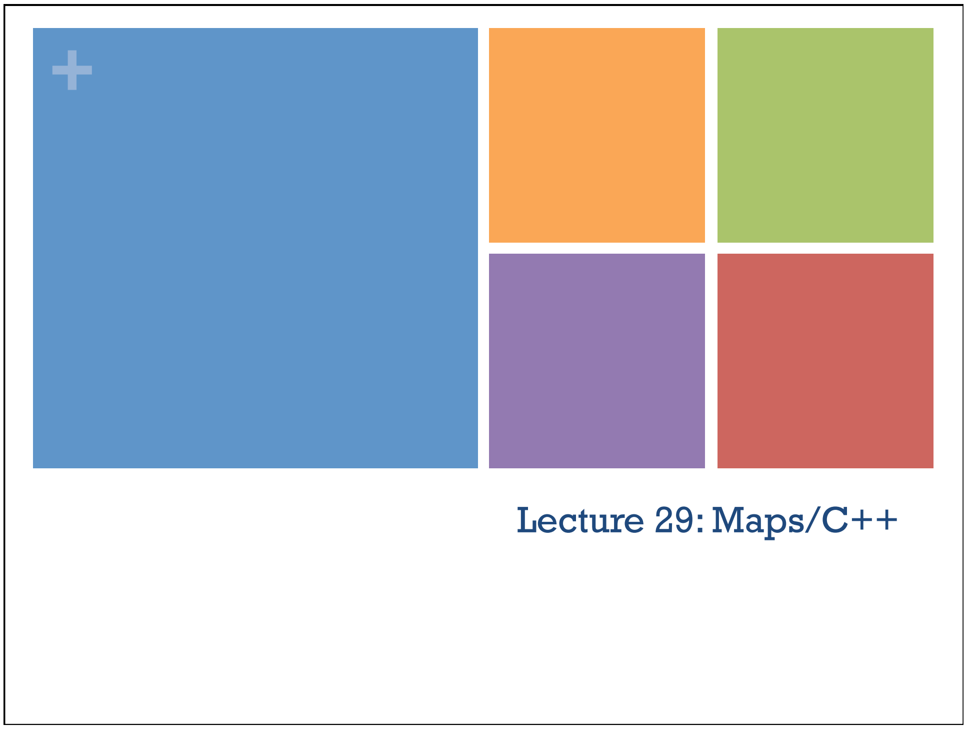
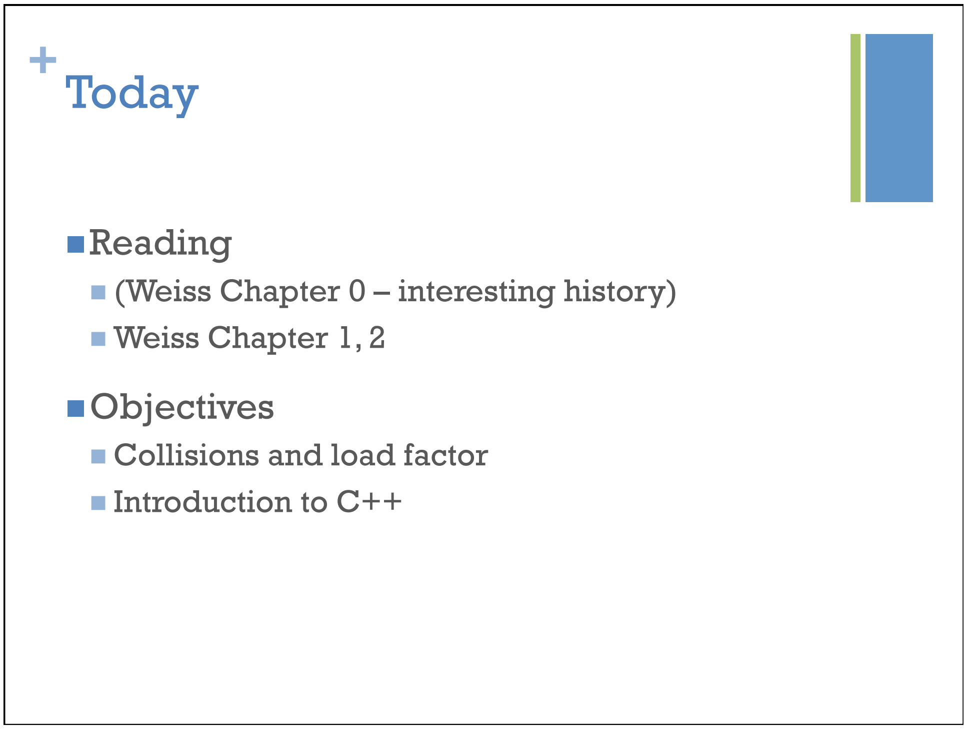


