

# Advanced Programming

ACSE-5: Lecture 3

Adriana Paluszny

Royal Society University Research Fellow

# Overview

- Create objects
  - STL library
  - Debug
- 
- We start by looking at many interconnected concepts that will advance your understanding of object orientated libraries
  - True understanding will require additional reading (homework) and independent coding
  - This class tries to marry different levels of progression catering for starting and intermediate level programmers

# Objects

- These are ways of creating new types beyond the native types
- Create objects:
  - class
  - enum
  - typedef
  - And... templates

These are “typename”s.

---

**simple-type-specifier:**

```
::opt nested-name-specifieropt type-name
::opt nested-name-specifier template simple-template-id

char
char16_t C++0x
char32_t C++0x
wchar_t
bool
short
int
long
signed
unsigned
float
double
void
auto C++0x
decltype-specifier C++0x
```

**type-name:**

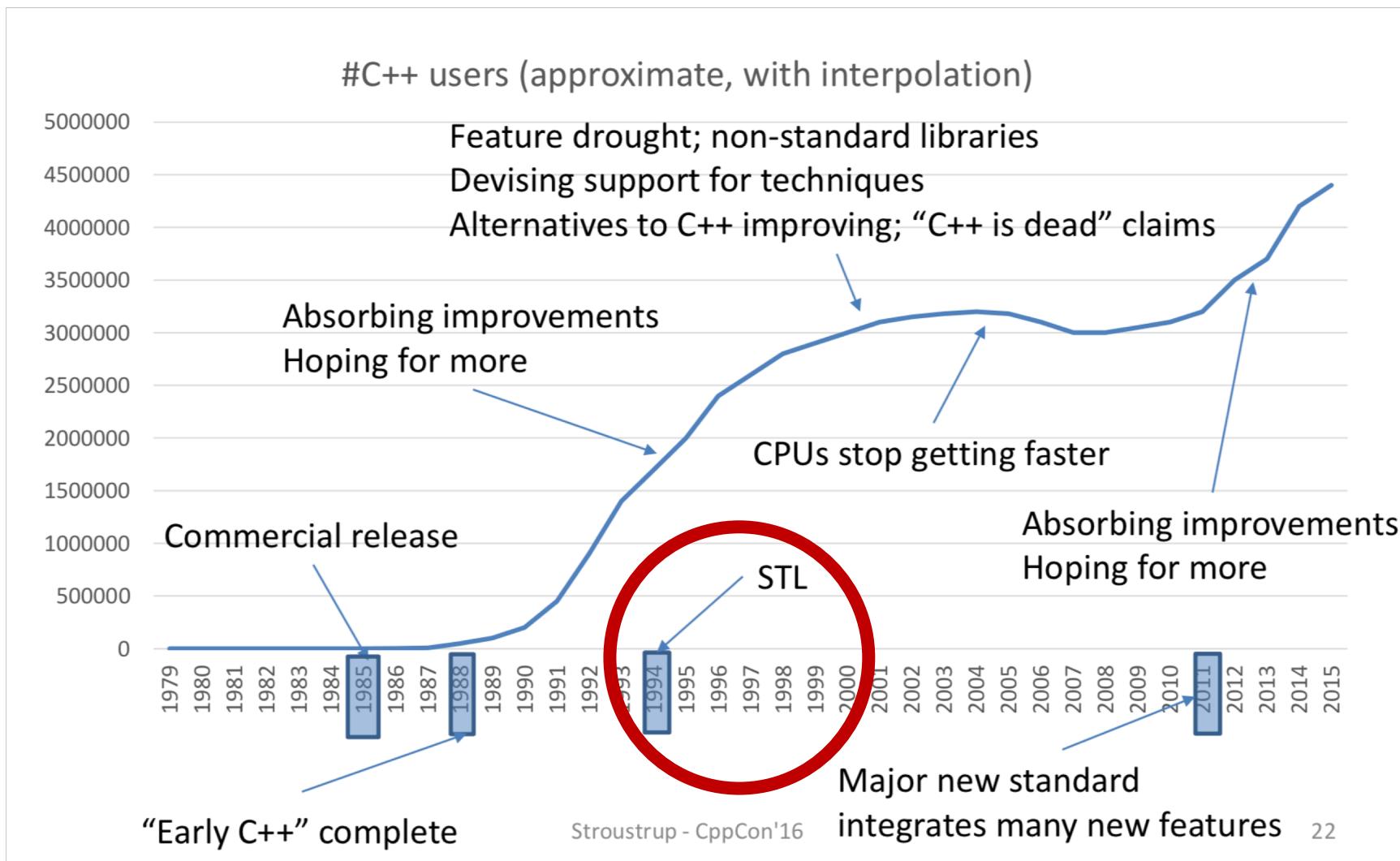
```
class-name
enum-name
typedef-name
simple-template-id C++0x
```

