



C++11 and Compiler Update

John “JT” Thomas

Sr. Director Application Developer
Products

About this Session

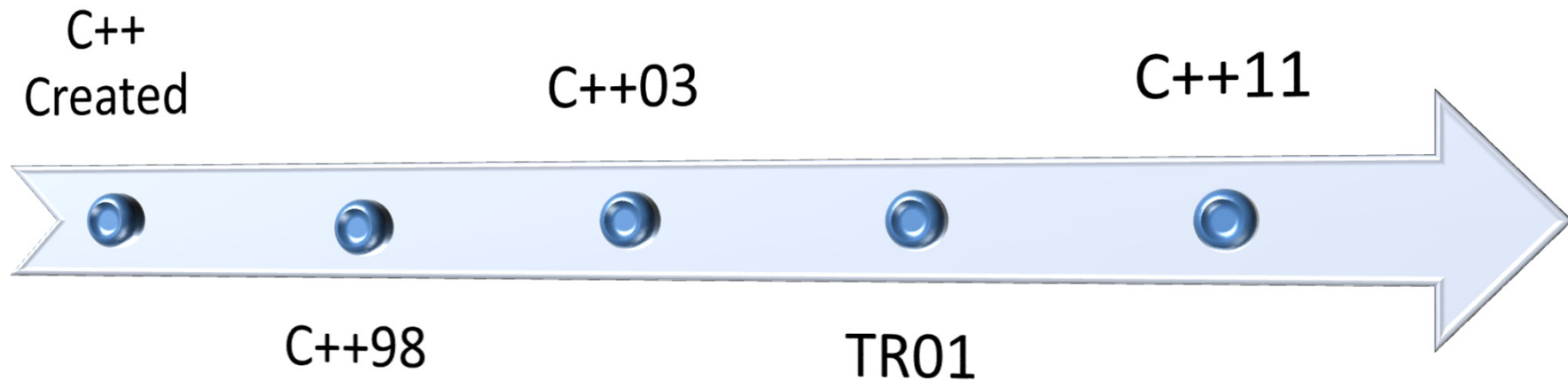
- A Brief History
- Features of C++11 you should be using now
- Questions

Bjarne Stroustrup



- C with Objects (1979)
 - Modeled OO after Simula and Ada
 - But syntax and RTL based on C
 - Classes
 - Inheritance
 - Inlining
 - Default arguments
 - Type checking
 - CFront compiler

A Brief History of C++



New Turbo C++ Professional

When the object is programming



Be Objective Object-Oriented Programming

(OOP) is programming in the '90s. It's the next step after structured programming and is the best way to write applications. So Borland combined the power of OOP with the efficiency of C to produce new Turbo C++ Professional.

And Turbo C++ Professional is the first Turbo-charged native code C++ compiler that brings Object-Oriented Programming to your PC. Since Turbo C++ Professional also compiles ANSI C code, you can be productive with C now, and move to C++ at your own pace.

Environment ++

The best compiler deserves the best environment, and our new Programmer's Platform™ environment makes you more productive. It features overlapping windows and mouse support. And sports a new multi-file editor, an integrated debugger, and a smart project manager. Its advanced open architecture lets you integrate the tools you need to feel right at home.

VROOOM adds room

VROOOM™ (Virtual Runtime Object-Oriented Memory Manager) lets you break

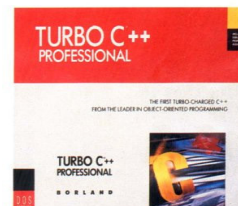
the 640K barrier. Just select the application code you want to overlay, and VROOOM does the rest—swapping modules on demand. It's fast, easy, automatic.

Another +

Turbo C++ Professional gives you all the tools you need to build fast, reliable C++ programs.

Turbo Debugger® 2.0 debugs your object-oriented programs. This powerful new version is the first and only debugger to support *reverse execution*. Letting you step backwards through your code to find the bugs you might have missed.

New Turbo Profiler™, the world's first interactive profiler, displays histograms of your program's performance. With it, you



can easily spot execution bottlenecks, and see where improvements or redesign of your code will yield maximum performance gains.

