# CSCE240 Final Exam (Stuff and Things)

# The differences between the programming languages C++ and Java

Java doesn’t really compile. C++ goes down to machine code, both java and C++ have similar constructs that are slightly different, java interface is not the same as c++ header file. Java is different for declaring symbols.

**C++**

Was designed for systems and applications programming (a.k.a., infrastructure programming), extending the [procedural programming](https://en.wikipedia.org/wiki/Procedural_programming) language [C](https://en.wikipedia.org/wiki/C_(programming_language)), which was designed for efficient execution. To C, C++ added support for [static typing](https://en.wikipedia.org/wiki/Static_typing), [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), [exception handling](https://en.wikipedia.org/wiki/Exception_handling), lifetime-based resource management ([RAII](https://en.wikipedia.org/wiki/RAII)), [generic programming](https://en.wikipedia.org/wiki/Generic_programming), [template metaprogramming](https://en.wikipedia.org/wiki/Template_metaprogramming), and the [C++ Standard Library](https://en.wikipedia.org/wiki/C%2B%2B_Standard_Library) which includes generic containers and algorithms (STL), and many other general purpose facilities.

**Java**

Is a general-purpose, concurrent, class-based, object-oriented programming language that is designed to minimize implementation dependencies. It relies on a [Java virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine) to be [secure](https://en.wikipedia.org/wiki/Computer_security) and highly [portable](https://en.wikipedia.org/wiki/Porting). It is bundled with an extensive library designed to provide a full abstraction of the underlying platform. Java is a statically typed object-oriented language that uses a syntax similar to C++, but incompatible. It includes a documentation system called [Javadoc](https://en.wikipedia.org/wiki/Javadoc).

The different goals in the development of C++ and Java resulted in different principles and design trade-offs between the languages. The differences are as follows:

