

C + + Programming

Week 1. Introduction

Dr. Owen Chen
Cary Chinese School
Director of Math and Computer Science

2023 Summer

- 1 A Little History of C / C++ Programming Language**
- 2 Areas of Application and Popularity**
- 3 C++ Philosophy**
- 4 C++ Weaknesses**
- 5 The Course**
- 6 Installation and Hello World!**

A Little History of

C/C++

Programming

Language

The Assembly Programming Language



A long time ago, in a galaxy far,
far away...there was **Assembly**

- Extremely simple instructions
- Requires lots of code to do simple tasks
- Can express anything your computer can do
- Hard to read, write
- ...redundant, boring programming, bugs proliferation

```
main:
.Lfunc_begin0:
    push rbp
.Lcfi0:
.Lcfi1:
    mov rbp, rsp
.Lcfi2:
    sub rsp, 16
    movabs rdi, .L.str
.Ltmp0:
    mov al, 0
    call printf
    xor ecx, ecx
    mov dword ptr [rbp - 4], eax
    mov eax, ecx
    add rsp, 16
    pop rbp
    ret
.Ltmp1:
.Lfunc_end0:
.L.str:
.asciz "Hello World\n"
```

A Little History of C

In the 1969 **Dennis M. Ritchie** and **Ken Thompson** (AT&T, Bell Labs) worked on developing an operating system for a large computer that could be used by a thousand users. The new operating system was called **UNIX**

The whole system was still written in assembly code. Besides assembler and Fortran, UNIX also had an interpreter for the **programming language B**. A high-level language like B made it possible to write many pages of code task in just a few lines of code. In this way the code could be produced much faster than in assembly

A drawback of the B language was that it did not know data-types (everything was expressed in machine words). Another functionality that the B language did not provide was the use of “structures”. The lag of these things formed the reason for Dennis M. Ritchie to develop the **programming language C**. In 1988 they delivered the final standard definition ANSI C

A Little History of C



Dennis M. Ritchie, and Ken Thompson

```
#include "stdio.h"  
  
int main() {  
    printf("Hello World\n");  
}
```

A Little History of C

Areas of Application:

- UNIX operating system
- Computer games
- Due to their power and ease of use, C were used in the programming of the special effects for Star Wars



Star Wars - The Empire Strikes Back

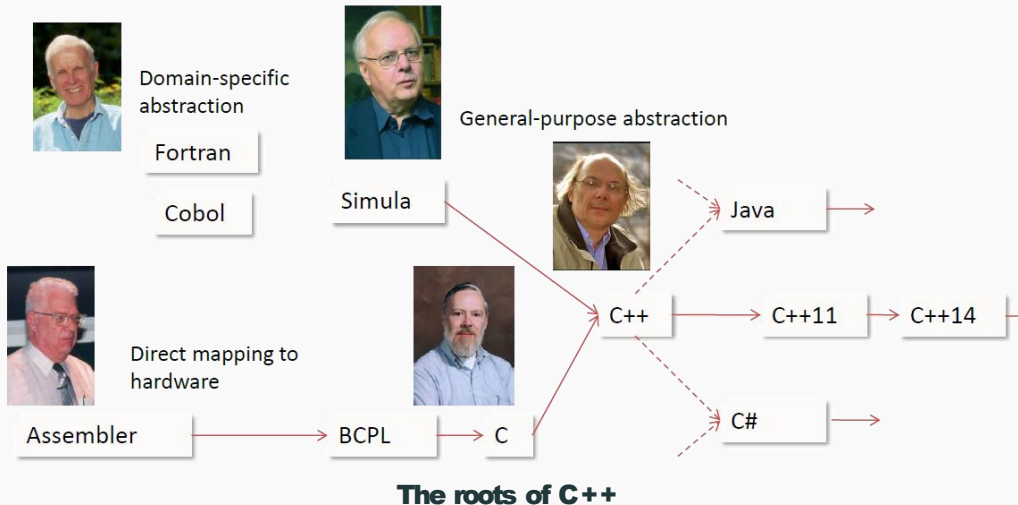
A Little History of C++

The **C++ programming language** (originally named "C with Classes") was devised by **Bjarne Stroustrup** also an employee from Bell Labs (AT&T). Stroustrup started working on C with Classes in 1979.