And Turbo Assembler® 2.0 lets you replace time-critical segments of your code using the world's fastest MASM-compatible assembler.

Turbo C++ Professional Compiler

- C++ conforming to AT&T's 2.0 specification
- C++ class libraries
- Full ANSI C compiler
- VROOOM overlay manager
- Complete documentation and tutorials

Programmer's Platform

- Open architecture for integration of your own tools
- Overlapping windows with mouse support
- Multifile, macro-based editor
- Smart project manager provides visual MAKE
- Integrated debugging and hypertext help

Turbo Debugger 2.0

- Class hierarchy browser and inspectors
- Reverse execution provides "true" undo
- 286 protected-mode and 386 virtual-mode debugging
- Keystroke record and playback

NEW Turbo Profiler

- Displays histograms of program execution
- Tracks call history, overlays, interrupts, file I/O

Turbo Assembler 2.0

- Multipass assembler with NOP squishing and 486 support

Special Introductory Offer

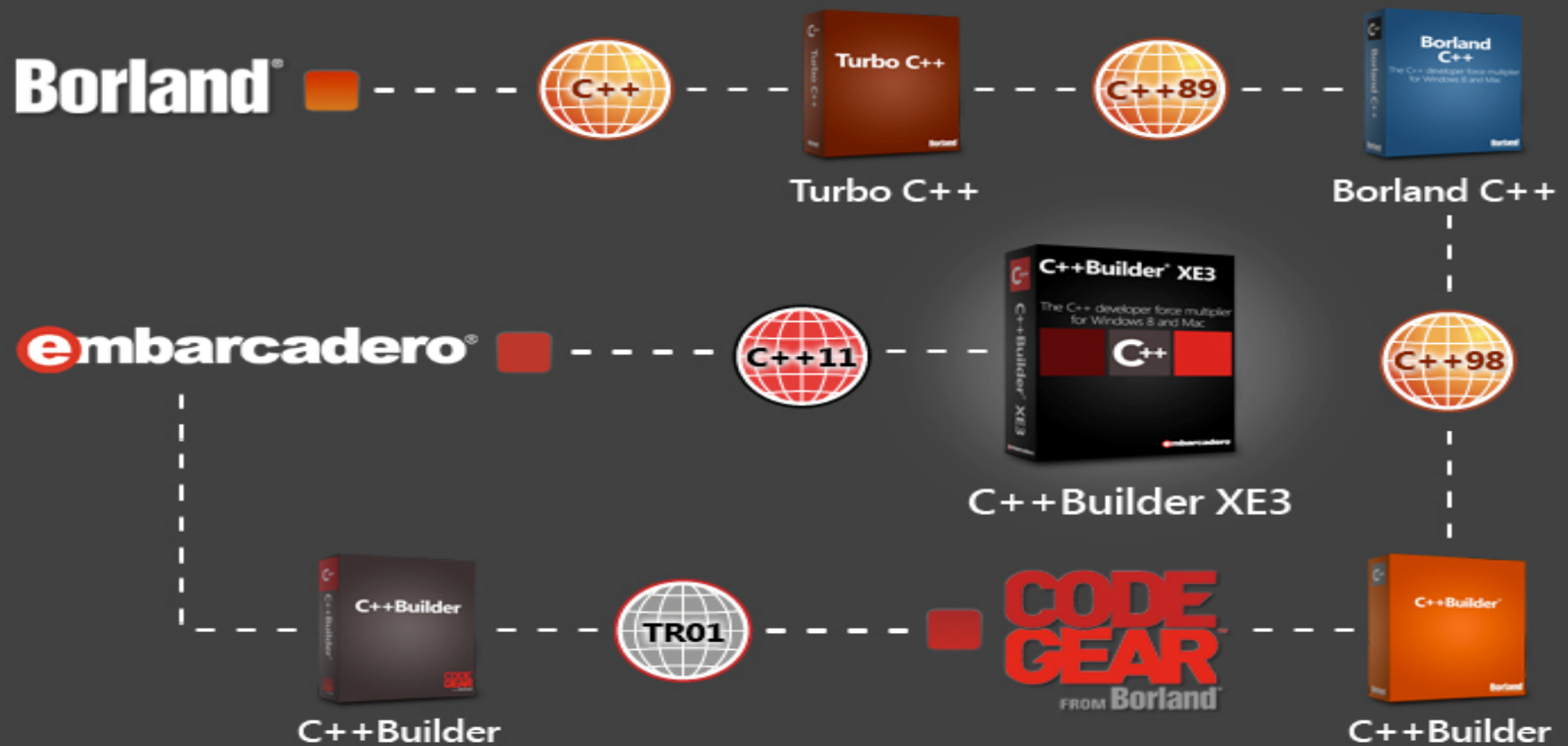
The suggested retail price for Turbo C++ Professional is \$299.⁹⁵ (\$199.⁹⁵ for Turbo C++). For a limited time, Borland is offering its dealers and distributors special introductory discounts.* So be objective, and **SEE YOUR DEALER** or call Borland** at 1-800-331-0877 now!