|  |  |
| --- | --- |
| **C++** | **Java** |
| Extends [C](https://en.wikipedia.org/wiki/C_(programming_language)) with [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming) and [generic programming](https://en.wikipedia.org/wiki/Generic_programming). C code can most properly be used. | Strongly influenced by C++/C syntax. |
| Compatible with [C](https://en.wikipedia.org/wiki/C_(programming_language)) source code, except for a few [corner cases](https://en.wikipedia.org/wiki/Corner_case). | Provides the [Java Native Interface](https://en.wikipedia.org/wiki/Java_Native_Interface) and recently [Java Native Access](https://en.wikipedia.org/wiki/Java_Native_Access) as a way to directly call C/C++ code. |
| [Write once, compile anywhere](https://en.wikipedia.org/wiki/Write_once,_compile_anywhere) (WOCA). | [Write once, run anywhere](https://en.wikipedia.org/wiki/Write_once,_run_anywhere)/everywhere (WORA/WORE). |
| Allows [procedural programming](https://en.wikipedia.org/wiki/Procedural_programming), [functional programming](https://en.wikipedia.org/wiki/Functional_programming), [object-oriented programming](https://en.wikipedia.org/wiki/Object-oriented_programming), [generic programming](https://en.wikipedia.org/wiki/Generic_programming), and [template metaprogramming](https://en.wikipedia.org/wiki/Template_metaprogramming). Favors a mix of paradigms. | Allows [procedural programming](https://en.wikipedia.org/wiki/Procedural_programming), [functional programming](https://en.wikipedia.org/wiki/Functional_programming) (since Java 8) and [generic programming](https://en.wikipedia.org/wiki/Generic_programming) (since Java 5), but strongly encourages the [object-oriented](https://en.wikipedia.org/wiki/Object-oriented_programming) [programming paradigm](https://en.wikipedia.org/wiki/Programming_paradigm). Includes support for creating [scripting languages](https://en.wikipedia.org/wiki/Scripting_language). |
| Runs as native executable machine code for the target [instruction set](https://en.wikipedia.org/wiki/Instruction_set)(s). | Runs on a [virtual machine](https://en.wikipedia.org/wiki/Java_virtual_machine). |
| Provides object types and type names. Allows reflection via [run-time type information](https://en.wikipedia.org/wiki/Run-time_type_information) (RTTI). | Is [reflective](https://en.wikipedia.org/wiki/Reflection_(computer_programming)), allowing metaprogramming and dynamic code generation at runtime. |
| Has multiple binary compatibility standards (commonly Microsoft (for MSVC compiler) and Itanium/GNU (for almost all other compilers)). | Has one binary compatibility standard, [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) for OS and compiler. |
| Optional automated [bounds checking](https://en.wikipedia.org/wiki/Bounds_checking) (e.g., the at() method in vector and string containers). | All operations are required to be bound-checked by all compliant distributions of Java. [HotSpot](https://en.wikipedia.org/wiki/HotSpot" \o "HotSpot) can remove bounds checking. |
| Native [unsigned arithmetic](https://en.wikipedia.org/wiki/Unsigned_(arithmetic)) support. | Native unsigned arithmetic unsupported. Java 8 changes some of this, but aspects are unclear.[[1]](https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B#cite_note-1) |
| Standardized minimum limits for all numerical types, but the actual sizes are implementation-defined. Standardized types are available via the standard library <cstdint>. | Standardized limits and sizes of all primitive types on all platforms. |
| Pointers, references, and pass-by-value are supported for all types (primitive or user-defined). | All types (primitive types and reference types) are always passed by value.[[2]](https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B#cite_note-2) |
| [Memory management](https://en.wikipedia.org/wiki/Memory_management) can be done [manually](https://en.wikipedia.org/wiki/Manual_memory_management) via new / delete, automatically by scope, or by smart pointers. Supports deterministic destruction of objects. [Garbage collection](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)) ABI standardized in C++11, though compilers are not required to implement garbage collection. | Automatic [garbage collection](https://en.wikipedia.org/wiki/Garbage_collection_(computer_science)). Supports a non-deterministic finalize() method which use is not recommended.[[3]](https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B#cite_note-3) |
| [Resource management](https://en.wikipedia.org/wiki/Resource_management_(computing)) can be done manually or by automatic lifetime-based resource management ([RAII](https://en.wikipedia.org/wiki/RAII)). | Resource management must be done manually, or automatically via finalizers, though this is generally discouraged. Has try-with-resources for automatic scope-based resource management (version 7 onwards). |
| Supports classes, structs ([passive data structure](https://en.wikipedia.org/wiki/Passive_data_structure) (PDS) types), and unions, and can allocate them on the [heap](https://en.wikipedia.org/wiki/Dynamic_memory_allocation) or the [stack](https://en.wikipedia.org/wiki/Stack-based_memory_allocation). | Classes are allocated on the [heap](https://en.wikipedia.org/wiki/Dynamic_memory_allocation). [Java SE 6](https://en.wikipedia.org/wiki/Java_version_history#Java_SE_6_Update_14) optimizes with [escape analysis](https://en.wikipedia.org/wiki/Escape_analysis) to allocate some objects on the [stack](https://en.wikipedia.org/wiki/Stack-based_memory_allocation). |
| Allows explicitly overriding types, and some implicit narrowing conversions (for compatibility with C). | Rigid [type safety](https://en.wikipedia.org/wiki/Type_safety) except for widening conversions. |
| The [C++ Standard Library](https://en.wikipedia.org/wiki/C%2B%2B_Standard_Library) was designed to have a limited scope and functions, but includes language support, diagnostics, general utilities, strings, locales, containers, algorithms, [iterators](https://en.wikipedia.org/wiki/Iterator#C.2B.2B), numerics, input/output, random number generators, regular expression parsing, threading facilities, type traits (for static type introspection) and Standard C Library. The [Boost library](https://en.wikipedia.org/wiki/Boost_(C%2B%2B_libraries)) offers more functions including network I/O.  A rich amount of third-party libraries exist for GUI and other functions like: [Adaptive Communication Environment](https://en.wikipedia.org/wiki/Adaptive_Communication_Environment) (ACE), [Crypto++](https://en.wikipedia.org/wiki/Crypto%2B%2B), various [XMPP](https://en.wikipedia.org/wiki/XMPP) [Instant Messaging](https://en.wikipedia.org/wiki/Instant_Messaging) (IM) libraries,[[4]](https://en.wikipedia.org/wiki/Comparison_of_Java_and_C%2B%2B#cite_note-XMPP_Software_.C2.BB_Libraries-4) [OpenLDAP](https://en.wikipedia.org/wiki/OpenLDAP" \o "OpenLDAP), [Qt](https://en.wikipedia.org/wiki/Qt_(software)" \o "Qt (software)), [gtkmm](https://en.wikipedia.org/wiki/Gtkmm" \o "Gtkmm). | The standard library has grown with each release. By version 1.6, the library included support for locales, logging, containers and iterators, algorithms, GUI programming (but not using the system GUI), graphics, multi-threading, networking, platform security, introspection, dynamic class loading, blocking and non-blocking I/O. It provided interfaces or support classes for [XML](https://en.wikipedia.org/wiki/XML), [XSLT](https://en.wikipedia.org/wiki/XSLT), [MIDI](https://en.wikipedia.org/wiki/MIDI), database connectivity, naming services (e.g. [LDAP](https://en.wikipedia.org/wiki/LDAP)), cryptography, security services (e.g. [Kerberos](https://en.wikipedia.org/wiki/Kerberos_(protocol))), print services, and web services. SWT offers an abstraction for platform-specific GUIs. |
| [Operator overloading](https://en.wikipedia.org/wiki/Operator_overloading) for most operators. Preserving meaning (semantics) is highly recommended. | Operators are not overridable. The language overrides + and += for the String class. |
| Single and [Multiple inheritance](https://en.wikipedia.org/wiki/Multiple_inheritance) of classes, including virtual inheritance. | Single inheritance of classes. Supports multiple inheritance via the [Interfaces](https://en.wikipedia.org/wiki/Interface_(Java)) construct, which is equivalent to a C++ class composed of abstract methods. |
| Compile-time templates. Allows for [Turing complete](https://en.wikipedia.org/wiki/Turing_complete) meta-programming. | [Generics](https://en.wikipedia.org/wiki/Generics_in_Java) are used to achieve basic type-parametrization, but they do not translate from source code to byte code due to the use of [type erasure](https://en.wikipedia.org/wiki/Type_erasure) by the compiler. |
| Function pointers, function objects, lambdas (in [C++11](https://en.wikipedia.org/wiki/C%2B%2B11)), and interfaces. | References to functions achieved via the [reflection](https://en.wikipedia.org/wiki/Reflection_(computer_science)) API. OOP idioms using Interfaces, such as Adapter, Observer, and Listener are generally preferred over direct references to methods. |
| No standard inline documentation mechanism. Third-party software (e.g. [Doxygen](https://en.wikipedia.org/wiki/Doxygen" \o "Doxygen)) exists. | Extensive [Javadoc](https://en.wikipedia.org/wiki/Javadoc) documentation standard on all system classes and methods. |
| const keyword for defining immutable variables and member functions that do not change the object. Const-ness is propagated as a means to enforce, at compile-time, correctness of the code with respect to mutability of objects (see [const-correctness](https://en.wikipedia.org/wiki/Const-correctness" \o "Const-correctness)). | final provides a version of const, equivalent to type\* const pointers for objects and const for primitive types. Immutability of object members achieved via read-only interfaces and object encapsulation. |
| Supports the [goto](https://en.wikipedia.org/wiki/Goto" \o "Goto) statement. | Supports labels with loops and statement blocks. |
| Source code can be written to be [cross-platform](https://en.wikipedia.org/wiki/Cross-platform) (can be compiled for [Windows](https://en.wikipedia.org/wiki/Microsoft_Windows), [BSD](https://en.wikipedia.org/wiki/BSD), [Linux](https://en.wikipedia.org/wiki/Linux), [OS X](https://en.wikipedia.org/wiki/OS_X), [Solaris](https://en.wikipedia.org/wiki/Solaris_(operating_system)), etc., without modification) and written to use platform-specific features. Typically compiled into native machine code, must be recompiled for each target platform. | Compiled into byte code for the [JVM](https://en.wikipedia.org/wiki/JVM). Byte code is dependent on the Java platform, but is typically independent of [operating system](https://en.wikipedia.org/wiki/Operating_system) specific features. |