+

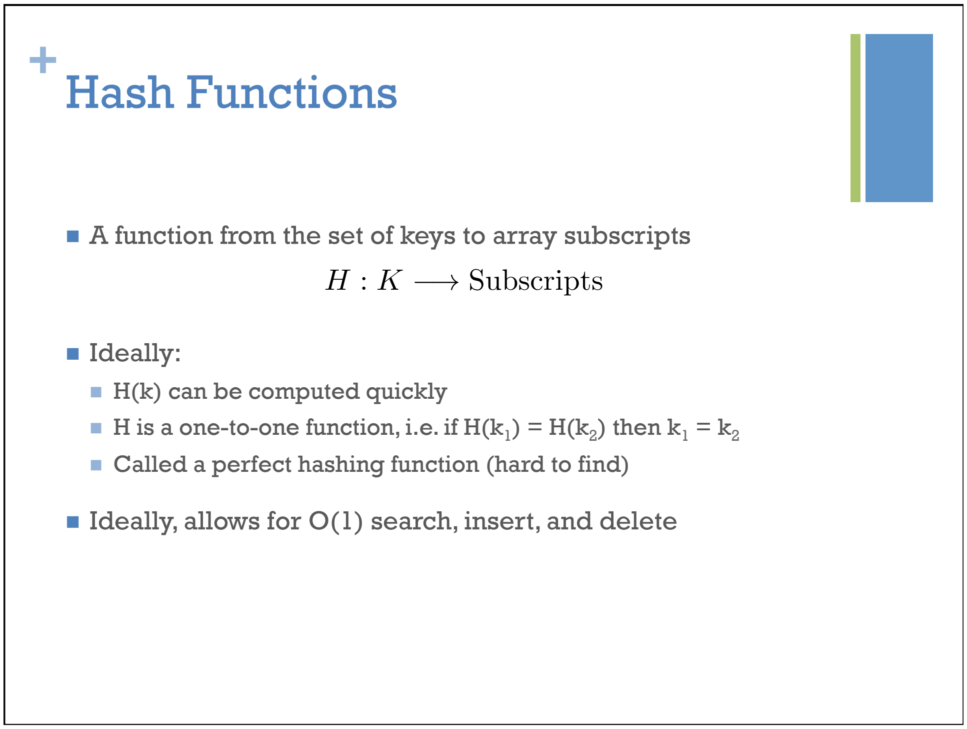
Lecture 29: Maps/C++



+

Today

- Reading
  - (Weiss Chapter 0 – interesting history)
  - Weiss Chapter 1, 2
- Objectives
  - Collisions and load factor
  - Introduction to C++