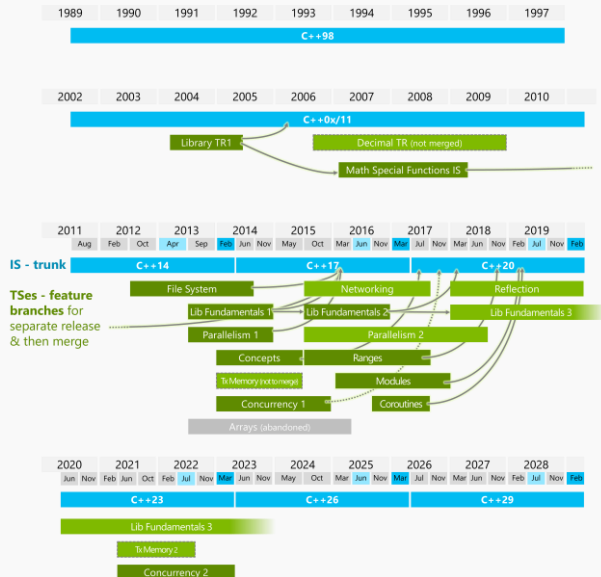
The first commercial release of the C++ language was in October 1985



A Little History of C++



A Little History of C++



“If you’re teaching today what you were teaching five years ago, either the field is dead or you are”

Prof. Noam Chomsky, MIT
father of modern linguistics










Areas of Application **and Popularity**

Most Popular Programming Languages (IEEE Spectrum - 2022)

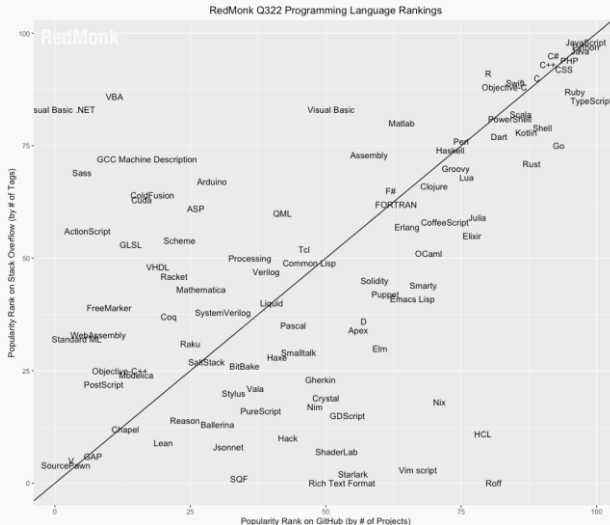
Rank	Language	Type	Score
1	Python [▼]	  	100.0
2	Java [▼]	  	95.4
3	C [▼]	  	94.7
4	C++ [▼]	  	92.4
5	JavaScript [▼]		88.1
6	C# [▼]	   	82.4
7	R [▼]		81.7
8	Go [▼]	 	77.7

[Interactive: The Top Programming Languages 2022](#)

Most Popular Programming Languages (TIOBE - December. 2022)

Programming Language		Ratings	Change
	Python	16.66%	+3.76%
	C	16.56%	+4.77%
	C++	11.94%	+4.21%
	Java	11.82%	+1.70%
	C#	4.92%	-1.48%
	Visual Basic	3.94%	-1.46%
	JavaScript	3.19%	+0.90%

Most Popular Programming Languages (Redmonk - June, 2022)



**There may be more than 200 billion lines
of C/C++ code globally**

- **Performance is the defining aspect of C++**. No other programming language provides the performance-critical facilities of C++
- **Provide the programmer control over every aspect of performance**
- **Leave no room for a lower level language**

- **Ubiquity.** C++ can run from a low-power embedded device to large-scale supercomputers
- **Multi-Paradigm.** Allow writing efficient code without losing high-level abstraction
- **Allow writing low-level code.** Drivers, kernels, assembly (asm), etc.
- **Ecosystem.** Many support tools such as debuggers, memory checkers, coverage, static analysis, profiling, etc.
- **Maturity.** C++ has a 40 years history. Many software problems have been already addressed and developing practices have been investigated