# White-box testing versus black-box testing

Two important techniques for functionality testing are white-box testing and black-box testing.

White-box testing, sometimes called structural testing or internal testing, focuses on the text of the program. The tester constructs a test suite (a collection of inputs and corresponding expected outputs) that demonstrates that all branches of the program’s choice and loop constructs — if, while, switch, try-catch-finally, and so on — can be executed. The test suite is said to cover the statements of the program.

Black-box testing, sometimes called external testing, focuses on the problem that the program is supposed to solve; or more precisely, the problem statement or specification for the 3 program. The tester constructs a test data set (inputs and corresponding expected outputs) that includes ‘typical’ as well as ‘extreme’ input data. In particular, one must include inputs that are described as exceptional or erroneous in the problem description.

White-box testing and black-box testing are complementary approaches to test case generation. White-box testing does not focus on the problem area, and therefore may not discover that some subproblem is left unsolved by the program, whereas black-box testing should. Black-box testing does not focus on the program text, and therefore may not discover that some parts of the program are completely useless or have an illogical structure, whereas white-box testing should.

Software testing can never prove that a program contains no errors, but it can strengthen one’s faith in the program. Systematic software testing is necessary if the program will be used by others, if the welfare of humans or animals depends on it (so-called safety-critical software), or if one wants to base scientific conclusions on the program’s results.

