Fun with C++11

```
#define fun
```
or,
```
#define fun true
```

Disclaimers

- I like a lot of languages in addition to C++
  - C# (mostly); Python; Java; PHP; Groovy; would like to learn Ruby and Clojure, and someday: LISP
- I use Microsoft C++ mostly.
- I'm not a C++ expert. I find even *these* challenging: http://cppquiz.org/quiz/question/1
- I practice EDD

EDD: **Error Driven Development**

- You can learn a LOT from the compiler warnings and errors. Fun!

- Template errors and warnings are getting better. (Have been notoriously bad.)

reddit   CPP   **comments**   related

Optional Types and Lightweight Continuation Passing in C++   (treswalsh.com)

18   submitted 23 hours ago by SAHChandler

6 comments   share

all 6 comments

sorted by: **best** ▼

[−] **dbchfjdksisjxb**   4 points 6 hours ago

C++ is getting weird

permalink

[−] **edric_garran**   0 points 6 hours ago

weird is good

permalink   parent

← → C ⌂   🗋 www.reddit.com/r/cpp/comments/2mrzla/what_is_stddecay_and_when_it_should_be_used/

MY SUBREDDITS ▼   FRONT - ALL - RANDOM | FOOD - FUTUROLOGY - MOVIES - NEWS - IAMA - SPACE - DOCUMENTARIES - BOOKS - HISTORY - WORLDNEWS - ASKRE

reddit CPP **comments** related

▲ What is std::decay and when it should be used? (stackoverflow.com)
10 submitted 2 hours ago by ericjaviersaura
▼ 3 comments share

all 3 comments

sorted by: best ▼

▲ [–] **TomSwirly** 2 points 2 hours ago
▼ I'm split about this sort of thing. It makes perfect sense if you understand C++, and yet when I think about
explaining it to friends who program in other languages, I realize that the necessity of such abstruse constructs is
one of the downsides of C++11/14.

permalink

   ▲ [–] **bames53** 4 points an hour ago
   ▼ Consider what you're doing when you're implementing a generic pair type: You're not creating a type that
holds two objects: In effect you're writing a program that knows how to *design* such types. That design
knowledge must get encoded in there somehow. You just have to compare that complexity to the trade offs
made by other languages.

- In C we can't represent the abstraction at all and instead just write each kind of pair manually, which
produces efficient results.
- In Java we can have the abstraction, but the pair types it produces won't be efficient.

And finally it's important to note that writing this kind of C++ isn't required. It's available if and when you
need it, but it's perfectly reasonable to write C++ without getting into this complexity. New C++
programmers can be productive writing C++ for years without learning this stuff.

So I think C++ makes a fairly good compromise: simple, efficient, good abstractions; pick any two.
Languages like C and Java, on the other hand, have already picked for you.

▲ Ask HN: How to refresh on modern features of C++?

100 points by ctb_mg 6 days ago | 62 comments

I first learned C++ 14 years ago and have only used it in a non-trivial fashion on one or two short projects since then.

Today, reading an article on popular C++ myths [1], I now realized there are a TON of things I didn't realize modern C++ can do now. auto, unique_ptr, for_each, and range-for loops were new to me. I'm not 100% certain if all of these were recent additions to the language.

What are some things I can do to brush up on the modern features of C++?

https://news.ycombinator.com/item?id=8788454

https://isocpp.org/blog/2014/12/myths-1

What's changed?

- C++11 standard *finally* published
  - Took too long.
  - For a *long time,* temporarily named "C++0x"
  - Finally published - but now we're in **2015**…and **C++14** is about to be finalized
- Mobile devices want fewer CPU cycles; battery consumption: a big deal
- Microsoft: C++ [was] 2nd-class citizen in the .NET era of early 2000's; that changed with 'RT'
- Big internet players *really, really* need performance and massive scale

Fun with C++ 11

"Conclusion: C++ has changed dramatically over the last decade. It's no more C with Classes. If you not looked at C++ recently, it will be the right time to have another look."

http://blog.madhukaraphatak.com/functional-programming-in-c++/

good/bad news

- C++ was big already - and got a lot bigger with C++11
- C++14: much smaller set of changes compared to C++11
- Regardless, pace of change is increasing

C++ History

Year	C++ Standard	Informal name
1998	ISO/IEC 14882:1998[12]	C++98
2003	ISO/IEC 14882:2003[13]	C++03
2007	ISO/IEC TR 19768:2007[14]	C++TR1
2011	ISO/IEC 14882:2011[4]	C++11
2014	N3690 (working draft C++14)[15]	C++14
2017	to be determined	C++17

source: wikipedia C++ article

C++ History: What happened to Technical Report 2 (TR2)?

