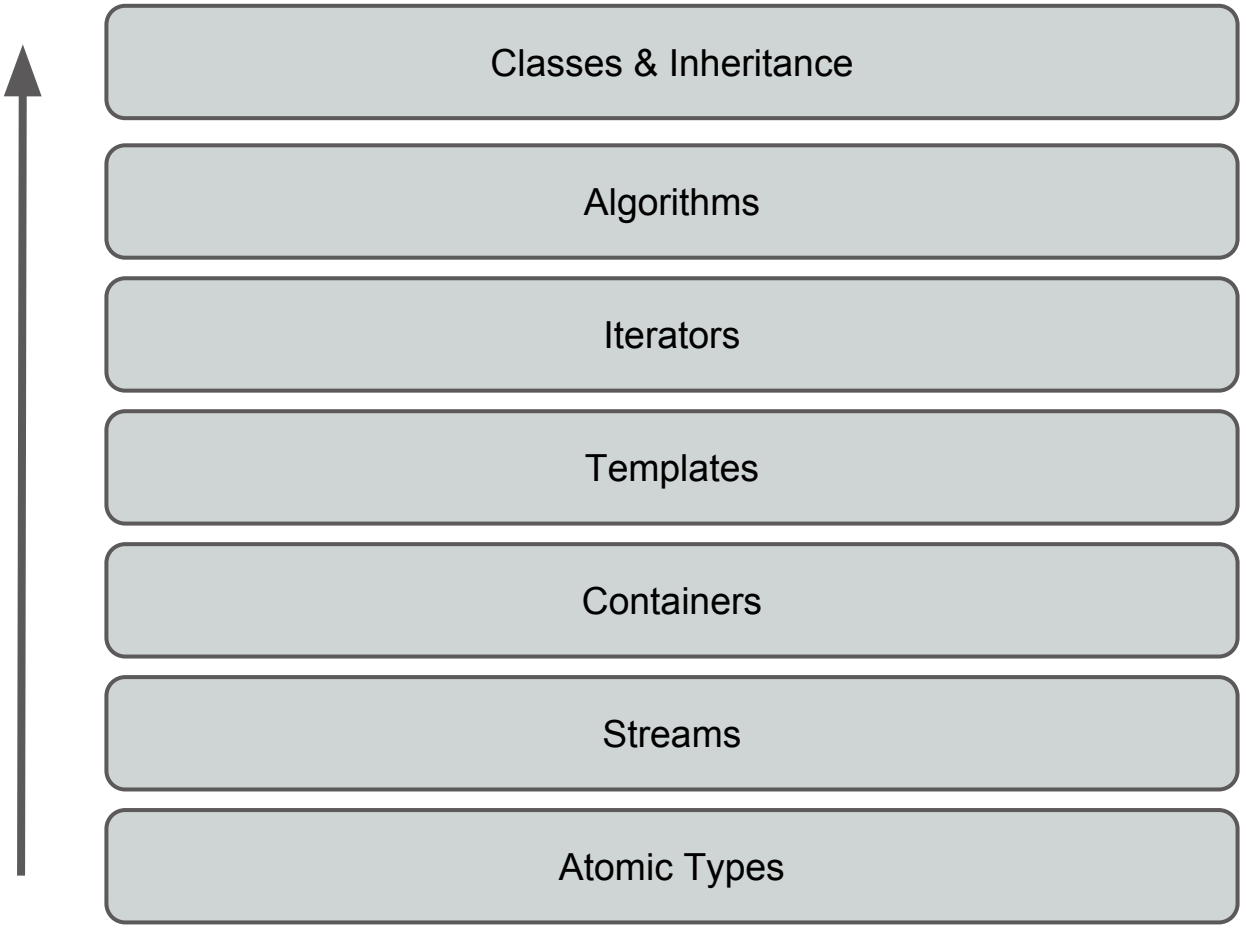
C++ Style

Cristian Cibils
ccibils@stanford.edu

The Design of C++



Streams

- Big takeaways
  - Don’t mix getline() and >>
  - To un-fail a stream:
    - close()
    - clear()
    - open()
  - In order to send an object through a stream, you have to overload the << operator, or define a string representation function

Containers

- Big takeaways:
  - Sequence Containers:
    - Design (push(), push_back(), pop(), pop_back())
    - Won't check boundaries for us! (most of the time)
    - All of the common ones are implemented
  - Associative Containers:
    - .insert(), and .erase()
    - We met the pair<Key, Value> type when iterating through maps
    - Maps have default values created when you use the [] operator!

Templates

- Big takeaways:
  - template<typename MY_TYPE> ...
  - You can templatize everything! Functions and Classes!
  - For functions, the compiler will often implicitly figure out the templatization for you
  - For classes, we often have to be more specific
  - The STL containers are all templatized!