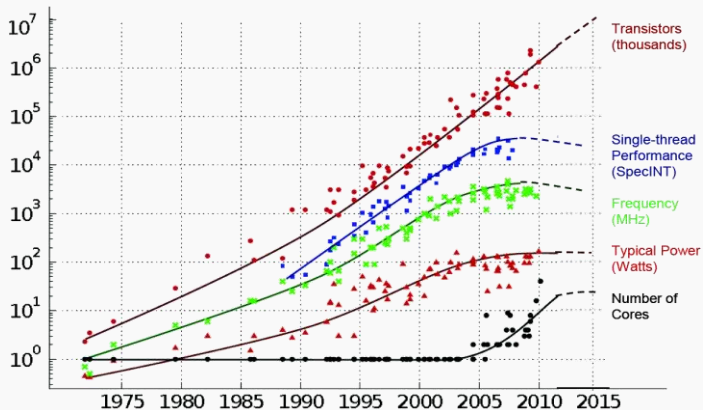
- **Operating systems:** Windows, Android, OS X, Linux
- **Compilers:** LLVM, Swift compiler
- **Artificial Intelligence:** TensorFlow, Caffe, Microsoft Cognitive Toolkit
- **Image Editing:** Adobe Premier, Photoshop, Illustrator
- **Web browser:** Firefox, Chrome, etc. + WebAssembly
- **High-Performance Computing:** drug developing and testing, large scale climate models, physic simulations
- **Embedded systems:** IoT, network devices (e.g. GSM), automotive
- Google and Microsoft use C++ for web indexing

- **Scientific Computing:** CERN/NASA*, SETI@home, Folding@home
- **Database:** MySQL, ScyllaDB
- **Video Games:** Unreal Engine, Unity
- **Entertainment:** Movie rendering (see [Interstellar black hole rendering](#)), virtual reality
- **Finance:** electronic trading systems (Goldman, JPMorgan, Deutsche Bank)**

... and many more

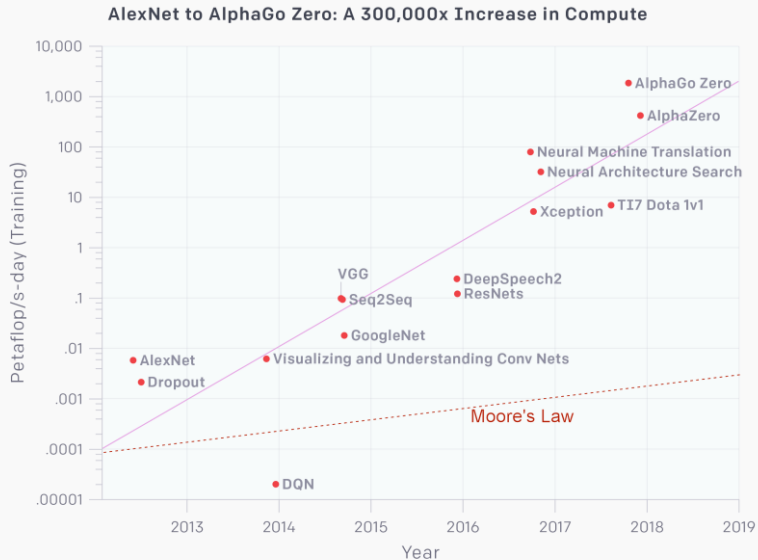
* The flight code of the NASA Mars drone for the **Perseverance** Mission, as well as the **Webb telescope** software, are mostly written in C++ github.com/nasa/fprime, [James Webb Space Telescope's Full Deployment](#)