In 2005, a request for proposals for a TR2 was made with a special interest in Unicode, XML/HTML, Networking and usability for novice programmers.[3]. Some of the proposals included:

- Threads [4]
- The Asio C++ library (networking [5][6]).
- Signals/Slots [7][8]
- Filesystem Library [9] – Based on the Boost Filesystem Library, for query/manipulation of paths, files and directories.
- Boost Any Library [10]
- Lexical Conversion Library [11]
- New String Algorithms [12]
- Toward a More Complete Taxonomy of Algebraic Properties for Numeric Libraries in TR2 [13]
- Adding heterogeneous comparison lookup to associative containers for TR2 [14]

Since the call for proposals for TR2, changes to ISO procedures meant that there will not be a TR2, instead enhancements to C++ will be published in a number of Technical Specifications. Some of the proposals listed above are already included in the C++ standard or in draft versions of the Technical Specifications.

https://en.wikipedia.org/wiki/C%2B%2B_Technical_Report_1#Technical_Report_2

C++ People: Bjarne Stroustrup

- On the ISO standards committee
- NOT a BDFL
- Still writing good books
- Hair: Big fun topic at cppcon.

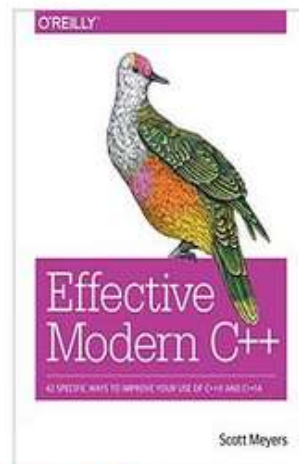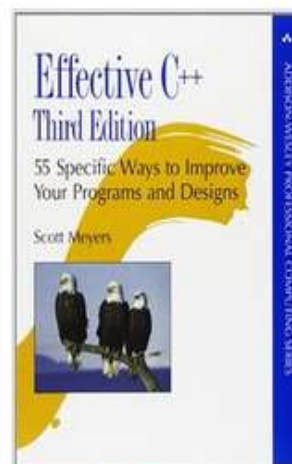http://scottmeyers.blogspot.com/2014/09/cppcon-hair-poll.html

C++ People: Scott Meyers

- Latest book is *hot, hot, hot!*
- Has 'the hair'