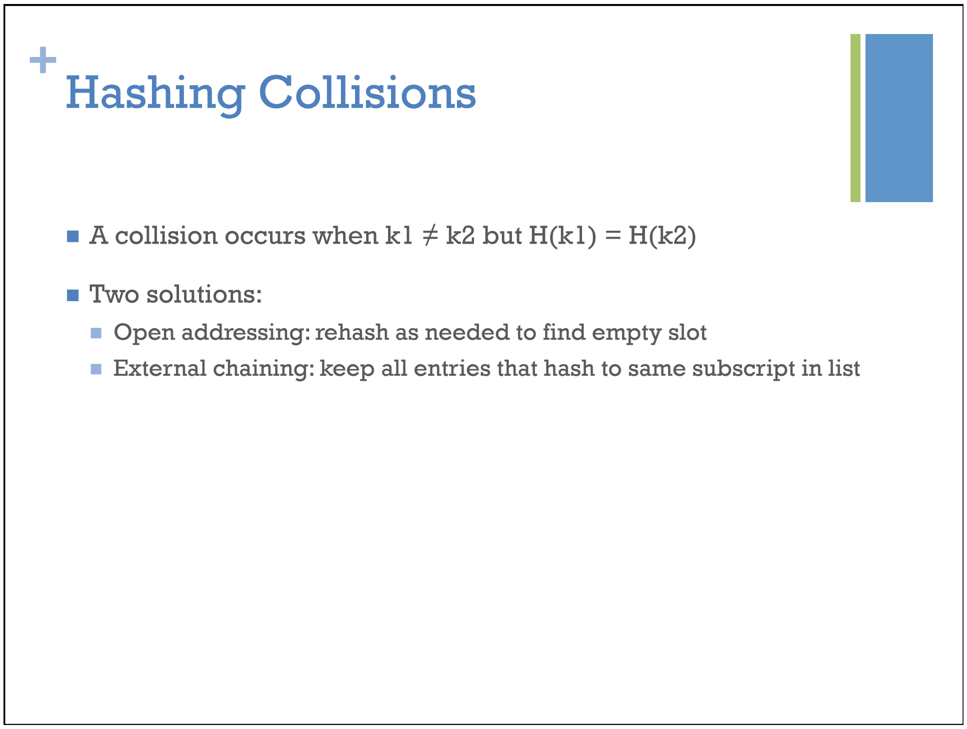


+ Hash Functions

- A function from the set of keys to array subscripts

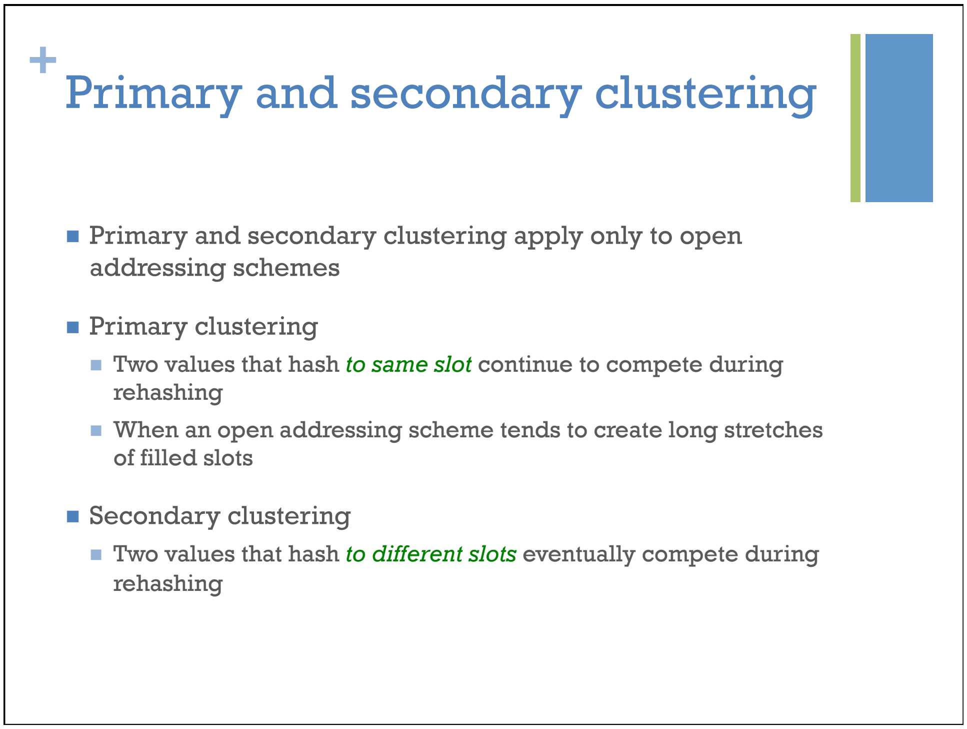
$$H : K \longrightarrow \text{Subscripts}$$