**decltype-specifier:**

```
decltype ( expression ) C++0x
```

---

# “C++ Success” by Bjarne Stroustrup



# The Standard Template Library, or STL

STL is a C++ library of container classes, algorithms, and iterators;

it provides many of the basic algorithms and data structures of computer science.

The STL is a generic library, meaning that its components are heavily parameterized: almost every component in the STL is a template.

You should make sure that you understand how templates work in C++ before you use the STL.

-SGI Introduction to the Standard Template Library

# STL Library

- Code reuse, abstraction and optimization of containers and algorithms
- Efficient (fast and less resources)
- Accurate (less bugs)
- Readable code
- Standardised
- A good example of what a library should be

# Types of Containers

- Sequence containers (arrays and linked lists)
  - Vector, deque, list, forward list
- Associative containers (binary tree)
  - Set, multiset
  - Map, multimap
- Unordered containers (hash table)
  - Unordered set/multiset
  - Unordered map/multimap

# Sequence Containers

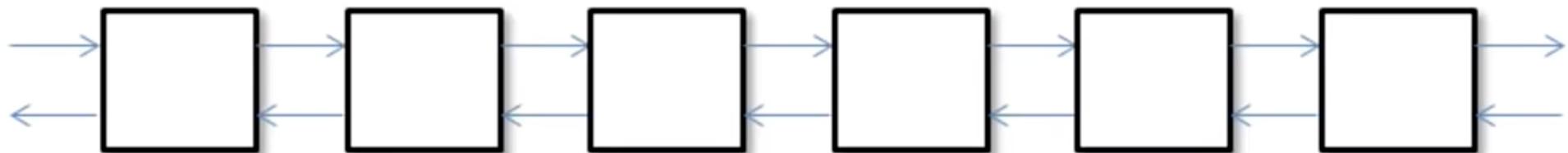
vector



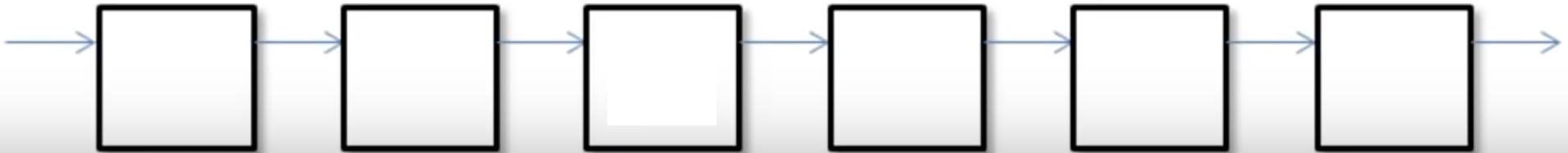
deque



list

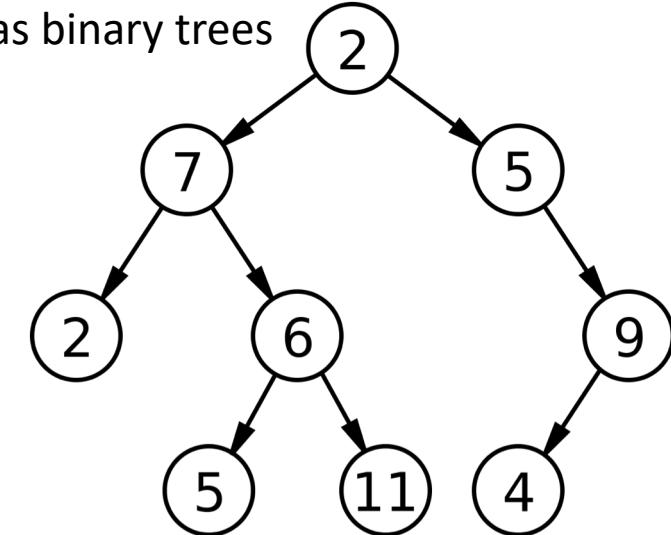


forward  
list

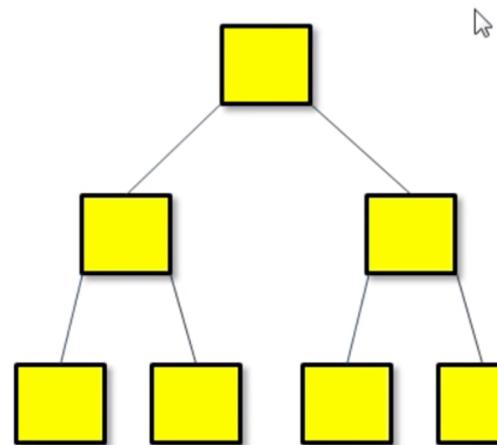


# Associative Containers

stored as binary trees



Set or Multiset:



Map or Multimap:

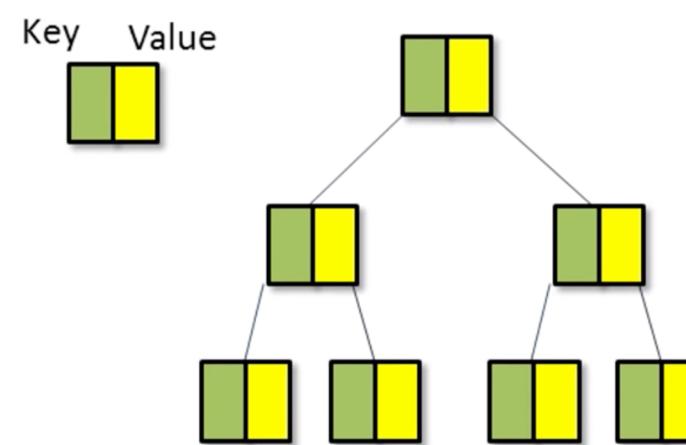


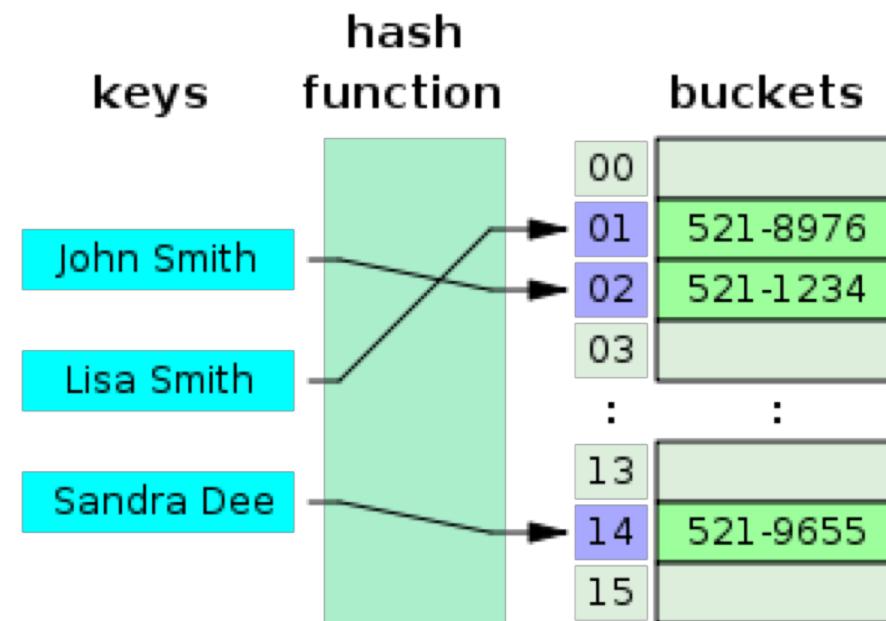
Image taken from Bo Qian's YouTube "Introduction of STL 3: Associative Containers"