Scott Meyers
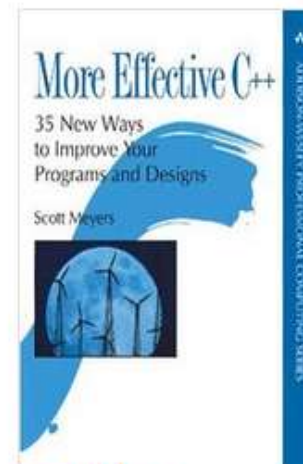
Books by Scott Meyers

Showing 10 Results   Sort by

C++ People

- Andrei Alexandrescu
  - Book: <u>Modern C++ Design: Generic Programming and Design Patterns Applied (2001)</u>
  - Loki [framework](#): "a C++ software library written by Andrei Alexandrescu as part of his book Modern C++ Design. The library makes extensive use of C++ template metaprogramming and implements several commonly used tools: typelist, functor, singleton, smart pointer, object factory, visitor and multimethods."
  - Facebook's 'Flint' – written in 'D': https://code.facebook.com/posts/729709347050548/under-the-hood-building-and-open-sourcing-flint/
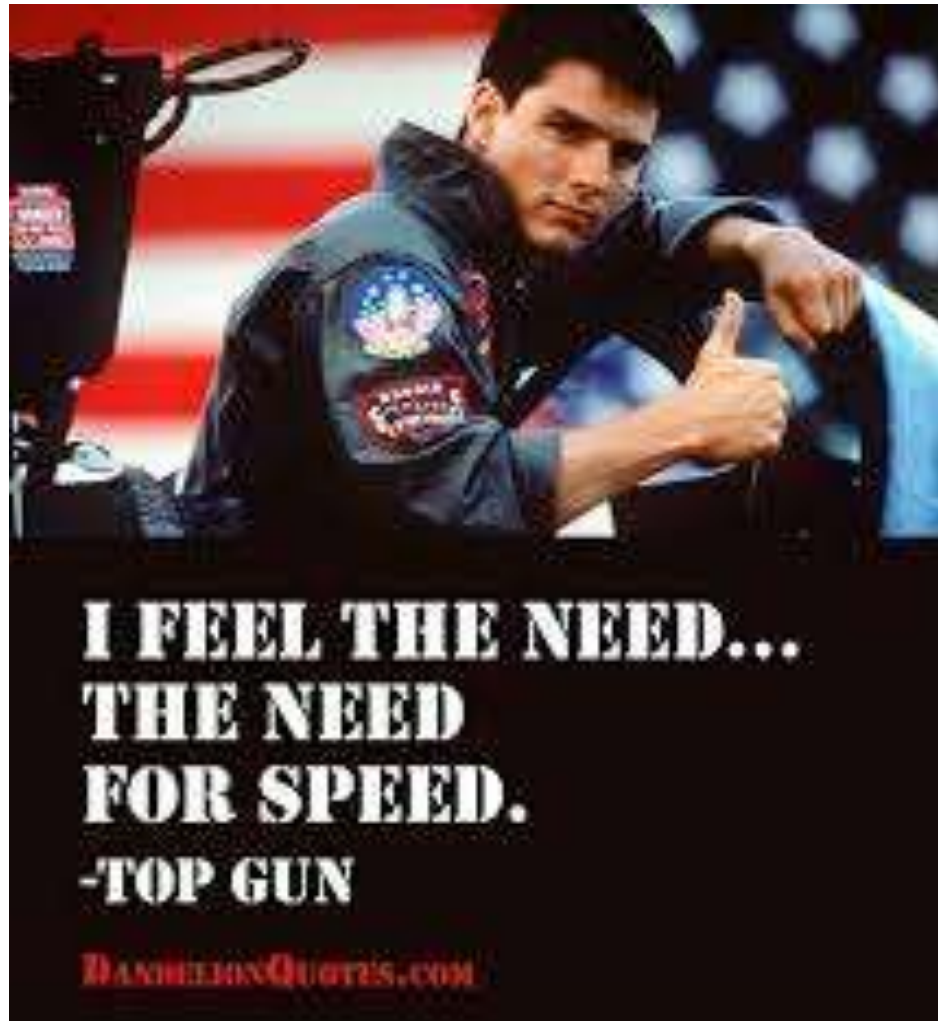
C++ Standard

- ## [ISO/IEC 14881:2011](#)

  ISO/IEC 14882:2011 specifies requirements for implementations of the C++ programming language. The first such requirement is that they implement the language, and so ISO/IEC 14882:2011 also defines C++. Other requirements and relaxations of the first requirement appear at various places within ISO/IEC 14882:2011.

  C++ is a general purpose programming language based on the C programming language as specified in ISO/IEC 9899:1999. In addition to the facilities provided by C, C++ provides additional data types, classes, templates, exceptions, namespaces, operator overloading, function name overloading, references, free store management operators, and additional library facilities.

- 1300+ pages

- isocpp.org: links to free PDF drafts. Fun!

- Purchase it here in the U.S. ($30 or $60?).

Why C++?

Sources of C++ Innovation

- all the other cool programming languages
- C++ Standard Library (a.k.a. STL)
- [LLVM](): all your languages belong to us…and [Clang]()…
- Boost library: Many things make their way into C++ std lib
- Facebook: HipHop transpiler (PHP => C++); many opensource libs
- Google – products and tools. Yes; lots
- Microsoft – Windows; tools; Office…
  - .NET not used in tools or Office…or drivers. ☹

Microsoft; Clang; LLVM

Microsoft – Nov 2014:

"The C++ team has made a goal to achieve C++11 and C++14 conformance in the Visual C++ compiler for Visual Studio 2015's final release. But there's more: Visual Studio 2015 will actually support another, modern conformant C++ compiler – Clang for projects targeting non-Microsoft platforms. In this video, Herb Sutter discusses how you'll be able to write a single cross-platform C++ source base and build it to target Windows, Windows Phone, Android, and soon iOS, all from within Visual Studio!

http://channel9.msdn.com/Events/Visual-Studio/Connect-event-2014/311

Sources of C++ Innovation

- Dropbox: [Cross-platform C++ layer](). iOS, Android, …
- [Catch](): "A **modern** C++-native, header-only, framework for unit-tests, TDD and BDD"
- JetBrains (of ReSharper and IntelliJ IDEA): [CLion IDE]() (and plug-in for VS).
- Google/Android: tools for C/C++
- Intel NDK – [native dev for Android]()
- Intel [Threading Building Blocks]()
- Cevelop: Eclipse-based [C++ IDE]() with unit testing, refactoring

C++: Alive and Well

- Facebook:

  Proxygen makes heavy use of the latest C++ features and depends on **Thrift** and **Folly** for its underlying network and data abstractions. We make use of move semantics to avoid extra copies for large objects like body buffers and header representations while avoiding typical pitfalls like memory leaks. Additionally, by using non-blocking IO and Linux's epoll under the hood, we are able to create a memory and CPU efficient server.

C++: Alive and Well

- Qt Framework: Staying current…

"C++14 for Qt programmers":

http://woboq.com/blog/cpp14-in-qt.html

Yes you Can!

- Develop with C++ on windows, linux, and OSX
- Develop iOS apps
- Develop Android apps
- Develop for cool little microcontrollers
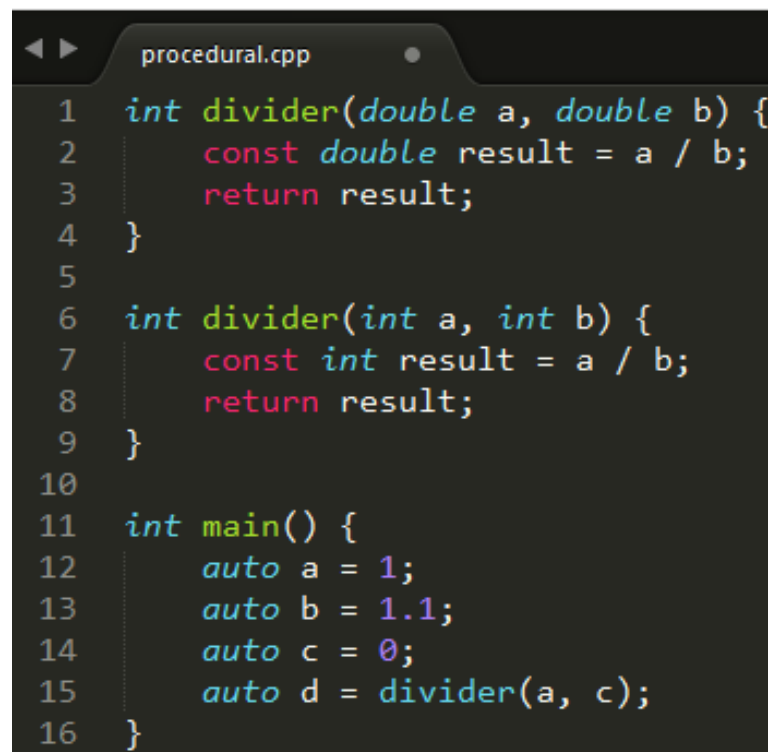 - Aurduino
  - Raspberry Pi
  - Beagle Bone

C++ is Multi-Paradigm

- Procedural

- Object Oriented

- Functional

- Generic

The last two are where the biggest C++ changes are taking place.

Paradigms: Procedural

- statements; expressions; functions; subroutines

```cpp
int divider(double a, double b) {
    const double result = a / b;
    return result;
}

int divider(int a, int b) {
    const int result = a / b;
    return result;
}

int main() {
    auto a = 1;
    auto b = 1.1;
    auto c = 0;
    auto d = divider(a, c);
}
```

Paradigms: Object Oriented

- Classes; objects

```cpp
procedural.cpp
1   #include <iostream>
2
3   class Animal {
4   public:
5       void Breathe() {std::cout << "sigh..." << std::endl;}
6   };
7
8   class Person : public Animal {
9   public:
10      Person(double age) : _age(age) {}
11      Person() = delete;
12  private:
13      double _age;
14  };
15
16  int main() {
17      Person fred(53.4);
18      Person bob; // compile fail
19      fred.Breathe();
20  }
```

Paradigms: Generic

- Templates
  - Templated functions
  - Templated classes

- As C++ developers, we *need* to understand how to read and understand C++ code that uses templates.