The End of Historical Performance Scaling



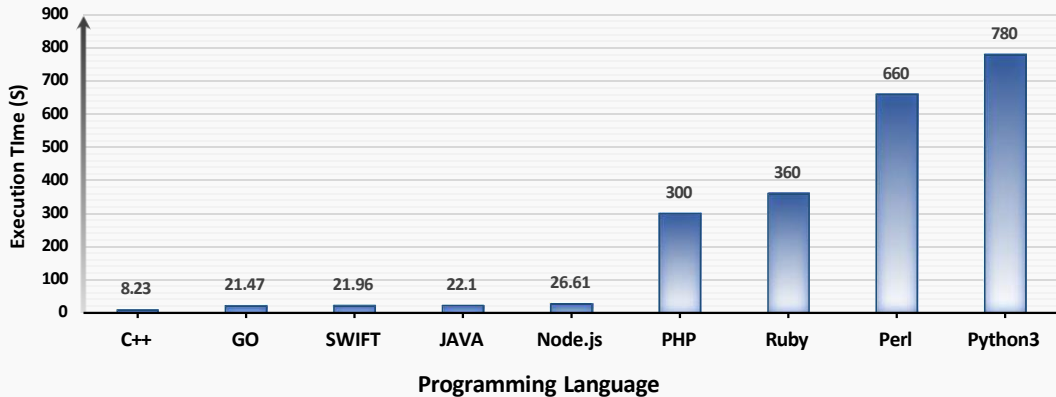
**Performance limitations influence algorithm design
and research directions**

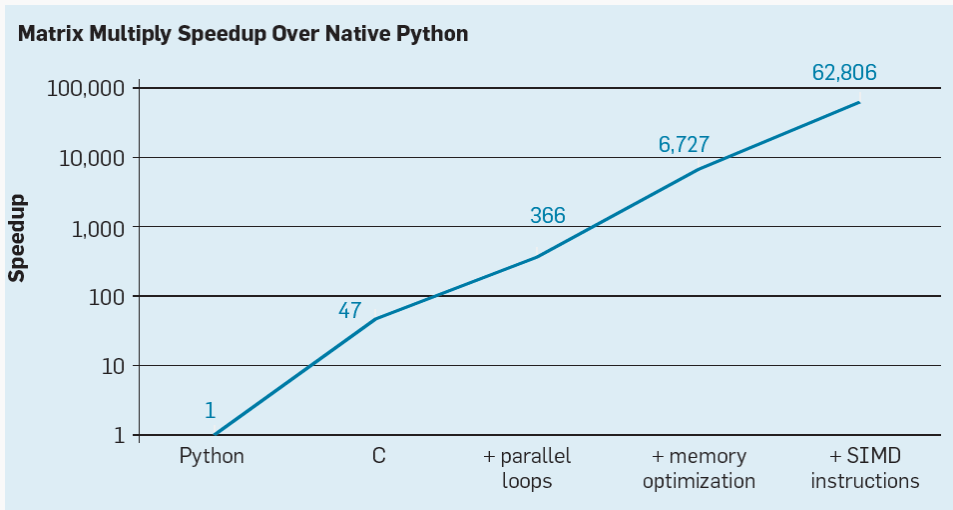
An Important Example... (AI Evolution)



N-BODY SIMULATION

PROGRAMMING LANGUAGES PERFORMANCE COMPARISON





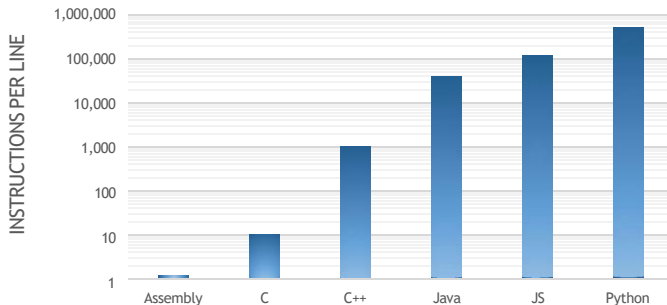
Hello World

Language	Execution Time
C (on my machine)	0.7 ms
C	2 ms
Go	4 ms
Crystal	8 ms
Shell	10 ms
Python	78 ms
Node	110 ms
Ruby	150 ms
jRuby	1.4 s