- Ideally:
  - H(k) can be computed quickly
  - H is a one-to-one function, i.e. if $H(k_1) = H(k_2)$ then $k_1 = k_2$
  - Called a perfect hashing function (hard to find)
- Ideally, allows for O(1) search, insert, and delete



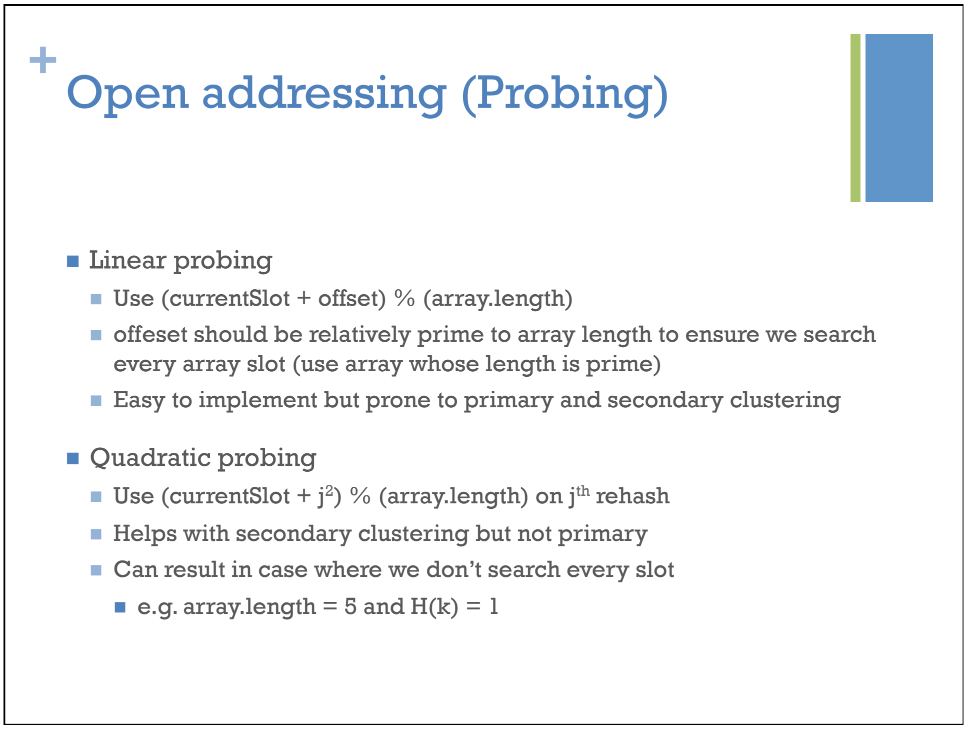
+ Hashing Collisions

- A collision occurs when $k1 \neq k2$ but $H(k1) = H(k2)$
- Two solutions:
  - Open addressing: rehash as needed to find empty slot
  - External chaining: keep all entries that hash to same subscript in list



+ Primary and secondary clustering

- Primary and secondary clustering apply only to open addressing schemes
- Primary clustering
  - Two values that hash *to same slot* continue to compete during rehashing
  - When an open addressing scheme tends to create long stretches of filled slots
- Secondary clustering
  - Two values that hash *to different slots* eventually compete during rehashing



+ Open addressing (Probing)