# std::vector

template < class T, class Alloc = allocator<T> > class vector; // generic template

Allows direct access to any element in the sequence, even through pointer arithmetics, and provides relatively fast addition/removal of elements at the end of the sequence.

|  |  |  |
| --- | --- | --- |
| **member type** | **definition** | **notes** |
| value\_type | The first template parameter (T) |  |
| allocator\_type | The second template parameter (Alloc) | defaults to: [allocator](http://www.cplusplus.com/allocator)<value\_type> |
| reference | allocator\_type::reference | for the default [allocator](http://www.cplusplus.com/allocator): value\_type& |
| const\_reference | allocator\_type::const\_reference | for the default [allocator](http://www.cplusplus.com/allocator): const value\_type& |
| pointer | allocator\_type::pointer | for the default [allocator](http://www.cplusplus.com/allocator): value\_type\* |
| const\_pointer | allocator\_type::const\_pointer | for the default [allocator](http://www.cplusplus.com/allocator): const value\_type\* |
| iterator | a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) to value\_type | convertible to const\_iterator |
| const\_iterator | a [random access iterator](http://www.cplusplus.com/RandomAccessIterator) to const value\_type |  |
| reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<iterator> |  |
| const\_reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<const\_iterator> |  |
| difference\_type | a signed integral type, identical to: iterator\_traits<iterator>::difference\_type | usually the same as [ptrdiff\_t](http://www.cplusplus.com/ptrdiff_t) |
| size\_type | an unsigned integral type that can represent any non-negative value of difference\_type | usually the same as [size\_t](http://www.cplusplus.com/size_t) |

**Member functions**

[**(constructor)**](http://www.cplusplus.com/reference/vector/vector/vector/)

Construct vector (public member function )

[**(destructor)**](http://www.cplusplus.com/reference/vector/vector/~vector/)

Vector destructor (public member function )

[**operator=**](http://www.cplusplus.com/reference/vector/vector/operator=/)

Assign content (public member function )

**Iterators**:

[**begin**](http://www.cplusplus.com/reference/vector/vector/begin/)

Return iterator to beginning (public member function )

[**end**](http://www.cplusplus.com/reference/vector/vector/end/)

Return iterator to end (public member function )

[**rbegin**](http://www.cplusplus.com/reference/vector/vector/rbegin/)

Return reverse iterator to reverse beginning (public member function )

[**rend**](http://www.cplusplus.com/reference/vector/vector/rend/)

Return reverse iterator to reverse end (public member function )

[**cbegin**](http://www.cplusplus.com/reference/vector/vector/cbegin/)

Return const\_iterator to beginning (public member function )

[**cend**](http://www.cplusplus.com/reference/vector/vector/cend/)

Return const\_iterator to end (public member function )

[**crbegin**](http://www.cplusplus.com/reference/vector/vector/crbegin/)

Return const\_reverse\_iterator to reverse beginning (public member function )

[**crend**](http://www.cplusplus.com/reference/vector/vector/crend/)

Return const\_reverse\_iterator to reverse end (public member function )

**Capacity**:

[**size**](http://www.cplusplus.com/reference/vector/vector/size/)

Return size (public member function )

[**max\_size**](http://www.cplusplus.com/reference/vector/vector/max_size/)

Return maximum size (public member function )

[**resize**](http://www.cplusplus.com/reference/vector/vector/resize/)

Change size (public member function )

[**capacity**](http://www.cplusplus.com/reference/vector/vector/capacity/)

Return size of allocated storage capacity (public member function )

[**empty**](http://www.cplusplus.com/reference/vector/vector/empty/)

Test whether vector is empty (public member function )

[**reserve**](http://www.cplusplus.com/reference/vector/vector/reserve/)

Request a change in capacity (public member function )

[**shrink\_to\_fit**](http://www.cplusplus.com/reference/vector/vector/shrink_to_fit/)

Shrink to fit (public member function )

**Element access**:

[**operator[]**](http://www.cplusplus.com/reference/vector/vector/operator%5b%5d/)

Access element (public member function )

[**at**](http://www.cplusplus.com/reference/vector/vector/at/)

Access element (public member function )

[**front**](http://www.cplusplus.com/reference/vector/vector/front/)

Access first element (public member function )

[**back**](http://www.cplusplus.com/reference/vector/vector/back/)

Access last element (public member function )

[**data**](http://www.cplusplus.com/reference/vector/vector/data/)

Access data (public member function )

**Modifiers**:

[**assign**](http://www.cplusplus.com/reference/vector/vector/assign/)

Assign vector content (public member function )

