Standard Library Containers

Kate Gregory

@gregcons www.gregcons.com/kateblog

Standard Containers Save

You don’t have to write the container

Lots of great ones in the Standard Library

Work great with standard algorithms too

They know their size, manage themselves

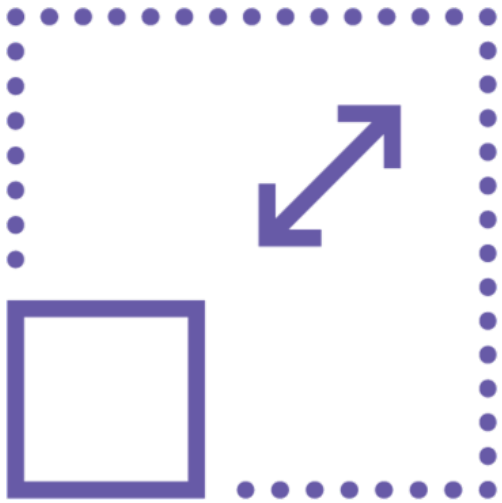
You Don't Have to Manage Memory

When the container destructs, so do its contents

It won't call delete on raw pointers

But it will do the right thing for smart pointers

You Don't Have to Handle Special Cases



They resize themselves when you add elements

- They throw exceptions when you access past the end, before the beginning

You never give up type safety

Can’t push an integer onto a vector of strings

Protects you from errors of thought

People who read your code know what it does

If You Just Want One Rule

Use vector

Grows itself when you add new items

Can traverse with an iterator or random access with []

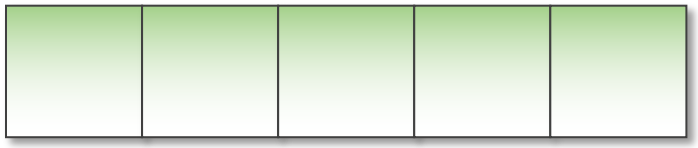
Cleans up after itself

Decent Performance

Actually better than you think

Keeps elements consecutive in memory, makes it++ just a memory add

Random access is possible (and fast)



```
vector<Employee> newHires;

vector numbers{ 0,1,2 };

numbers.push_back(-2);


numbers[0] = 3;

int num = numbers[2];

for (int i : numbers)

{

   cout << i << '\n';

}
```

- You can specify the type to be kept in the vector
- But you don't always have to
- Add an element to the end with push_back – vector will resize and copy if needed
- Access an element (reading or writing) with [] – 0 based
- Ranged for loop does all the elements of the vector

Vectors That Never Grow

Eg 12 elements, one for each month

Using std::array expresses that clearly

May improve performance as well

list

Implements a linked list

Saves the copying when new items are added

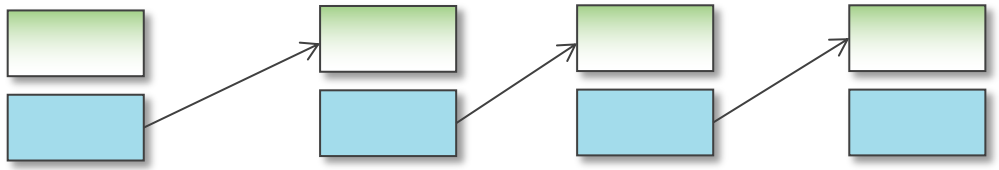
Changing container is pretty easy

list Can Be Faster or Slower

Less copying

More expensive to traverse

- it++ is an indirection

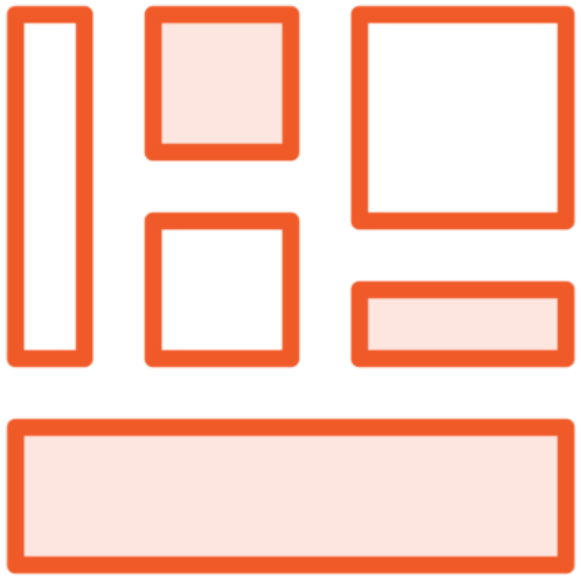


Keeping it sorted? Finding place to insert is a lot of it++

Never assume list will be faster

- Measure and see

Know Your Collections



vector, list, map are the most common dynamic collections

- array if the collection never grows or shrinks
- map is kept sorted, keys must be unique

multimap allows collisions

unordered_map and unordered_multimap have quicker adds

stack, queue, dequeue, priority_queue

Summary

Use vector

Not sure? Use vector

Think you need list? Go ahead and use vector

- Then measure a real load, try swapping in list and compare real perf

Think you need map, stack, queue, etc?

- You probably do, go ahead and use them
- Do not invent your own based on vector

Write your code to make switching containers easy

- auto
- begin() and end()
- Algorithms