Welcome Guests

Welcome guests should **take in the guests vector and return a vector of type string**.

Similarly to invite guests, this will take also take input; however, for every guest, there will be one gift. Continuously take in strings to signify the present's names into its own vector *in the order of the guests*[3]. Return it after every guest has input their presents.

For iterating through all the guests, **you must use size**[4].

[2] hint, hint: check the Hints section.

[3] As in, "Starikov"'s gift is "Socks" and "Ian"'s gift is a "Tesla", the arrays should be |Starikov|Ian| and |Socks|Tesla|

[4] As in, you can't pass in counters from input, no global variables, etc. You will solely rely on the vector's information size to iterate through the guests.

2.3 Exchange Presents

Exchange guests should **take in the guests vector and the presents vector by value, and not return anything**.

You will generate two random numbers, `from` and `to`, ranging from 0 to the number of presents − 1. You will use these values to determine the gift-giving relationship. `from` will be giving the gift, `to` will be receiving a gift. Afterward, erase `from` from the presents and gifts vector[5].

And yes, you must use size.

2.4 Kick Out

Kick out should **take in the guests vector and the presents vector by reference, and not return anything**.

Announce you are kicking people out individually (iterating through all of them), and clear[6] both the arrays.

2.5 Main

In main, simply call these functions in the order of

1. Invite Guests
2. Welcome Guests
3. Exchange Presents
4. Kick Out

so that the program acts as expected.

3 Deliverables

Please submit your file(s) with `cssubmit` using the provided sample input; you may check up against the sample output if you so wish.

[5] hint, hint: actually use `erase`

[6] hint, hint: use `clear`

3.1 Sample Input

```
q
Jarus
q
Price
Leopold
Wisely
Armita
Rhodes
q
Beer
Cow
AppleWatch
MoreBeer
Nothing
HandGrenade
```

3.2 Sample Output

```
Wisely gave Jarus a MoreBeer
Rhodes gave Armita a HandGrenade
Armita gave Price a Nothing
Leopold gave Jarus a AppleWatch
Jarus gave Price a Beer
Price gave Price a Cow
You kicked Jarus out!
You kicked Price out!
You kicked Leopold out!
You kicked Wisely out!
You kicked Armita out!
You kicked Rhodes out!
```

4 Hints

- Because it's not the point of the lab, random value generation is given. Don't forget `srand(time(NULL))` at the top of main and the relevant

header files.

```
1     template<typename T, typename S>
2     T randomArbitrary(const T lower, const S upper) {
3         return lower > upper ? 0 : lower + rand() % (upper -
      lower + 1);
4     }
```

- `randomArbitrary(false, true)` will return either true or false; however it must be in the order `false, true`.
- The solution contains 58 lines of C++ code and 16 lines of template/-header code (excluding comments, blank lines, etc.). If you solution is significantly more (or less!), there may be a problem.

Happy Holidays from the Computer Science Department!