- So much of C++11/14 Standard Library leverages Generic Programming / templates.
  - *Read the source, Luke: Open <string> for example…*

We should all be able to read this…

```cpp
#include <iostream>
#include <string>

template<typename T>
T const max_fun_with_cpp_11(T const& a, T const& b)
{
    const auto result = a > b ? a : b;
    return result;
}

int main() {
    std::cout << max_fun_with_cpp_11(7, 29) << std::endl;
    std::cout << max_fun_with_cpp_11(
        std::string("zebra"), std::string("fred"))
        << std::endl;
}
```

C++ Standard Library / STL:
Not Object Oriented!

- Read what the creator of the STL has to say about this:

  https://en.wikipedia.org/wiki/Alexander_Stepanov#Criticism_of_OOP

- He's got a new book, too:

  http://www.informit.com/store/from-mathematics-to-generic-programming-9780321942043

Paradigms: Functional

- Lambdas!
  - In C++11, "Lambda Expressions" is correct terminology
  - We know what C / C++ expressions are; makes it a little easier
- Pure Functions
  - Thread safe

[Gamasutra article](Gamasutra%20article) – John Carmack

Paradigms: Bjarne's recent take

(Copied from: http://www.stroustrup.com/Myths-final.pdf)

Consider an example:

```
void rotate_and_draw(vector<Shape*>& vs, int r) {
  for_each(vs.begin(),vs.end(), [](Shape* p) { p->rotate(r); }); // rotate all elements of vs
  for (Shape* p : vs) p->draw(); // draw all elements of vs
}
```

Is this object-oriented? Of course it is; it relies critically on a class hierarchy with virtual functions.

It is generic? Of course it is; it relies critically on a parameterized container (vector) and the generic function `for_each`.

Is this functional? Sort of; it uses a lambda (the [] construct).

So what is it? It is modern C++: C++11.

---

http://isocpp.org/blog/2014/12/myths-1 // series of 3 recent articles.

C++: What's [still] missing

- XML
- Packages / modules
  - PHP composer; Python pip; Ruby gems; LuaRocks
  - Java JPM4j anyone`?
  - Microsoft has #import <typelib> for C++ COM
- Garbage collection (as C#, Java, Python, …)
  - Discussion point: Make it optional?
- Strings in switch() statement as in C#
  - *Here's a cute solution on CodeProject*
- Decimal data type

What's [still] missing

- Serialization:
  - see [google FlatBuffers](#) – good, modern C++
- Run-time introspection (C#, Java, Python, …)
  - C++11 annotations – not much here
- Interfaces: Need [ugly] abstract classes
- Command-line parsing
- Binary-compatible outputs
  - No standard "name mangling" of C++ classes
  - Fallback is to expose 'C' interfaces instead of rich C++ datatypes and objects. ☹
  - '[hourglass interfaces](#)': interesting cppcon presentation.

C++: What's still missing

- Networking: Will probably be stealing Boost.Asio

"Boost.Asio is a cross-platform C++ library for network and low-level I/O programming that provides developers with a consistent asynchronous model using a **modern C++** approach."

http://www.boost.org/doc/libs/1_57_0/doc/html/boost_asio.html

Which C++ compiler am I using?

Windows driver kit (WDK):

```
c:\apps\WinDDK\7600.16385.1> bin\setenv.bat .
WLH
```

```
c:\apps\WinDDK\7600.16385.1> cl.exe
```

```
Microsoft (R) 32-bit C/C++ Optimizing
Compiler Version 15.00.30729.207 for 80x86
```

This is C++ '03'

Which C++ compiler?