Iterators

- Big takeaways
  - Iterators let C++ deal with algorithms without caring what the underlying data structures are
  - auto is your friend :)
  - A simple way to understand iterators, is that they are pointers to elements inside the containers, and the operations we perform on them (like ++) does not necessarily mean that the objects (or even the pointers) are sequential

Algorithms

- Big takeaways:
  - Most of the common stuff that you'll have to do, is already done for you
  - Sorting, min/max, conditional checks, copying, swapping, operations for iterators, etc
  - Combine this with lambdas and you're golden
  - Lambdas:
    - Syntax: [captured_variables](parameters) {

      //body

      };
    - Can capture stuff by reference, declare them using auto

Classes and Inheritance

- Big takeaways:
  - Where C++ got started, where C++ gets its power
  - Constructors:
    - Always use initialization lists! (especially handy for internal constants)
  - Const:
    - Const is extremely powerful and annoying, use it!
  - Operator overloading:
    - Defined like regular functions!
  - Templatizing
    - no .cpp file
    - Templatize all the things! (every function you define has to carry the template header)

Classes and Inheritance

- Big takeaways part 2:
  - Inheritance:
    - class Derived: public Base { // }
    - Always call the Base class' constructor in the Derived class' constructor
    - Pretty much always make the destructors virtual!
    - You can inherit from multiple classes, but be careful because it can get really messy!

What is style?

“Code that looks good”

What is style?

- There is a sense in which we want to make code look beautiful
  - As programmers, this is our art, and we should definitely strive for it
- But there is also a more practical description
  - Consistency
  - Convention
  - Enforcement

What is *practical* style?

"The consistent enforcement of convention"

Why do we need style?

- As code and programs grow, and more people come into teams, you begin to really focus on productivity
- If you want to be productive, then the less surprises (or variation) in the codebase, the better
- That is, you want your codebase to always be consistent and predictable, throughout all programmers, so that productivity can increase

Okay fine, so what does that look like?

- The fuzzy version is:
  - “There’s really no way to tell you what good style is, it’s just code that’s beautiful”
- The practical version is:
  - “Here’s a list of requirements you have to comply with”
  - We’ll get into these in a sec

Beautiful Code

```
template<class FwdIt, class Compare = std::less<>>
void quickSort(FwdIt first, FwdIt last, Compare cmp = Compare
{})
{
    auto const N = std::distance(first, last);
    if (N <= 1) return;
    auto const pivot = std::next(first, N / 2);
    std::nth_element(first, pivot, last, cmp);
    quickSort(first, pivot, cmp);
    quickSort(pivot, last, cmp);
}
```

Some of the practical requirements

- Google has a great public Style guide
  - Focuses on simplicity and readability
  - Somewhat polarizing
- We don’t have time to cover all of it, but we can take a look in https://goo.gl/7lNvP2

Some stuff worth mentioning

- Naming:
  - Constants:
    - kNameOfConstant;
  - Variables:
    - Names as descriptive as humanly possible
    - Lower case, separated by an underscore
    - Instance Variables in a class:
      - Add another underscore at the end!
  - Types:
    - Name of type must be Capital CamelCase
    - ThisIsAnExample

Some stuff worth mentioning

- Naming:
  - Functions:
    - Getters:
      - If I have an instance variable called varname_, then the getter for it should be varname();
    - Setters:
      - Using varname_ from above, the setter would be, set_varname(ElemType new_varname);
    - Other:
      - CapitalCamelCase again!

Some stuff worth mentioning

- Function parameters:
  - MyFunc(const InputParameterObject& i, // input only
    OutputParameter* o); // output only
  - Always order input parameters to the left, and output parameters to the right. Input parameters are const references, and output parameters are pointers?
  - Why pointers?
    - Because when you call the function, you see
    - Container to_be_filled;
      Fill(SomeOtherContainer, &to_be_filled);
    - That's telling you something!

Some stuff worth metioning

- Classes:
  - Declaration Order:
    - Typedefs and enums
    - Constants
    - Constructors
    - Destructor
    - Methods
    - Variables

Some stuff worth mentioning

- Misc
  - Use smart pointers as much as you can!
  - Always cast with static_cast<>
  - Keep lines at 80 chars long max
    - Somewhat arbitrary, but makes sense
  - Keep functions at ~10 lines
  - Comment as little as you can!
    - Always separate you comment from the // with at least a space
  - Don’t end your lines with a blank space!
  - Always put {} around your conditionals and loops

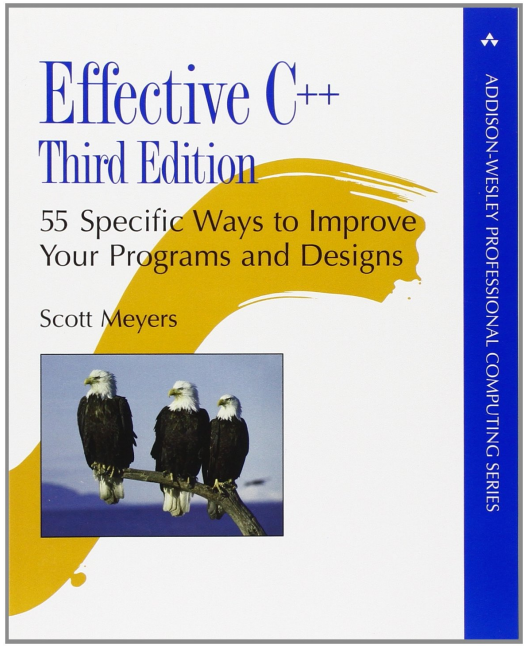
What's New in C++

C++14 just came out!

- auto return type
- variables can be templates
- make_uniqe
- other small changes

Where to Go From Here

- We have just touched the surface of C++ these past 10 weeks
- For more C++ knowledge, I HIGHLY recommend *Effective C++*



Where to Go From Here

- cplusplus.com
- cppreference.com

Closing thoughts