[**push\_back**](http://www.cplusplus.com/reference/vector/vector/push_back/)

Add element at the end (public member function )

[**pop\_back**](http://www.cplusplus.com/reference/vector/vector/pop_back/)

Delete last element (public member function )

[**insert**](http://www.cplusplus.com/reference/vector/vector/insert/)

Insert elements (public member function )

[**erase**](http://www.cplusplus.com/reference/vector/vector/erase/)

Erase elements (public member function )

[**swap**](http://www.cplusplus.com/reference/vector/vector/swap/)

Swap content (public member function )

[**clear**](http://www.cplusplus.com/reference/vector/vector/clear/)

Clear content (public member function )

[**emplace**](http://www.cplusplus.com/reference/vector/vector/emplace/)

Construct and insert element (public member function )

[**emplace\_back**](http://www.cplusplus.com/reference/vector/vector/emplace_back/)

Construct and insert element at the end (public member function )

**How to iterate over a vector of double with an iterator?**

**Vector<double>::const\_iterator iter;**

**String return\_value;**

**For(iter = the\_data\_.begin(); iter != the\_data\_.end(); iter++)**

**{**

**Return\_value = return\_value + (\*iter.ToString()) + “\n”;**

**}**

# std::set

template < class T, // set::key\_type/value\_type

class Compare = less<T>, // set::key\_compare/value\_compare

class Alloc = allocator<T> // set::allocator\_type

> class set;

**Set**

* Sets are containers that store unique elements following a specific order.

|  |  |  |
| --- | --- | --- |
| **member type** | **definition** | **notes** |
| key\_type | The first template parameter (T) |  |
| value\_type | The first template parameter (T) |  |
| key\_compare | The second template parameter (Compare) | defaults to: [less](http://www.cplusplus.com/less)<key\_type> |
| value\_compare | The second template parameter (Compare) | defaults to: [less](http://www.cplusplus.com/less)<value\_type> |
| allocator\_type | The third template parameter (Alloc) | defaults to: [allocator](http://www.cplusplus.com/allocator)<value\_type> |
| reference | allocator\_type::reference | for the default [allocator](http://www.cplusplus.com/allocator): value\_type& |
| const\_reference | allocator\_type::const\_reference | for the default [allocator](http://www.cplusplus.com/allocator): const value\_type& |
| pointer | allocator\_type::pointer | for the default [allocator](http://www.cplusplus.com/allocator): value\_type\* |
| const\_pointer | allocator\_type::const\_pointer | for the default [allocator](http://www.cplusplus.com/allocator): const value\_type\* |
| iterator | a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) to value\_type | convertible to const\_iterator |
| const\_iterator | a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) to const value\_type |  |
| reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<iterator> |  |
| const\_reverse\_iterator | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<const\_iterator> |  |
| difference\_type | a signed integral type, identical to: iterator\_traits<iterator>::difference\_type | usually the same as [ptrdiff\_t](http://www.cplusplus.com/ptrdiff_t) |
| size\_type | an unsigned integral type that can represent any non-negative value of difference\_type | usually the same as [size\_t](http://www.cplusplus.com/size_t) |

**Member functions**

[**(constructor)**](http://www.cplusplus.com/reference/set/set/set/)

Construct set (public member function )

[**(destructor)**](http://www.cplusplus.com/reference/set/set/~set/)

Set destructor (public member function )

[**operator=**](http://www.cplusplus.com/reference/set/set/operator=/)

Copy container content (public member function )

**Iterators**:

[**begin**](http://www.cplusplus.com/reference/set/set/begin/)

Return iterator to beginning (public member function )

[**end**](http://www.cplusplus.com/reference/set/set/end/)

Return iterator to end (public member function )

[**rbegin**](http://www.cplusplus.com/reference/set/set/rbegin/)

Return reverse iterator to reverse beginning (public member function )

[**rend**](http://www.cplusplus.com/reference/set/set/rend/)

Return reverse iterator to reverse end (public member function )

[**cbegin**](http://www.cplusplus.com/reference/set/set/cbegin/)

Return const\_iterator to beginning (public member function )

[**cend**](http://www.cplusplus.com/reference/set/set/cend/)

Return const\_iterator to end (public member function )

[**crbegin**](http://www.cplusplus.com/reference/set/set/crbegin/)

Return const\_reverse\_iterator to reverse beginning (public member function )

[**crend**](http://www.cplusplus.com/reference/set/set/crend/)

Return const\_reverse\_iterator to reverse end (public member function )

**Capacity**:

[**empty**](http://www.cplusplus.com/reference/set/set/empty/)