B O R L A N D

Code: MC66

*Offer expires July 31, 1990 or while supplies last. Offer good in United States and Canada only. **Special discounts for registered Turbo C++ users are available from Borland. Mail orders to: Borland, P.O. Box 660001, Scotts Valley, CA 95067-0001. For orders outside the U.S., call (408) 438-5300. Turbo C++ and Turbo Debugger, Turbo Profiler and Turbo Assembler are trademarks or registered trademarks of Borland International, Inc. Copyright © 1990, Borland International, Inc. All rights reserved. 90-1333
embarcadero

Path to C++

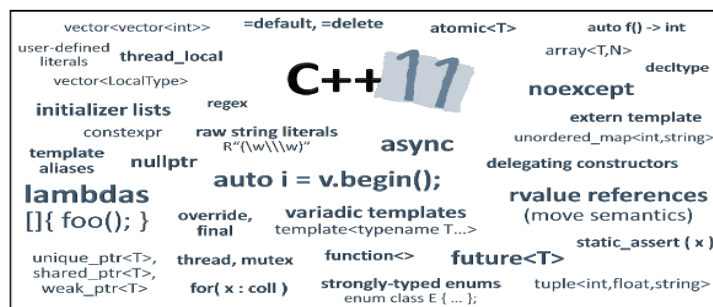


C++11 – A new Standard



Language

- Rvalue references and move constructors
- constexpr - Generalized constant expressions
- Core language usability enhancements
- Initializer lists
- Uniform initialization
- Type inference
- Range-based for-loop
- Lambda functions and expressions
- Alternative function syntax
- Object construction improvement
- Explicit overrides and final
- Null pointer constant
- Strongly typed enumerations
- Right angle bracket
- Explicit conversion operators
- Alias templates
- Unrestricted unions