Set the environment; run CL.EXE

- Visual Studio help/about: fail
- Look for "C++" [here](#) for all MS C++ compiler versions

vs2013env: Microsoft (R) C/C++ Optimizing Compiler Version **18.00.31010** for x86

vs2012env: Microsoft (R) C/C++ Optimizing Compiler Version **17.00.61030** for x86

vs**2010**env: Microsoft (R) 32-bit C/C++ Optimizing Compiler Version **16.00.40219.01** for 80x86

vs**2008**env: Microsoft (R) 32-bit C/C++ Optimizing Compiler Version **15.00.30729.01** for 80x86

vs**2014**env: Microsoft (R) C/C++ Optimizing Compiler Version 19.00.22129.1 for x86

Which C++ compiler?

On a fairly up-to-date linux box:

```
dwight@dwight-mint17 ~ $ g++ --version
g++ (Ubuntu 4.8.2-19ubuntu1) 4.8.2
Copyright (C) 2013 Free Software Foundation, Inc.
```

- GCC: "[GNU compiler collection](#)"
- g++ is the C++ compiler command
- gcc is the C compiler command

MS C/C++ runtime dependencies

```
c:\apps\vs2013\VC\redist\x64> tree /f /a
+---Microsoft.VC120.CRT
|       msvcp120.dll        // C++ runtime; 600+ Kb
|       msvcr120.dll        // C runtime; 2+ Mb
|       vccorlib120.dll     // C++/CLI (.NET)
+---Microsoft.VC120.CXXAMP // 'accelerated massive
                            //     parallelism'
|       vcamp120.dll
+---Microsoft.VC120.MFC
|       mfc120u.dll
|       mfcm120u.dll
\---Microsoft.VC120.OpenMP
        vcomp120.dll
```

That last one: "…multi-platform shared-memory parallel programming in C/C++ and Fortran"

C++ Idioms

- [RAII](#)
  - "Resource Acquisition Is Initialization"
  - Acquire and release things in constructor and destructor

- [SFINAE](#)
  - "Substitution Failure Is Not An Error"
  - You won't get compile-time errors when fiddling with templates

RAII

- Constructor acquires resource
  - e.g. opens file; allocate memory
- All other member functions know resource is acquired
  - Do not need to test and make sure
- Destructor releases resources
  - Works even in the presence of exceptions

C++: Rule of Three

http://en.cppreference.com/w/cpp/language/rule_of_three

"If a class requires a user-defined destructor, a user-defined copy constructor, or a user-defined copy assignment operator, it almost certainly requires all three.

C++11: Rule of 5

http://en.cppreference.com/w/cpp/language/rule_of_three

"Because the presence of a user-defined destructor, copy-constructor, or copy-assignment operator prevents implicit definition of the move constructor and the move assignment operator, any class for which move semantics are desirable, has to declare all five special member functions."

(Ugh.)

The Big C++11 List

- Wikipedia: take a look.
  http://en.wikipedia.org/wiki/C%2B%2B11

Some of My Favorite Things…

String Literals

Create a

Page | Discussion

C++ / C++ language / Expressions

string literal

Syntax

" (*unescaped_character*|*escaped_character*)* **"**	(1)	
L " (*unescaped_character*|*escaped_character*)* **"**	(2)	
u8 " (*unescaped_character*|*escaped_character*)* **"**	(3)	(since C++11)
u " (*unescaped_character*|*escaped_character*)* **"**	(4)	(since C++11)
U " (*unescaped_character*|*escaped_character*)* **"**	(5)	(since C++11)
prefix(optional) **R "** *delimiter* **(** *raw_character** **)** *delimiter* **"**	(6)	(since C++11)

String Literals

1) Narrow multibyte string literal. The type of an unprefixed string literal is const char[]

2) Wide string literal. The type of a L"..." string literal is const wchar_t[]

3) UTF-8 encoded string literal. The type of a u8"..." string literal is const char[]

4) UTF-16 encoded string literal. The type of a u"..." string literal is const char16_t[]

5) UTF-32 encoded string literal. The type of a U"..." string literal is const char32_t[]

6) Raw string literal. Used to avoid escaping of any character, anything between the delimiters becomes part of the string, if *prefix* is present has the same meaning as described above.

Note: C and C++ do not have string types; the libs have 'em!