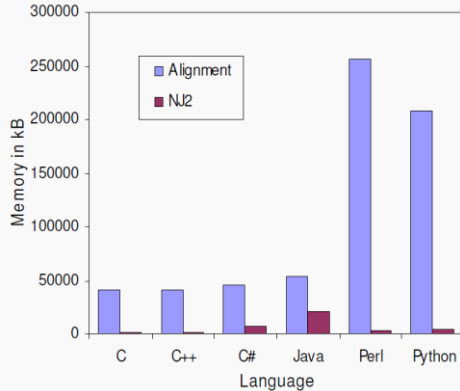
Performance/Expressiveness Trade-off



Mandelbrot Static Instructions per Line



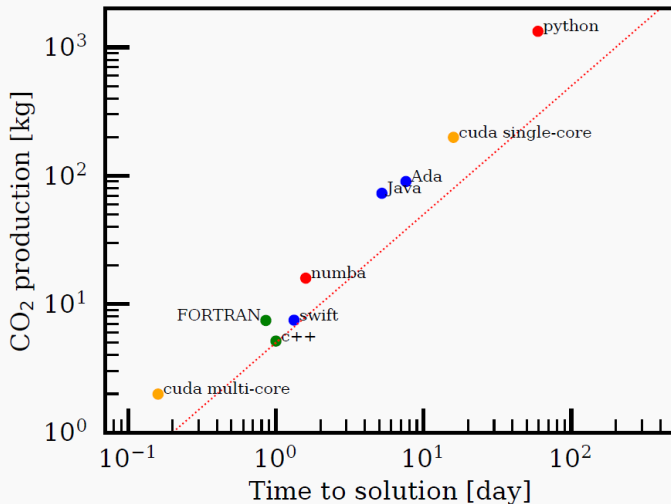
Memory Usage



Memory usage comparison of the
Neighbor-Joining and global alignment programs

[A comparison of common programming languages used in bioinformatics \(BMC Informatic\)](#)

	Energy		Time
(c) C	1.00	(c) C	1.00
(c) Rust	1.03	(c) Rust	1.04
(c) C++	1.34	(c) C++	1.56
(c) Ada	1.70	(c) Ada	1.85
(v) Java	1.98	(v) Java	1.89
(c) Pascal	2.14	(c) Chapel	2.14
(c) Chapel	2.18	(c) Go	2.83
(v) Lisp	2.27	(c) Pascal	3.02
(c) Ocaml	2.40	(c) Ocaml	3.09
(c) Fortran	2.52	(v) C#	3.14
(c) Swift	2.79	(v) Lisp	3.40
(c) Haskell	3.10	(c) Haskell	3.55
(v) C#	3.14	(c) Swift	4.20
(i) Hack	24.02	(i) PHP	27.64
(i) PHP	29.30	(v) Erlang	36.71
(v) Erlang	42.23	(i) Jruby	43.44
(i) Lua	45.98	(i) TypeScript	46.20
(i) Jruby	46.54	(i) Ruby	59.34
(i) Ruby	69.91	(i) Perl	65.79
(i) Python	75.88	(i) Python	71.90
(i) Perl	79.58	(i) Lua	82.91



USA Computing Olympiad

OVERVIEW

TRAINING

CONTESTS

HISTORY

STAFF

RESOURCES

2023 US OPEN CONTEST -- FINAL RESULTS

The 2023 US Open contest featured algorithmic programming problems covering a wide range of techniques and levels of difficulty.

A total of 6672 distinct users logged into the contest during its 4-day span. A total of 4913 participants submitted at least one solution, hailing from 74 different countries:

2751 USA	1341 CHN	147 CAN	99 KOR	54 IND	43 ISR	36 DEU
35 SGP	29 ROU	27 ARM	25 EGY	23 TWN	21 VNM	20 AUS
19 POL	17 GBR	14 HKG	12 FRA	11 TUR	11 SLV	11 GEO
9 MYS	8 TUN	8 NZL	8 MNG	7 UZB	7 SYR	7 JPN
7 BRA	6 CHE	5 ZAF	5 UKR	5 KGZ	5 BGD	4 RUS
4 KAZ	4 FIN	4 AZE	3 THA	3 PER	3 PAK	3 NLD
3 IRL	3 IDN	3 GRC	3 BLR	3 BGR	2 TKM	2 SRB
2 SAU	2 NGA	2 MEX	2 IRN	2 ETH	2 EST	2 CUB
2 COL	1 SVK	1 REU	1 PRK	1 PHL	1 MLT	1 MKD
1 LTU	1 LKA	1 KWT	1 KHM	1 ITA	1 ISL	1 ESP
1 COD	1 CHL	1 ARE	1 AFG			

