MTH 4300, Lecture 19a

Actual Vectors

E Fink

April 21, 2025

1. Actual Vectors

Let's talk about an actual type that C++ supports, which does what `Lists` try to do: `vectors`! These are part of the *Standard Template Library*, which contains many types of data containers.

To use `vectors`, you `#include <vector>`. To declare a `vector`, write

```
vector<my_type> x; // my_type is any data type
```

or

```
vector<double> x(20, 3.14); // vector with pi 20 times
```

or

```
vector<int> x = {1, 2, 4, 8, 10}; // Initializer list
```

Among other things, vectors support the following methods:

- `.push_back()`, which works like `.append`.
- `.at()`, which works like the normal index operator `[]`, but which checks for out-of-bounds reads (it *throws an exception* in this case – which is far better than the code silently running incorrectly). This safety comes with a performance penalty.

L19ax1_vect.cpp

Actual Vectors

To explain a couple more methods, we need to discuss *iterators*. Iterators are objects, whose main data member is simply a pointer to an element of a `vector`.

For a given `vector` named `x`, then, for example,

`x.begin()` would be an iterator pointing to the first element of the vector;
`x.begin() + 2` would be an iterator pointing to the third element of `x`;
`x.end()` would be an iterator pointing to one entry past the end of the vector.

In all cases, you would use `*` in front to access the value of the element itself.

You can also set iterator variables using a declaration like

```
auto it = x.begin();
```

where `auto` is a declaration for variables whose type can be deduced from its value.

You can see how to use these in loops in **L19ax2_iter.cpp** .

Actual Vectors

Iterators can also be used in the following methods:

- `x.erase(it1, it2);` will erase the elements from `it1`, up to, but not including, `it2`. Using a single iterator will simply erase a single element.
- `x.insert(it, value);` will insert `value` into the array at the element that `it` is pointing to, pushing everything else back one position.
- `x.insert(it, other_it1, other_it2 );` will take all the elements between iterators `other_it1` and `other_it2` for some other vector, and insert them in at iterators `it` in vector `x`.

L19ax3_middle.cpp

Actual Vectors

A quick warning about iterators: if you perform insertions or erasures on a vector, existing iterators on that vector can get *invalidated*: they may no longer point to the element that you expect them.

Why? Suppose that you have an iterator – basically, a pointer – holding the address of an entry of the underlying dynamic array. If you insert a new element into the vector, if capacity is reached, then there might be a "reserve()" call, with all the vector's contents migrating to an entirely new dynamic array, leaving the iterator pointing to the old, deallocated array.

So, don't use an iterator to traverse a vector if the vector changes size as you iterate – or at least, use extreme caution if you do.

L19ax4_inval.cpp