Test whether container is empty (public member function )

[**size**](http://www.cplusplus.com/reference/set/set/size/)

Return container size (public member function )

[**max\_size**](http://www.cplusplus.com/reference/set/set/max_size/)

Return maximum size (public member function )

**Modifiers**:

[**insert**](http://www.cplusplus.com/reference/set/set/insert/)

Insert element (public member function )

[**erase**](http://www.cplusplus.com/reference/set/set/erase/)

Erase elements (public member function )

[**swap**](http://www.cplusplus.com/reference/set/set/swap/)

Swap content (public member function )

[**clear**](http://www.cplusplus.com/reference/set/set/clear/)

Clear content (public member function )

[**emplace**](http://www.cplusplus.com/reference/set/set/emplace/)

Construct and insert element (public member function )

[**emplace\_hint**](http://www.cplusplus.com/reference/set/set/emplace_hint/)

Construct and insert element with hint (public member function )

**Observers**:

[**key\_comp**](http://www.cplusplus.com/reference/set/set/key_comp/)

Return comparison object (public member function )

[**value\_comp**](http://www.cplusplus.com/reference/set/set/value_comp/)

Return comparison object (public member function )

**Operations**:

[**find**](http://www.cplusplus.com/reference/set/set/find/)

Get iterator to element (public member function )

[**count**](http://www.cplusplus.com/reference/set/set/count/)

Count elements with a specific value (public member function )

[**lower\_bound**](http://www.cplusplus.com/reference/set/set/lower_bound/)

Return iterator to lower bound (public member function )

[**upper\_bound**](http://www.cplusplus.com/reference/set/set/upper_bound/)

Return iterator to upper bound (public member function )

[**equal\_range**](http://www.cplusplus.com/reference/set/set/equal_range/)

Get range of equal elements (public member function )

# std::map

template < class Key, // map::key\_type

class T, // map::mapped\_type

class Compare = less<Key>, // map::key\_compare

class Alloc = allocator<pair<const Key,T> > // map::allocator\_type

> class map;

**Map**

Maps are associative containers that store elements formed by a combination of a *key value* and a *mapped value*, following a specific order.  
  
In a map, the *key values* are generally used to sort and uniquely identify the elements, while the *mapped values* store the content associated to this *key*. The types of *key* and *mapped value* may differ, and are grouped together in member type value\_type, which is a [pair](http://www.cplusplus.com/pair) type combining both:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | typedef pair<const Key, T> value\_type; | |  |
|  | |  | |  |
| **member type** | | **definition** | | | **notes** |
| key\_type | | The first template parameter (Key) | | |  |
| mapped\_type | | The second template parameter (T) | | |  |
| value\_type | | [pair](http://www.cplusplus.com/pair)<const key\_type,mapped\_type> | | |  |
| key\_compare | | The third template parameter (Compare) | | | defaults to: [less](http://www.cplusplus.com/less)<key\_type> |
| value\_compare | | *Nested function class to compare elements* | | | see [value\_comp](http://www.cplusplus.com/map::value_comp) |
| allocator\_type | | The fourth template parameter (Alloc) | | | defaults to: [allocator](http://www.cplusplus.com/allocator)<value\_type> |
| reference | | allocator\_type::reference | | | for the default [allocator](http://www.cplusplus.com/allocator): value\_type& |
| const\_reference | | allocator\_type::const\_reference | | | for the default [allocator](http://www.cplusplus.com/allocator): const value\_type& |
| pointer | | allocator\_type::pointer | | | for the default [allocator](http://www.cplusplus.com/allocator): value\_type\* |
| const\_pointer | | allocator\_type::const\_pointer | | | for the default [allocator](http://www.cplusplus.com/allocator): const value\_type\* |
| iterator | | a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) to value\_type | | | convertible to const\_iterator |
| const\_iterator | | a [bidirectional iterator](http://www.cplusplus.com/BidirectionalIterator) to const value\_type | | |  |
| reverse\_iterator | | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<iterator> | | |  |
| const\_reverse\_iterator | | [reverse\_iterator](http://www.cplusplus.com/reverse_iterator)<const\_iterator> | | |  |
| difference\_type | | a signed integral type, identical to: iterator\_traits<iterator>::difference\_type | | | usually the same as [ptrdiff\_t](http://www.cplusplus.com/ptrdiff_t) |
| size\_type | | an unsigned integral type that can represent any non-negative value of difference\_type | | | usually the same as [size\_t](http://www.cplusplus.com/size_t) |

**Member functions**

[**(constructor)**](http://www.cplusplus.com/reference/map/map/map/)

Construct map (public member function )