The underlying data structure is a balanced search tree:

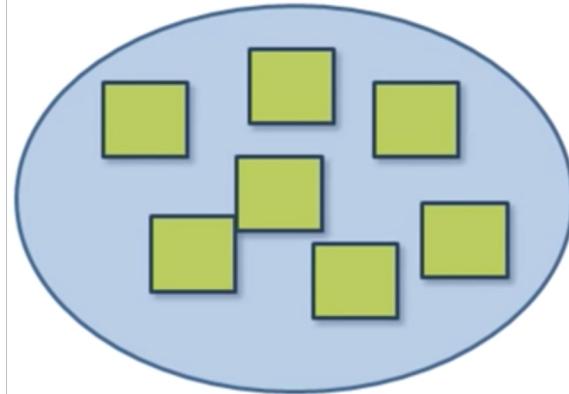
- logarithmic access time
- requires order comparisons of keys
- iteration in key order
- Iterators, pointers and references stay valid until the pointed to element is removed.

# Unordered Containers

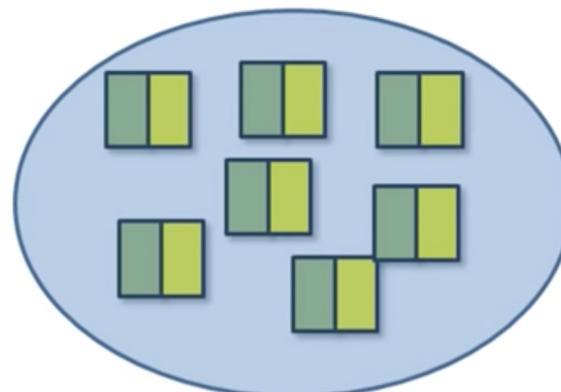
Stored using hash tables:



Unordered  
Set or Multiset:



Unordered  
Map or Multimap:



search, insert and delete in constant time

# Containers have different benefits and costs

	<b>Array</b>	<b>Vector</b>	<b>Deque</b>	<b>List</b>	<b>Forward List</b>	<b>Associative Containers</b>	<b>Unordered Containers</b>
Available since	TR1	C++98	C++98	C++98	C++11	C++98	TR1
Typical internal data structure	Static array	Dynamic array	Array of arrays	Doubly linked list	Singly linked list	Binary tree	Hash table
Element type	Value	Value	Value	Value	Value	Set: value Map: key/value	Set: value Map: key/value
Duplicates allowed	Yes	Yes	Yes	Yes	Yes	Only multiset or multimap	Only multiset or multimap
Iterator category	Random access	Random access	Random access	Bidirectional	Forward	Bidirectional (element/key constant)	Forward (element/key constant)
Growing/shrinking	Never	At one end	At both ends	Everywhere	Everywhere	Everywhere	Everywhere
Random access available	Yes	Yes	Yes	No	No	No	Almost
Search/find elements	Slow	Slow	Slow	Very slow	Very slow	Fast	Very fast
Inserting/removing invalidates iterators	—	On reallocation	Always	Never	Never	Never	On rehashing
Inserting/removing references, pointers	—	On reallocation	Always	Never	Never	Never	Never
Allows memory reservation	—	Yes	No	—	—	—	Yes (buckets)
Frees memory for removed elements	—	Only with <code>shrink_to_fit()</code>	Sometimes	Always	Always	Always	Sometimes
Transaction safe (success or no effect)	No	Push/pop at the end	Push/pop at the beginning and the end	All insertions and all erasures	All insertions and all erasures	Single-element insertions and all erasures if comparing doesn't throw	Single-element insertions and all erasures if hashing and comparing don't throw

# Homework

1) Investigate for each STL containers the cost of:

- Resizing
- Inserting an element (at the front, end, random location)
- Delete an element (at the front, end, random location)

2) **Read Chapters 3, 4, and 5 of Programming: Principles and Practice using C++ by Bjarne Stroustrup** (this includes an introduction to the C++ grammar)

3) ADVANCED:

What are container adaptors (such as stack queue and priority\_queue)? Create them and compare them to the traditional containers.

