C++: vector & iterators

Ben Langmead

ben.langmead@gmail.com

www.langmead-lab.org

C++: vector & iterators

`vector` is an array that automatically grows/shrinks as you need more/less room

- Use `[x]` or `.at(x)` to access an element, like `std::string`
    - `.at(x)` does bounds check, like `std::string`
- Allocation, resizing, deallocation handled by C++
- Like Java's `java.util.ArrayList` or Python's `list` type

`#include <vector>` to use it

`std::string` is like (but not same as) `std::vector<char>`

C++: vector

Declare a vector:

```
using std::vector;
vector<std::string> names;
```

Add elements to vector (at the back):

```
names.push_back("Alex Hamilton");
names.push_back("Ben Franklin");
names.push_back("George Washington");
```

Print number of items in vector, and first and last items:

```
cout << "Size=" << names.size()
     << ", first=" << names.front()
     << ", last=" << names.back() << endl;
```

C++: `vector`

`vector` handles memory for you

Behind the scenes, dynamic memory allocations are needed both to create `strings` and to add them to the growing `vector`:

```
names.push_back("Alex Hamilton");
names.push_back("Ben Franklin");
names.push_back("George Washington");
```

Allocations happen automatically; everything (`vector` & `strings`) is deallocated when names goes out of scope

C++: vector

```cpp
#include <iostream>
#include <vector>
#include <string>

using std::vector; using std::string;
using std::cin;    using std::cout;
using std::endl;

int main() {
    vector<string> names;
    names.push_back("Alex Hamilton");
    names.push_back("Ben Franklin");
    names.push_back("George Washington");

    cout << "First name was " << names.front() << endl;
    cout << "Last name was " << names.back() << endl;
    // names.front() is like names[0]
    // names.back() is like names[names.size()-1]

    return 0;
} // names goes out of scope and memory is freed
```

C++: vector

```
$ g++ -c names_1.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o names_1 names_1.o
$ ./names_1
First name was Alex Hamilton
Last name was George Washington
```

C++: vector

Two ways to print all elements of a vector. With indexing:

```
for(size_t i = 0; i < names.size(); i++) {
    cout << names[i] << endl;
}
```

With an *iterator*:

```
for(vector<string>::iterator it = names.begin();
    it != names.end();
    ++it)
{
    cout << *it << endl;
}
```

C++: `vector`

Iterators are “clever pointers” that know how to move over the components of a data structure

Structure could be simple (linked list) or complicated (tree)

They are safer & less error-prone than pointers; pointers cannot generally be used with STL containers

C++: iterators

For STL container of type `T`, iterator has type `T::iterator`

```
for(vector<string>::iterator it = names.begin();
    it != names.end();
    ++it)
{
    cout << *it << endl;
}
cout << endl;
```

Here, iterator type is `vector<string>::iterator`

C++: iterators

Looking harder at the loop:

```
for(vector<string>::iterator it = names.begin();
    it != names.end();
    ++it)
```

First line: declares `it`, sets it to point to first element initially

Second: stops loop when iterator has moved past vector end

Third: tells iterator to advance by 1 each iteration

- `++it` isn't really pointer arithmetic; ++ is "overloaded" to move forward 1 element *like* a pointer

C++: iterators

Looking harder at the body:

```
    cout << *it << endl;
```

`*it` is *like* dereferencing; `*` is “overloaded” to get the element currently pointed to by the iterator

For `vector`, `*it`’s type equals the element type, `string` in this case

C++: vector

```cpp
#include <iostream>
#include <vector>
#include <string>

using std::vector; using std::string;
using std::cin;    using std::cout;
using std::endl;

int main() {
    vector<string> names;
    names.push_back("Alex Hamilton");
    names.push_back("Ben Franklin");
    names.push_back("George Washington");
    for(vector<string>::iterator it = names.begin();
        it != names.end();
        ++it)
    {
        cout << *it << endl;
    }
    return 0;
}
```

C++: vector

```
$ g++ -c names_2.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o names_2 names_2.o
$ ./names_2
Alex Hamilton
Ben Franklin
George Washington
```

C++: vector

Iterate in *reverse* order by using `T::reverse_iterator`, `.rbegin()` and `.rend()` instead:

```
for(vector<string>::reverse_iterator it = names.rbegin();
    it != names.rend();
    ++it)
{
    cout << *it << endl;
}
```

```
$ g++ -c names_3.cpp -std=c++11 -pedantic -Wall -Wextra
$ g++ -o names_3 names_3.o
$ ./names_3
George Washington
Ben Franklin
Alex Hamilton
```

C++: vector

See C++ reference for more `vector` functionality

- www.cplusplus.com/reference/vector/vector/

Don't miss:

- `front` – get firstelement
- back – get last element
- pop_back – return and delete final element
- begin/end – iterators for beginning/end
- cbegin/cend – `const_iterators` for beginning/end
- rbegin/rend – `reverse_iterators` for beginning/end
- `erase`, `insert`, `clear`, `at`, `empty` – just like `string`