- Linear probing
  - Use (currentSlot + offset) % (array.length)
  - offeset should be relatively prime to array length to ensure we search every array slot (use array whose length is prime)
  - Easy to implement but prone to primary and secondary clustering
- Quadratic probing
  - Use (currentSlot + j^2) % (array.length) on j^{th} rehash
  - Helps with secondary clustering but not primary
  - Can result in case where we don't search every slot
    - e.g. array.length = 5 and H(k) = 1

Open addressing (Probing)

- Double hashing
  - Use second hash function to determine the offset
  - e.g. Suppose we use $H_1(x)$ = x mod N for the array subscript and $H_2(x)$ = x mod (N-2) + 1 for offset for N=5
  - Helps with primary and secondary collisions

Collisions!	Different offsets	Next subscripts to try
$H_1(1) = 1$	$H_2(1) = 2$	1 + 2
$H_1(6) = 1$	$H_2(6) = 1$	1 + 1
$H_1(11) = 1$	$H_2(11) = 3$	1 + 3

External Chaining

- Each slot in table (array) holds unlimited number of entries
  - Each slot contains a list data structure (e.g. linked list)
  - Each list should be short
  - No elements hashed can be greater than size of array
  - Avoids secondary clustering

+ Analysis of Hashing

- The load factor is given by the ratio

$$\alpha = \frac{\text{number of values stored in table}}{\text{size of table}}$$

- Open addressing (alpha is <= 1)
- External chaining (alpha can be greater than 1)
- The higher the load factor, the more likely you are to have collisions

+ Analysis of Hashing

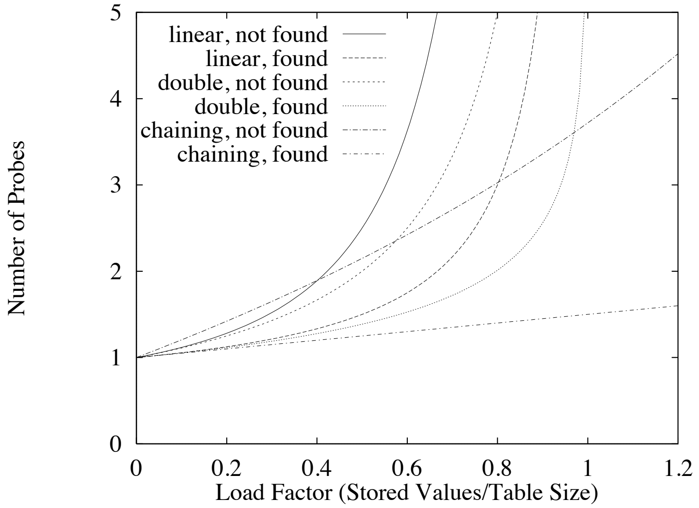
linear and quadratic probing

Method	Successful	Unsuccessful
Linear probes	$\frac{1}{2}\left(1+\frac{1}{(1-\alpha)}\right)$	$\frac{1}{2}\left(1+\frac{1}{(1-\alpha)^2}\right)$
Double hashing	$\frac{1}{\alpha}\ln\frac{1}{(1-\alpha)}$	$\frac{1}{1-\alpha}$
External chaining	$1+\frac{1}{2}\alpha$	$\alpha + e^{\alpha}$