[**(destructor)**](http://www.cplusplus.com/reference/map/map/~map/)

Map destructor (public member function )

[**operator=**](http://www.cplusplus.com/reference/map/map/operator=/)

Copy container content (public member function )

**Iterators**:

[**begin**](http://www.cplusplus.com/reference/map/map/begin/)

Return iterator to beginning (public member function )

[**end**](http://www.cplusplus.com/reference/map/map/end/)

Return iterator to end (public member function )

[**rbegin**](http://www.cplusplus.com/reference/map/map/rbegin/)

Return reverse iterator to reverse beginning (public member function )

[**rend**](http://www.cplusplus.com/reference/map/map/rend/)

Return reverse iterator to reverse end (public member function )

[**cbegin**](http://www.cplusplus.com/reference/map/map/cbegin/)

Return const\_iterator to beginning (public member function )

[**cend**](http://www.cplusplus.com/reference/map/map/cend/)

Return const\_iterator to end (public member function )

[**crbegin**](http://www.cplusplus.com/reference/map/map/crbegin/)

Return const\_reverse\_iterator to reverse beginning (public member function )

[**crend**](http://www.cplusplus.com/reference/map/map/crend/)

Return const\_reverse\_iterator to reverse end (public member function )

**Capacity**:

[**empty**](http://www.cplusplus.com/reference/map/map/empty/)

Test whether container is empty (public member function )

[**size**](http://www.cplusplus.com/reference/map/map/size/)

Return container size (public member function )

[**max\_size**](http://www.cplusplus.com/reference/map/map/max_size/)

Return maximum size (public member function )

**Element access**:

[**operator[]**](http://www.cplusplus.com/reference/map/map/operator%5b%5d/)

Access element (public member function )

[**at**](http://www.cplusplus.com/reference/map/map/at/)

Access element (public member function )

**Modifiers**:

[**insert**](http://www.cplusplus.com/reference/map/map/insert/)

Insert elements (public member function )

[**erase**](http://www.cplusplus.com/reference/map/map/erase/)

Erase elements (public member function )

[**swap**](http://www.cplusplus.com/reference/map/map/swap/)

Swap content (public member function )

[**clear**](http://www.cplusplus.com/reference/map/map/clear/)

Clear content (public member function )

[**emplace**](http://www.cplusplus.com/reference/map/map/emplace/)

Construct and insert element (public member function )

[**emplace\_hint**](http://www.cplusplus.com/reference/map/map/emplace_hint/)

Construct and insert element with hint (public member function )

**Observers**:

[**key\_comp**](http://www.cplusplus.com/reference/map/map/key_comp/)

Return key comparison object (public member function )

[**value\_comp**](http://www.cplusplus.com/reference/map/map/value_comp/)

Return value comparison object (public member function )

**Operations**:

[**find**](http://www.cplusplus.com/reference/map/map/find/)

Get iterator to element (public member function )

[**count**](http://www.cplusplus.com/reference/map/map/count/)

Count elements with a specific key (public member function )

[**lower\_bound**](http://www.cplusplus.com/reference/map/map/lower_bound/)

Return iterator to lower bound (public member function )

[**upper\_bound**](http://www.cplusplus.com/reference/map/map/upper_bound/)

Return iterator to upper bound (public member function )

[**equal\_range**](http://www.cplusplus.com/reference/map/map/equal_range/)

Get range of equal elements (public member function )