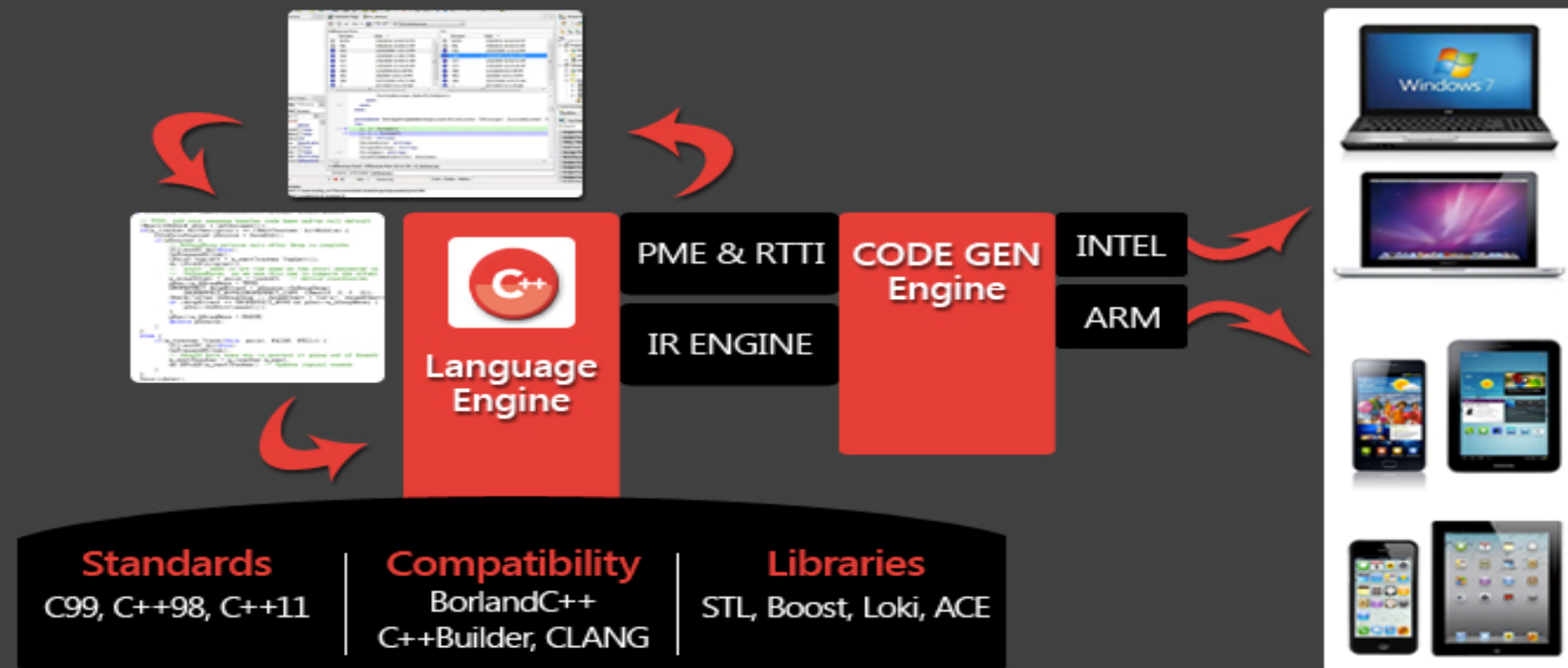
Library

- Variadic templates
- New string literals
- User-defined literals
- Multithreading memory model
- Thread-local storage
- Explicitly defaulted and deleted special member functions
- Type long long int
- Static assertions
- Allow sizeof to work on members of classes without an explicit object
- Control and query object alignment
- Allow garbage collected implementations
- Threading facilities
- Tuple types
- Hash tables
- Regular expressions
- General-purpose smart pointers
- Extensible random number facility
- Wrapper reference
- Polymorphic wrappers for function objects
- Type traits for metaprogramming



An Architecture for the Future

Rapid Application Development



auto keyword

- Type inference
- auto asks the compiler to deduce the type
- Helps with readability of code especially with template container iterators
- `auto i = 9; //int`
- `auto f = 3.14f; //float`
- `auto f = new foo(); //foo*`

```
//-----  
void __fastcall TForm5::Button1Click(TObject *Sender)  
(X)  std::map<std::string, std::vector<int>>> map;  
    for(auto i = map.begin(); i != map.end(); i++);  
}  
//-----
```

ranged for loop

- a `for_each` concept to loop through standard containers
- Any type with a `begin()` and `end()`
- `for (auto i : map)`
 - access `i` by value
- Use `auto& i` for ref to modify
- Use `auto const &i` for ref to read only

```
//-----  
void __fastcall TForm5::Button1Click(TObject *Sender)  
{  
    std::map<std::string, std::vector<int>> map;  
    for(std::map<std::string, std::vector<int>>::iterator i = map.begin(); i != map.end(); i++);  
}  
//-----  
  
//-----  
void __fastcall TForm5::Button1Click(TObject *Sender)  
{  
    std::map<std::string, std::vector<int>> map;  
    for(auto i = map.begin(); i != map.end(); i++);  
}  
//-----
```