Note typo in text

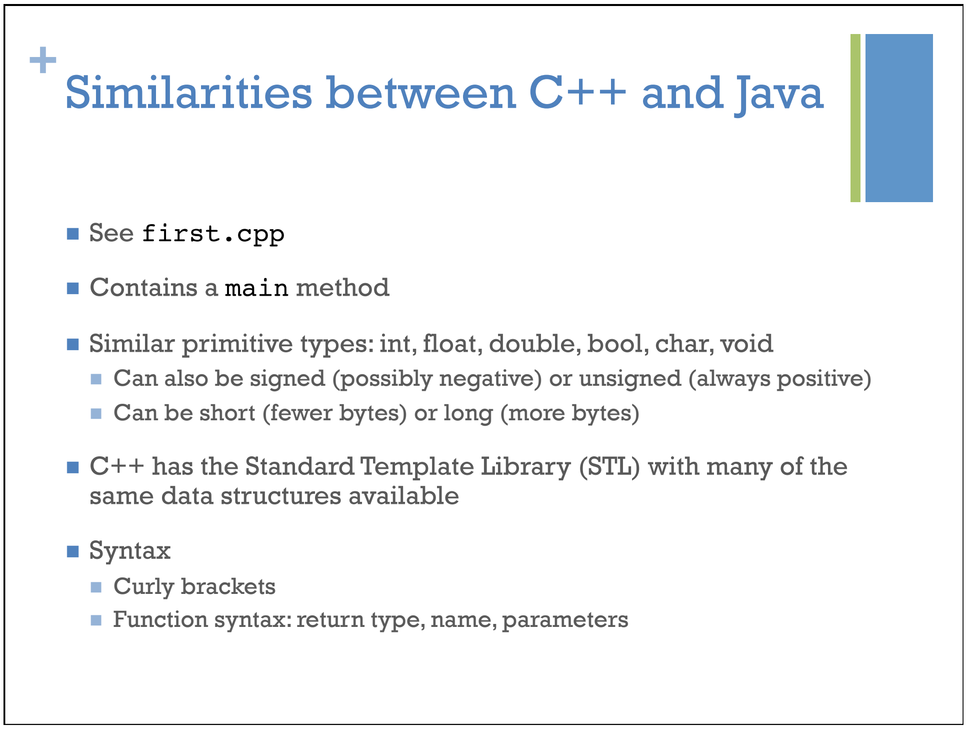
alpha = 0.9	Successful	Unsuccessful
Linear probes	5.5	50.5
Double hashing	2.6	10
External chaining	1.45	3.3

+ Analysis of Hashing



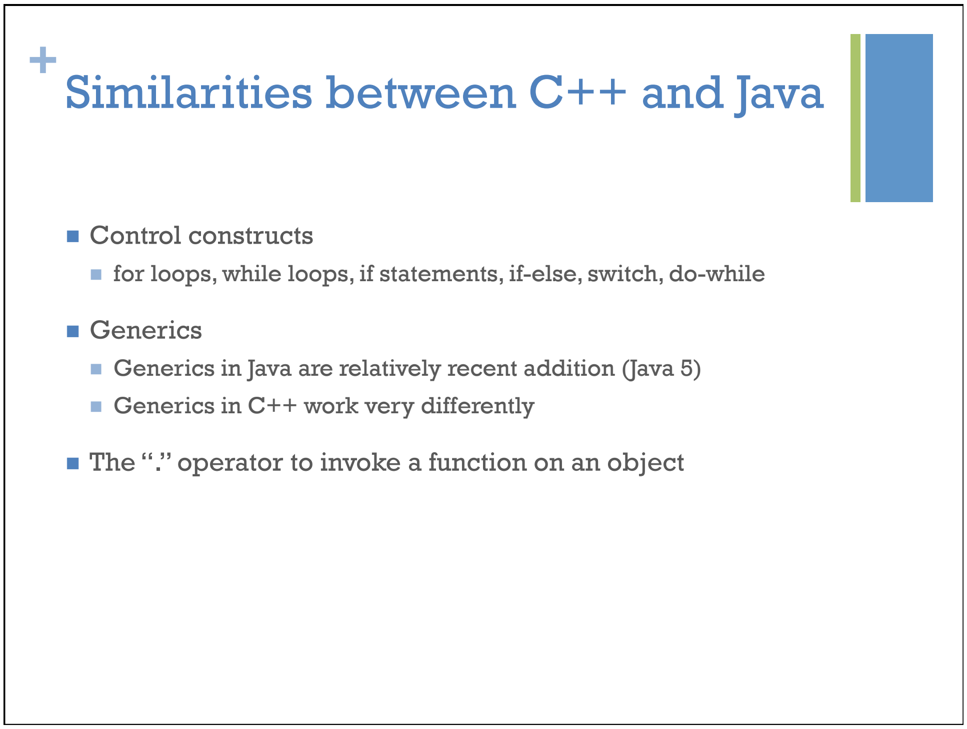
+ Extremely simplified history!