4) SUPER ADVANCED:

Compute (yourself) the overhead of each container: how much more space does the actual container occupy in addition to its contents

# Homework 2

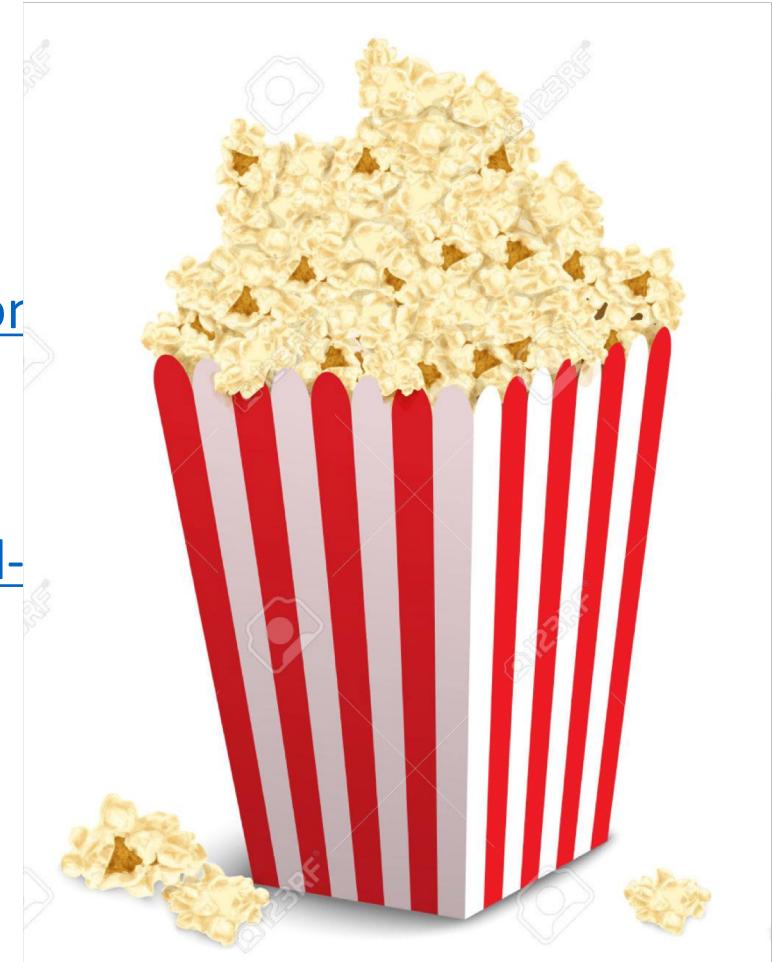
- Take the time to watch these amazing videos (3+ hours well spent!)

<https://channel9.msdn.com/Series/C9-Lectures-Stephan-T-Lavavej-Standard-Template-Library-STL/C9-Lectures-Introduction-to-STL-with-Stephan-T-Lavavej>

<https://channel9.msdn.com/Series/C9-Lectures-Stephan-T-Lavavej-Advanced-STL/C9-Lectures-Stephan-T-Lavavej-Advanced-STL-1-of-n>

And more advanced: Core C++

<https://channel9.msdn.com/Series/C9-Lectures-Stephan-T-Lavavej-Core-C-/Stephan-T-Lavavej-Core-C-1-of-n>



THE #1 PROGRAMMER EXCUSE  
FOR LEGITIMATELY SLACKING OFF:  
"MY CODE'S COMPILING."