|  |  |  |  |
| --- | --- | --- | --- |
|  | Uses of CONST  |  |  | | --- | --- | |  | **Binding some temporary to reference-to-const, to lengthen its lifetime.** The reference can be a base - and the destructor of it doesn't need to be virtual - the right destructor is still called:  ScopeGuard const& guard = MakeGuard(&cleanUpFunction);  *Explanation*, using code:  struct ScopeGuard {  ~ScopeGuard() { } // not virtual  };  template<typename T> struct Derived : ScopeGuard {  T t;  Derived(T t):t(t) { }  ~Derived() {  t(); // call function  }  };  template<typename T> Derived<T> MakeGuard(T t) { return Derived<T>(t); }  This trick is used in Alexandrescu's ScopeGuard utility class. Once the temporary goes out of scope, the destructor of Derived is called correctly. The above code misses some small details, but that's the big deal with it.  **Use const to tell others methods won't change the logical state of this object.**  struct SmartPtr {  int getCopies() const { return mCopiesMade; }  };  **Use const for copy-on-write classes**, to make the compiler help you to decide when and when not you need to copy.  struct MyString {  char \* getData() { /\* copy: caller might write \*/ return mData; }  char const\* getData() const { return mData; }  };  *Explanation*: You might want to share data when you copy something as long as the data of the originally and the copie'd object remain the same. Once one of the object changes data, you however need now two versions: One for the original, and one for the copy. That is, you *copy* on a *write* to either object, so that they now both have their own version.  *Using code*:  int main() {  string const a = "1234";  string const b = a;  // outputs the same address for COW strings  cout << (void\*)&a[0] << ", " << (void\*)&b[0];  }  The above snippet prints the same address on my GCC, because the used C++ library implements a copy-on-write std::string. Both strings, even though they are distinct objects, share the same memory for their string data. Making b non-const will prefer the non-const version of the operator[]and GCC will create a copy of the backing memory buffer, because we could change it and it must not affect the data of a!  int main() {  string const a = "1234";  string b = a;  // outputs different addresses!  cout << (void\*)&a[0] << ", " << (void\*)&b[0];  }  **For the copy-constructor to make copies from const objects and temporaries**:  struct MyClass {  MyClass(MyClass const& that) { /\* make copy of that \*/ }  };  **For making constants that trivially can't change**  double const PI = 3.1415;  **For passing arbitrary objects by reference instead of by value** - to prevent possibly expensive or impossible by-value passing  void PrintIt(Object const& obj) {  // ...  } |   **Binding some temporary to reference-to-const, to lengthen its lifetime.** The reference can be a base - and the destructor of it doesn't need to be virtual - the right destructor is still called:  ScopeGuard const& guard = MakeGuard(&cleanUpFunction);  *Explanation*, using code:  struct ScopeGuard {  ~ScopeGuard() { } // not virtual  };  template<typename T> struct Derived : ScopeGuard {  T t;  Derived(T t):t(t) { }  ~Derived() {  t(); // call function  }  };  template<typename T> Derived<T> MakeGuard(T t) { return Derived<T>(t); }  This trick is used in Alexandrescu's ScopeGuard utility class. Once the temporary goes out of scope, the destructor of Derived is called correctly. The above code misses some small details, but that's the big deal with it.  **Use const to tell others methods won't change the logical state of this object.**  struct SmartPtr {  int getCopies() const { return mCopiesMade; }  };  **Use const for copy-on-write classes**, to make the compiler help you to decide when and when not you need to copy.  struct MyString {  char \* getData() { /\* copy: caller might write \*/ return mData; }  char const\* getData() const { return mData; }  };  *Explanation*: You might want to share data when you copy something as long as the data of the originally and the copie'd object remain the same. Once one of the object changes data, you however need now two versions: One for the original, and one for the copy. That is, you *copy* on a *write* to either object, so that they now both have their own version.  *Using code*:  int main() {  string const a = "1234";  string const b = a;  // outputs the same address for COW strings  cout << (void\*)&a[0] << ", " << (void\*)&b[0];  }  The above snippet prints the same address on my GCC, because the used C++ library implements a copy-on-write std::string. Both strings, even though they are distinct objects, share the same memory for their string data. Making b non-const will prefer the non-const version of the operator[]and GCC will create a copy of the backing memory buffer, because we could change it and it must not affect the data of a!  int main() {  string const a = "1234";  string b = a;  // outputs different addresses!  cout << (void\*)&a[0] << ", " << (void\*)&b[0];  }  **For the copy-constructor to make copies from const objects and temporaries**:  struct MyClass {  MyClass(MyClass const& that) { /\* make copy of that \*/ }  };  **For making constants that trivially can't change**  double const PI = 3.1415;  **For passing arbitrary objects by reference instead of by value** - to prevent possibly expensive or impossible by-value passing  void PrintIt(Object const& obj) {  // ...  } |

# Dr. Gay’s Slides

**Software Requirements**

Requirement is a high level statement that is a singular documented physical or functional need that a particular product must be able to perform. A specification is a comprehensive technical description of how that requirement will be realized. Requirements and spec. are a description of what the system show do, now how. A requirement should be correct, precise, unambiguous, clear, and able to be tested.

A use case captures some visible function of the system. It is a goal that an actor can accomplish using the system. I.e. transfer funds. Query balance. Etc. A use case description should include all scenarios that can occur when attempting to achieve that use case.

**Software Testing**

Input, predicted output, procedure needed to exercise the system. A Test Plan features : what is being tested, when it will be tested, how it will be tested, where we are testing it, why are we testing it, and who is responsible for writing test cases.

**Functional Testing**

Requirements based testing is the process of deriving tests from the requirement specs. All tests should be independently testable. i.e. don’t test for multiple things at once. Group test info into equivalent partitions.