- C was developed in 1972
  - Designed for systems programming
  - Provides low-level access to memory
  - Extremely popular still today
  - Fast
- C++ developed in mid 70's by Bjarne Stroustrup
  - C with object oriented support
  - Backwards compatible with C
  - Still fast and still widely used today
- Java developed by Sun Microsystems in 90s
  - Uses C/C++ syntax
  - Explicitly disallows "bad programming"
  - Bytecode runs on virtual machine



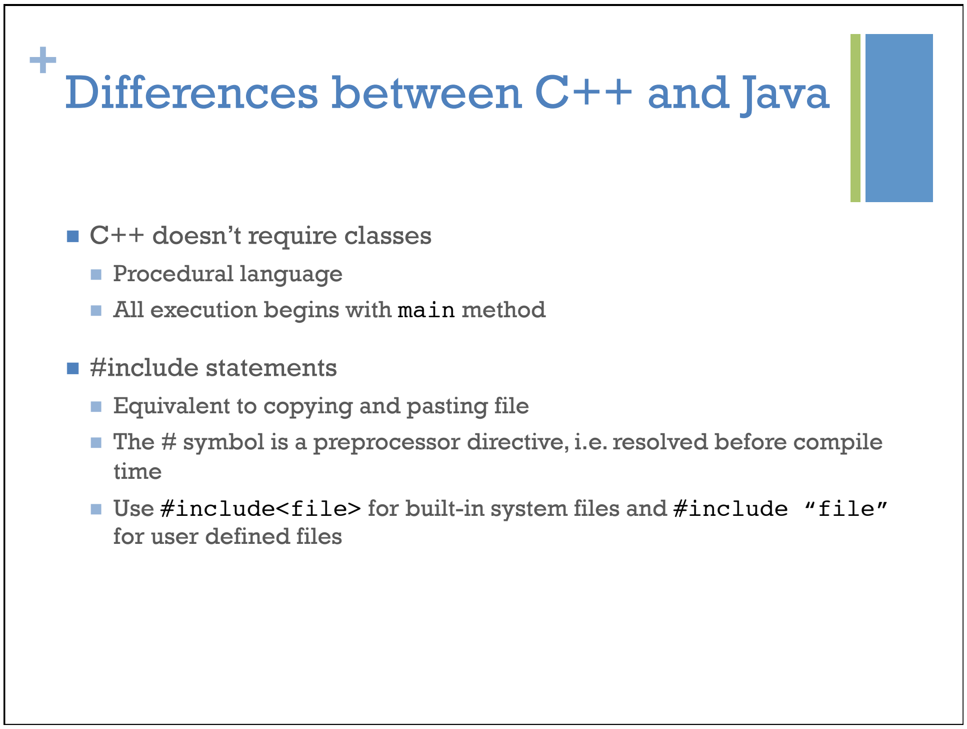
+ Similarities between C++ and Java

- See `first.cpp`
- Contains a `main` method
- Similar primitive types: int, float, double, bool, char, void
  - Can also be signed (possibly negative) or unsigned (always positive)
  - Can be short (fewer bytes) or long (more bytes)
- C++ has the Standard Template Library (STL) with many of the same data structures available
- Syntax
  - Curly brackets
  - Function syntax: return type, name, parameters



