# Lecture 12: Programming Languages

Dr. Mohammed Hawa
Electrical Engineering Department
University of Jordan



## Which language should we pick?

- Many programming languages are in-use nowadays. The following are popular:
  - C/C++
  - Java (Oracle/Google/Android)
  - C# (Microsoft/.NET Framework)
  - Objective-C and Swift (Apple/iOS)
  - Python, Perl, Ruby (scripting)
  - MATLAB
  - Visual BASIC, Fortran, COBOL (High-Level)
  - Assembly (Low-Level)

# High- vs. Low-Level

- **High-level** programming languages provide strong abstraction (e.g., Java, Python, Ruby, COBOL and Fortran)
  - Natural language elements (similar to what humans speak)
  - Easier to use for non-engineers (IT)
  - Some automation is possible (e.g. memory management)
  - No need to know: registers, data segments/stack segment/instruction pointer, cache, …
  - Performance can be affected (sometimes significantly)
- **Low-level** programming languages (e.g., Assembly)
  - More difficult to program (more time-consuming)
  - Speaks the language of the CPU (registers and memory)
  - Fine-grain (complete) control of the CPU
  - Resulting in the highest possible performance
- **Medium-Level** (e.g., C/C++) attempts to balance the advantages of both worlds

# Recent: Native vs. Managed Code

- **Managed code** executes under the management of a virtual machine (VM)
  - Source code becomes *bytecode* for the VM ("simulates" a processor in software)
  - The result is not machine code.
  - Eases handling memory management (Garbage Collector) and exceptions (safer code against mistakes and memory leaks)
  - Performance hit due to running the code inside the VM
  - One way to speed the execution (but with slower startup time) is JIT (Just-in-Time) compiling or Ahead-of-Time (AOT) compiling the byte code by the VM into machine code
- Examples:
  - C# runs under the .NET Framework.
  - Java runs under the Java VM (Dalvik and ART in Android)
  - Most dynamically-typed languages — such as Python, PHP, Ruby and even Javascript— are managed code.

# Native vs. Managed Code [2]

- **Native code** is compiled directly into machine code (direct instructions for the microprocessor).
  - Manual allocation and de-allocation of memory
  - Possible mistakes (memory leaks and segmentation faults)
  - No run-time checks for common errors such as null-pointer dereferencing or array bounds overflow.
  - Very fast. No VM between the code and the CPU
- Examples: C and C++
- Many computer network simulators use C/C++
  - OPNET, OMNET++, ns-2, ns-3 (network simulator), GloMoSim/QualNet, cnet, etc

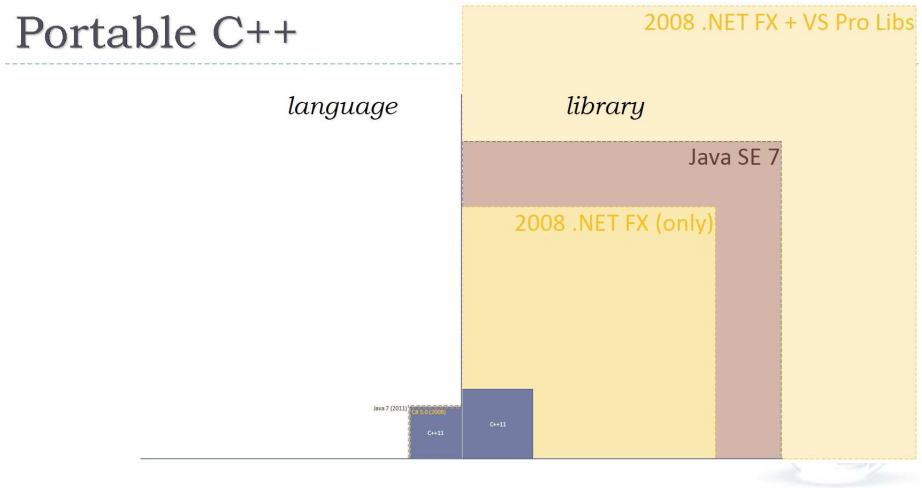
 

# Performance Matters in Simulation

- Sending a packet and processing it at the receiver
  - C/C++: 100 ms
  - Java (no JIT): 300 ms
- Comparing two networking protocols
  - Five experiments
  - High, medium, and low traffic scenarios (5 x 3 = 15 runs)
  - Each with 1000 nodes each sending 1000 packets
    - Randomness requires large numbers (flipping a coin)
  - C/C++: 15 x 1,000,000 x 100 ms = 17.36 days (2.5 weeks)
  - Java: 15 x 1,000,000 x 300 ms = 52.08 days (1.7 months)
- What about more protocols, and repeated refinements?