Note: these Unicode literals are not in MSVC++ ('raw' is)

String Literals: Raw

- YAY! Needed this from day 1:

```cpp
#include <string>
#include <iostream>
int main() {
  std::string s = R"#(\>%^\t\n&*(<)#";
  std::cout << s << std::endl;
  s = R"gobbledygook(a raw string literal with "gobbledygook"
     as the delimiter)gobbledygook";
  std::cout << s << std::endl;
  return 0;
}
```

\>%^\t\n&*(<

a raw string literal with "gobbledygook" as the delimiter

nullptr

- NULL is dead. It's ambiguous.
- Prefer to zero, also.
- Use nullptr wherever you used to use NULL.
- nullptr is *part of the C++ language – not the standard library.*

if (NULL == dumbPointer)… // bad

if (nullptr != dumbPointer) … // ok

delete nullptr;   // always works

Uniform Initialization

- Old:

```
std::vector <byte> bytes;
sendBytes.push_back(0x00);
sendBytes.push_back(0xC0);
sendBytes.push_back(0x00);
sendBytes.push_back(0x00);
sendBytes.push_back(0x12);
```

- New:

```
std::vector <byte> bytes {0x00, 0xC0, 0x00, 0x00, 0x12};
```

Uniform Initialization

- Avoids 'narrowing':

```
char c1 = 1.234e12;   // warning: C4244: 'initializing' :
conversion from 'double' to 'char', possible loss of data
char c2 = 54321;   // warning C4305 : '=' : truncation
from 'int' to 'char'


char c1 {1.234e12}; // error C2397: conversion from
'double' to 'char' requires a narrowing conversion
char c2 {54321};   // error C2397: conversion from 'int'
to 'char' requires a narrowing conversion
```

Smart Pointers

- C++ does not have garbage collection: It is deterministic in its acquisition and release of memory and other resources.

- `std::auto_ptr` is **deprecated; do not use it**
  - Failed to play well with std lib collections – std::list, std::vector
  - Still in use; don't panic.
  - Just don't write any new stuff with std::auto_ptr

C++ Smart Pointers
std::unique_ptr

- New to C++11

- Use instead of deprecated `std::auto_ptr`

- Use it wherever you are tempted to use an old fashioned dumb pointer (!!!)

- Plays well with std collections

old:

```
CPrinter* pPrinter = new CPrinter {};
```

new:

```
std::unique_ptr <CPrinter> printer (new CPrinter {});
```

C++ smart pointers

- std::unique_ptr if only one object needs access to the underlying pointer
- std::shared_ptr if several want to use the same underlying pointer
  - Cleaned up when the last copy goes out of scope
- In <memory> header file
- *"If you're using* new *or* delete, *you're doing it wrong."* –Kate Gregory (Microsoft)

Smart Pointers
`std::shared_ptr`

- New to C++11
- Similar to unique_ptr – but **reference counted**
- Plays well with std collections
  - Store objects
  - Store pointers to objects
  - Has some overhead
- As with unique_ptr, you'll still have a 'new' – but no 'delete': They're SMART!

std::to_string

- "Converts a numeric value to [std::string](http://en.cppreference.com/w/cpp/string/basic_string/to_string)."
http://en.cppreference.com/w/cpp/string/basic_string/to_string
- Avoid atof(), atoi(), Unicode macros, …
- Use to_wstring() for wide strings

```cpp
#include <iostream>
#include <string>

int main() {
    double f = 23.43;
    std::string f_str = std::to_string(f);
    std::cout << f_str << std::endl;
}

output: 23.430000
```

std::to_string

- Works with all sorts of numeric types
  - **Caveat** from Google FlatBuffers:

```
namespace flatbuffers {
  // Convert an integer or floating point value to a string.
  // In contrast to std::stringstream, "char" values are converted
  // to a string of digits.
  template<typename T> std::string NumToString(T t)
{
     // to_string() prints different numbers of digits
     // for floats depending on platform and isn't available
     // on Android, so we use stringstream
     std::stringstream ss;
     ss << t;
     return ss.str();
}
```

Auto: Type Inference

- C / C++ "auto" keyword was rarely used
  - Different behavior in different compiler implementations
- This is just like the "var" keyword in C#
- Compiler determines type at compile-time
  - Your IDE can be your friend: *Hover with your mouse, Luke…*
- Yes; there are a lot of pros/cons for usage in different context.