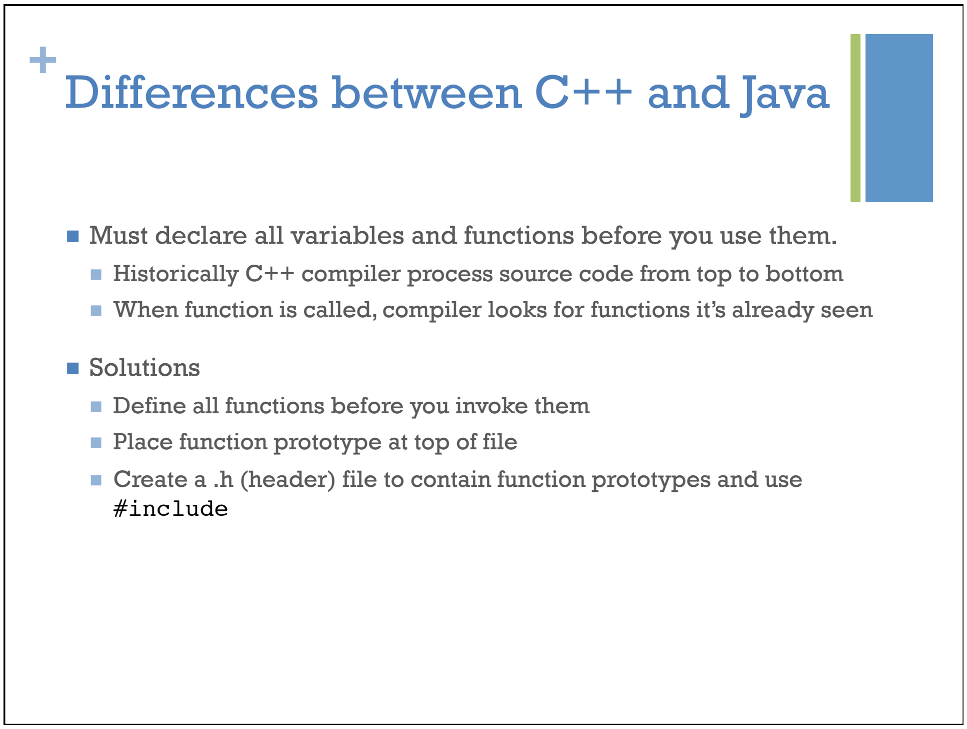
+ Similarities between C++ and Java

- Control constructs
  - for loops, while loops, if statements, if-else, switch, do-while
- Generics
  - Generics in Java are relatively recent addition (Java 5)
  - Generics in C++ work very differently
- The "." operator to invoke a function on an object



+ Differences between C++ and Java

- C++ doesn't require classes
  - Procedural language
  - All execution begins with `main` method
- #include statements
  - Equivalent to copying and pasting file
  - The # symbol is a preprocessor directive, i.e. resolved before compile time
  - Use `#include<file>` for built-in system files and `#include “file”` for user defined files



+ Differences between C++ and Java

- Must declare all variables and functions before you use them.
  - Historically C++ compiler process source code from top to bottom
  - When function is called, compiler looks for functions it's already seen
- Solutions
  - Define all functions before you invoke them
  - Place function prototype at top of file
  - Create a .h (header) file to contain function prototypes and use `#include`

+ Using the `std` namespace

- Namespaces are a generalization of packages
  - Named region of code contained in curly brackets
  - Helps disambiguate between variables and functions with same name
- The `std` namespace
  - Always have to specify
  - In C++, the `vector` type is in the `std` namespace
- To use `std`
  - Write `using std namespace;` at top of file
  - Use `::` operator, e.g. `std::vector` or `std::cout`

the "using" keyword is similar to import statement in Java

+ Object management

- In Java, most types are Objects
  - Except for primitives such as int, double, boolean, etc
- In C++, everything is a primitive
  - Allocated on the stack not the heap
  - Allocate from heap using `new` keyword explicitly
- *Changes the semantics of assignment and parameter passing*
  - Assignment means copying!
  - Parameters are copied (called pass-by-value)!
  - Variables are no longer names with arrows pointing to the same memory location!