# Comparison: Syntax *and* Library

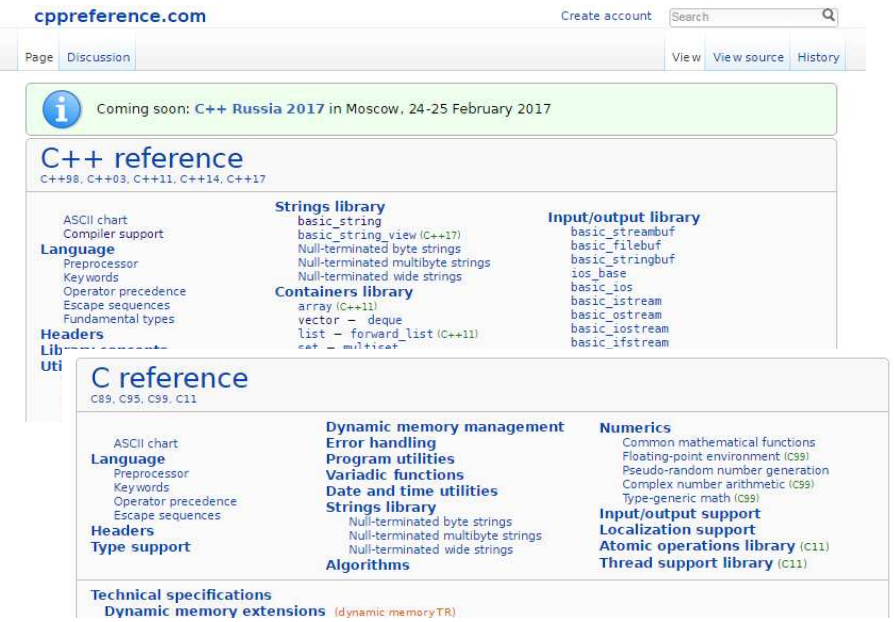


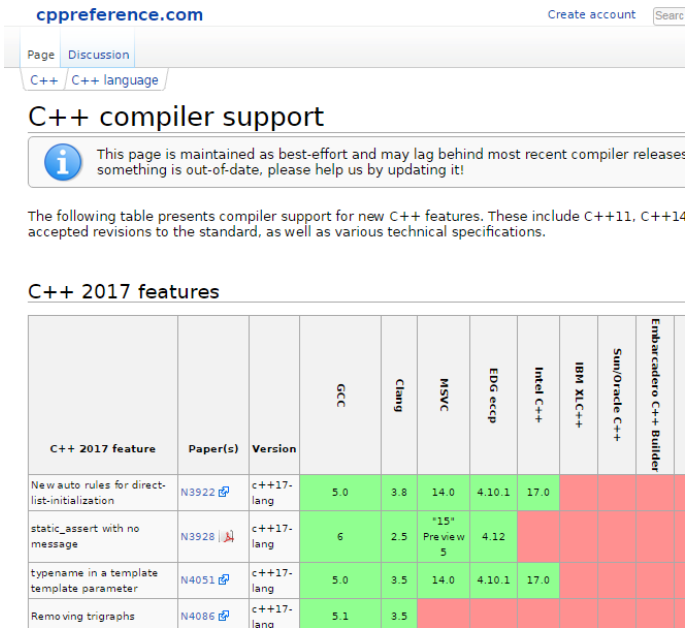
# C and C++ is standardized

- C++98 [1998] – ISO/IEC 14882:1998
- C++03 [2003] – ISO/IEC 14882:2003
  - C++TR1 [2007] – ISO/IEC TR 19768:2007
- **C++11 [2011] – ISO/IEC 14882:2011 (major overhaul)**
- C++14 [2014] – ISO/IEC 14882:2014 (minor fixes)
- C++17 [2017] – almost ready
- C89 [1989] – also known as ANSI C
- C90 [1990] – ISO 9899:1990
- C99 [1999] – ISO 9899:1999
- **C11 [2011] – ISO/IEC 9899:2011**
  - Technical Corrigendum 1 (ISO/IEC 9899:2011/Cor. 1:2012) published in 2012