In total, there were 10724 graded submissions, broken down by language as follows:

5573 C++17
 1878 C++11
 1862 Java
 1360 Python 3.6.9
 38 C
 13 Python 2.7.17

C + + Philosophy

*Do not sacrifice **performance** except as a last resort*

Zero Overhead Principle (zero-cost abstraction)

“it basically says if you have an abstraction it should not cost anything compared to write the equivalent code at lower level”

“so I have say a matrix multiply it should be written in a such a way that you could not drop to the C level of abstraction and use arrays and pointers and such and run faster”

Bjame Stroustrup

31

*Enforce **safety at compile time** whenever possible*

Statically Typed Language

“The C++ compiler provides type safety and catches many bugs at compile time instead of run time (a critical consideration for many commercial applications.)”

www.python.org/doc/FAQ.html

- The *type annotation* makes the code more readable
- Promote compiler optimizations and runtime efficiency
- Allow users to define their own type system

- **Programming model:** *compartmentalization*, only add features if they solve an actual problem, and allow *full control*
- **Predictable runtime** (under constraints): no garbage collector, no dynamic type system → *real-time systems*
- **Low resources:** low memory and energy consumption → *restricted hardware platforms*
- **Well suited for static analysis** → *safety critical software*
- **Portability** → Modern C++ standards are highly portable

Who is C++ for?

“C++ is for people who want to use hardware very well and manage the complexity of doing that through abstraction”

Bjame Stroustrup

“a language like C++ is not for everybody. It is generated via sharp and effective tool for professional basically and definitely for people who aim at some kind of precision”

Bjame Stroustrup



C++ Weaknesses

... and why teaching C++ as first programming language is a bad idea?

C++ is the hardest language from students to master

- *More languages in one*
 - Standard C/C++ programming
 - Preprocessor
 - Object-Oriented features
 - Templates and Meta-Programming
- *Huge set of features*
- *Worry about memory management*
- *Low-level implementation details:* pointer arithmetics, structure, padding, undefined behavior, etc.
- *Frustrating:* compiler/runtime errors (e.g. seg. fault)

“C makes it easy to shoot yourself in the foot; C++ makes it harder, but when you do, it blows your whole leg off”

Bjarne Stroustrup, Creator of the C++ language

“The problem with using C++...is that there's already a strong tendency in the language to require you to know everything before you can do anything”

Larry Wall, Creator of the Perl language

“Despite having 20 years of experience with C++, when I compile a non-trivial chunk of code for the first time without any error or warning, I am suspicious. It is not, usually, a good sign”

Daniel Lemire, Prof. at the University of Quebec

Backward-compatibility

“**Dangerous defaults and constructs**, often originating from C, cannot be removed or altered”

“Despite the hard work of the committee, **newer features sometimes have flaws that only became obvious after extensive user experience**, which cannot then be fixed”

“C++ practice has put an **ever-increasing cognitive burden** on the developer for what I feel has been very little gain in productivity or expressiveness and at a huge cost to code clarity”

C++ critics and replacements:

- [Epochs: a backward-compatible language evolution mechanism](#)
- [Goals and priorities for C++](#)
- [Carbon Language](#)
- [Circle C++ Compiler](#)
- [Cppfront: Can C++ be 10x simpler & safer ... ?](#)

*Every second spent trying to understand the
language is one not spent understanding the
problem*

(Un)official C++ reference:*

- en.cppreference.com

Tutorials:

- www.learncpp.com
- www.tutorialspoint.com/cplusplus
- en.wikibooks.org/wiki/C++
- [yet another insignificant...programming notes](#)

Other resources:

- stackoverflow.com/questions/tagged/c++

* The full C++ standard draft can be found at eel.is/c++draft/full
Don't open it! it is a html web page of 32 MB!