Auto: Type Inference

```
auto num_printers = 23;
```

```cpp
 8    std::vector <std::wstring> ribbonNames {L"YMCKT", L"KT", L"UV"};
 9
10    for (std::vector <std::wstring>::iterator it = ribbonNames.begin(); it < ribbonNames.end(); it++) {
11        std::wcout << "ribbon name: " << it->c_str() << std::endl;
12    }
13
14    for (auto& it = cbegin(ribbonNames); it < cend(ribbonNames); it++) {
15        std::wcout << "ribbon name: " << it->c_str() << std::endl;
16    }
```

```
output (twice):
ribbon name: YMCKT
ribbon name: KT
ribbon name: UV
```

Google C++ Guide: Initialization

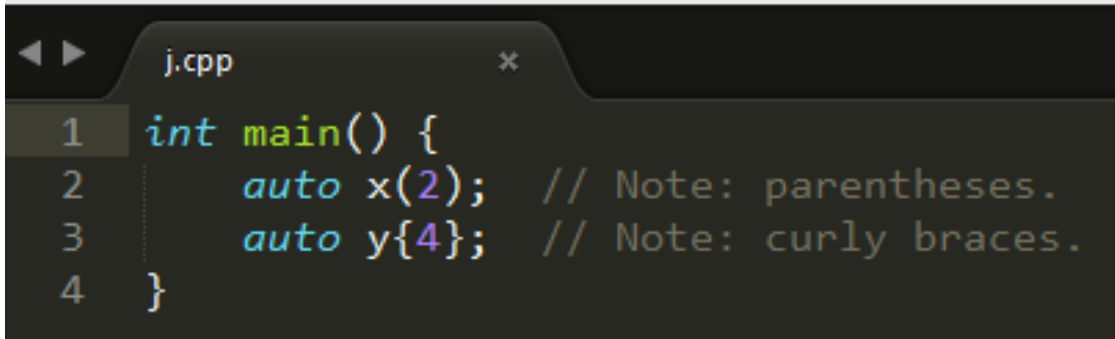http://google-styleguide.googlecode.com/svn/trunk/cppguide.html

"Programmers have to understand the difference between auto and const auto& or they'll get copies when they didn't mean to.

The interaction between auto and C++11 brace-initialization can be confusing. The declarations:

```
auto x(3);  // Note: parentheses.
auto y{3};  // Note: curly braces.
```

mean different things — x is an int, while y is a std::initializer_list<int>. The same applies to other normally-invisible proxy types.

Google C++ guide

```cpp
int main() {
    auto x(2);   // Note: parentheses.
    auto y{4};   // Note: curly braces.
}
```

```
cl /FA j.cpp    // show me your machine code

Directory of D:\temp\j
11/07/2014  08:23 AM                    564 j.asm
11/07/2014  08:17 AM                     94 j.cpp
11/07/2014  08:23 AM                 83,456 j.exe
11/07/2014  08:23 AM                    475 j.obj
```

Google C++ guide


```asm
; Listing generated by Microsoft (R) Optimizing Compiler Version 19.00.22129.1
        TITLE   D:\temp\j\j.cpp
PUBLIC  _main
; Function compile flags: /Odtp
_TEXT   SEGMENT
_y$ = -8                           ; size = 4
_x$ = -4                           ; size = 4
_main   PROC
; File d:\temp\j\j.cpp
; Line 1
    push    ebp
    mov ebp, esp
    sub esp, 8
; Line 2
    mov DWORD PTR _x$[ebp], 2
; Line 3
    mov DWORD PTR _y$[ebp], 4
; Line 4
    xor eax, eax
    mov esp, ebp
    pop ebp
    ret 0
_main   ENDP
_TEXT   ENDS
END
```

Lambdas:
Lambda Expressions

http://isocpp.org/wiki/faq/cpp11-language#lambda

http://codexpert.ro/blog/2014/10/25/c11-lets-write-a-hello-lambda/

Lambda Expressions

- Part of C++11 *language* – not the C++ Standard Library
- Alternative to function objects ('functors'), and plain functions
- Per the ISO standard document: "Lambda expressions provide a concise way to create simple function objects."
- Unnamed
  - But you can give them names: They are expressions, and you can just assign them to a variable name.
- Very useful inside templated functions and classes