Smart pointers

- C++ version of ARC
- `auto_ptr` is deprecated
- 3 new smart pointers in C++11
 - `unique_ptr`, `shared_ptr`, and `weak_ptr`

```
//-----
void __fastcall TForm5::Button1Click(TObject *Sender)
{
    std::unique_ptr<int> p1(new int(9));
    std::unique_ptr<int> p2 = p1; //transfers ownership, p1 = nullptr
}
//-----

//-----
void __fastcall TForm5::Button1Click(TObject *Sender)
{
    std::shared_ptr<int> p1(new int(9));
    std::shared_ptr<int> p2 = p1; //ref_count++
}
//-----

//-----
void __fastcall TForm5::Button1Click(TObject *Sender)
{
    std::shared_ptr<int> p1(new int(9));
    std::weak_ptr<int> p2 = p1; //no ref_count
}
//-----
```

- `unique_ptr`
 - Ownership of memory does not have to be shared
 - Can be transferred to another `unique_ptr` with a move constructor
- `shared_ptr`
 - Ownership of memory is shared (reference counted)
- `weak_ptr`
 - Reference to an object managed by `shared_ptr` but does not contribute to the ref count
 - Used to avoid reference cycle

Move semantics

- Modify rvalue references
- Specified by &&
- Implemented as a move constructor and move assignment operator

```
#include <memory>
//-----

template <typename T>
class foobar
{
    std::string      _name;
    size_t           _size;
    size_t - Projects/Unit3.h (12) _buffer;
public:
    // default constructor
    foobar();

    // constructor
    foobar(const std::string& name, size_t size);

    // copy constructor
    foobar(const foobar& copy);

    // copy assignment operator
    foobar& operator=(const foobar& copy);

    // move constructor
    foobar(foobar&& temp);

    // move assignment operator
    foobar& operator=(foobar&& temp);
};
```

Uniform initialization syntax

- C++ have a long history with sporadic compatibility with C
 - Thus, there are several ways to initiate a variable
- STL containers required dynamic initialization
 - i.e. push_back

```
//-----  
void __fastcall TForm5::Button1Click(TObject *Sender)  
{  
    std::vector<std::string> vs {"C++Builder", " likes ", "C++11"};  
}  
//-----
```

Override and final

- Deals with typical inheritance issues/mistakes
 - Differing signatures accidentally resulting in a non override
- **Override** indicates a method is supposed to override a virtual method in its base

- **final** indicates a method can no longer be overridden

```
//-----  
class foo  
{  
public:  
    virtual void func(int) {std::cout << "foo::func" << std::endl;}  
};  
  
class foo : public foo  
{  
public:  
    virtual void func(int) override final {std::cout << "foo::func" << std::endl;}  
};  
  
class bar : public foo  
{  
public:  
    virtual void func(int) override {std::cout << "bar::func" << std::endl;} //can't be overridden  
};  
//-----
```

nullptr

- Zero (integer) used to be value of null pointers
 - Could result in implicit type conversion
- nullptr is of type `std::nullptr_t` that represents a null pointer literal
- Implicit conversion from
 - nullptr to null pointer value of any pointer type

lambda expressions

- New syntax for defining anonymous functions
- Can be used to define closures to capture variables
- [capture-list] (params) {body;}
- [capture-list] (params) ->ret {body;}
- [capture-list] (params) mutable exception attributed -> ret {body;}

```
//-----  
void __fastcall TForm4::Button1Click(TObject *Sender)  
{  
    std::vector<std::string> vs {"C++Builder", " supports", " C++11"};  
    std::string s;  
    for(auto i : vs)  
        [&] () {s += i;} ();  
    ShowMessage(s.c_str());  
}  
//-----
```



C++ Compiler Update

Active projects

- Move to CLANG 3.3
 - Full C++ compliance
 - futures/async
- Support C++11 on all platforms
 - Win32 CLANG based toolchain
- Current version owners will be invited to participate in a beta when it is released



Questions?