cppreference.com
Create account
Search
Page
Discussion
View
View source
History
Coming soon: C++ Russia 2017 in Moscow, 24-25 February 2017
C++ reference
C++98, C++03, C++11, C++14, C++17
ASCII chart
Compiler support
Language
Preprocessor
Keywords
Operator precedence
Escape sequences
Fundamental types
Headers
Strings library
basic_string
basic_string_view (C++17)
Null-terminated byte strings
Null-terminated multibyte strings
Null-terminated wide strings
Containers library
array (C++11)
vector — deque
list — forward_list (C++11)
Input/output library
basic_streambuf
basic_filebuf
basic_stringbuf
ios_base
basic_ios
basic_istream
basic_ostream
basic_iostream
basic_ifstream
C reference
C89, C95, C99, C11
ASCII chart
Language
Preprocessor
Keywords
Operator precedence
Escape sequences
Headers
Type support
Dynamic memory management
Error handling
Program utilities
Variadic functions
Date and time utilities
Strings library
Null-terminated byte strings
Null-terminated multibyte strings
Null-terminated wide strings
Algorithms
Numerics
Common mathematical functions
Floating-point environment (C99)
Pseudo-random number generation
Complex number arithmetic (C99)
Type-generic math (C99)
Input/output support
Localization support
Atomic operations library (C11)
Thread support library (C11)
Technical specifications
Dynamic memory extensions (dynamic memory TR)




cppreference.com
Create account
Page
Discussion
C++
C++ language
C++ compiler support
This page is maintained as best-effort and may lag behind most recent compiler releases
something is out-of-date, please help us by updating it!
The following table presents compiler support for new C++ features. These include C++11, C++14
accepted revisions to the standard, as well as various technical specifications.
C++ 2017 features
C++ 2017 feature
Paper(s)
Version
GCC
Clang
MSVC
EDG eccp
Intel C++
IBM XLC++
Sun/Oracle C++
Embarcadero C++ Builder
New auto rules for direct-list-initialization
N3922
c++17-lang
5.0
3.8
14.0
4.10.1
17.0
static_assert with no message
N3928
c++17-lang
6
2.5
"15" Preview 5
4.12
typename in a template template parameter
N4051
c++17-lang
5.0
3.5
14.0
4.10.1
17.0
Removing trigraphs
N4086
c++17-lang
5.1
3.5

# We will use C and C++

- C++ is often considered to be a superset of C, with minor exceptions.
- One commonly encountered difference is that C allows implicit conversion from void* to other pointer types, but C++ does not (for type safety reasons).
- Another common issue is that C++ defines many new keywords, such as `new` and `class`, which may be used as identifiers (e.g. variable names) in C.
- Some incompatibilities have been removed by the 1999 revision of the C standard (C99), which now supports C++ features such as line comments (//), and declarations mixed with code.

# C/C++ Crash Course

- Data types and pointers
- Arrays and structures
- Control structures (if, else, for, while, do, break, continue, switch, …)
- Preprocessor: #define, #include, etc
- Algorithms and Classes
- Linked Lists
- std::vector and std::list



# Included in the Final Exam

- The 7th Edition of your textbook:
- Chapter 1 (An overview of computers and programming languages) all the way to Chapter 10 (Classes and data abstraction).
- **PLUS** Chapter 12 (Pointers, Classes, Virtual Functions and Abstract classes)
- **PLUS** Chapter 16 (Searching, Sorting, and Vector)
- **PLUS** Chapter 17 (Linked lists).
- Earlier editions are fine, but you should cover the above material.