News:

- isocpp.org (Standard C++ Foundation)
- cpp.libhunt.com/newsletter/archive
- www.meetingcpp.com/blog/blogroll/

Main conferences:

- www.meetingcpp.com (slides)
- cppcon.org (slides)
- isocpp.com conference list

Coding exercises and other resources:

- www.hackerrank.com/domains/cpp
- leetcode.com/problemset/algorithms
- open.kattis.com
- cpppatterns.com

The Course

The Course: 9 Weeks on C++20/ C++23

- Week 1: Installation and hello world program
- Week 2: Flow Control and Basic Variable Types
- Week 3: Strings, Vectors and Arrays
- Week 4: Expressions and Operators
- Week 5: Statements
- Week 6: Functions
- Week 7: Classes
- Week 8: C++ Library
- Week 9: Project

“The only way to learn a new programming language is by writing programs in it”

Dennis Ritchie

Creator of the C programming language

Installation

Software Installs

- Install WSL on Windows
- Install VS Code
- Install C++ Compiler
- Configure VS Code

- Install WSL on Windows

- Windows Subsystem for Linux (WSL) is required for this course
- Please install it by following this page:
- <https://learn.microsoft.com/en-us/windows/wsl/install>
- Follow this page to set up a username and a password for your Linux account:
- <https://learn.microsoft.com/en-us/windows/wsl/setup/environment#set-up-your-linux-username-and-password>
- Write down your password – if you will need later when you run “sudo” command

- Install VS Code

- Download VS Code here: <https://code.visualstudio.com/>
- Follow the setup instruction:
- Windows: <https://code.visualstudio.com/docs/setup/windows>
- Mac: <https://code.visualstudio.com/docs/setup/mac>

What Compiler Should I Use?

Most popular compilers:

- Microsoft Visual Code (**MSVC**) is the compiler offered by Microsoft
- The GNU Compiler Collection (**GCC**) contains the most popular C++ Linux compiler
- **Clang** is a C++ compiler based on LLVM Infrastructure available for Linux/Windows/Apple (default) platforms

Suggested compiler on Linux for beginner: **Clang**

- Comparable performance with GCC/MSVC and low memory usage
- Expressive diagnostics (examples and propose corrections)
- Strict C++ compliance. GCC/MSVC compatibility (inverse direction is not ensured)
- Includes very useful tools: memory sanitizer, static code analyzer, automatic formatting, linter, etc.

Install the Compilers

Install the last gcc/g++ on WSL

```
$ sudo apt update  
$ sudo apt install gcc g++  
$ gcc --version  
$ g++ --version
```

Install the last clang on Mac

Run the following command from a terminal (iTerm):

```
$ xcode-select --install
```

The above command will install C++ compiler clang

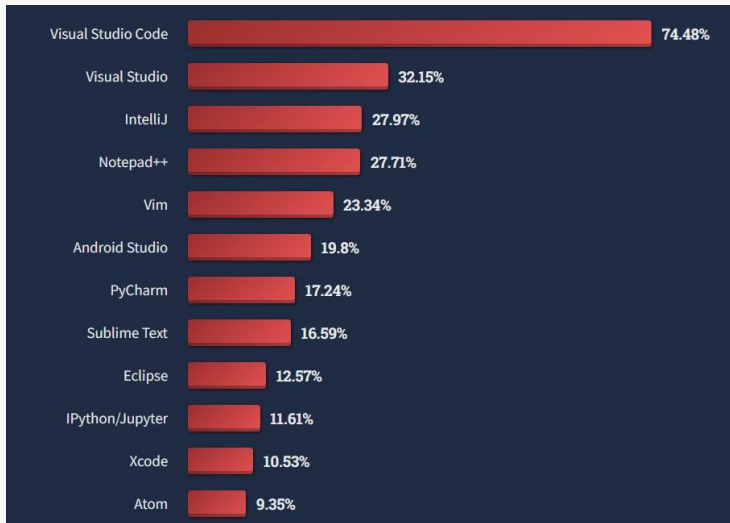
Commercial C++ IDE (Integrated Development Environment):

- **Microsoft Visual Studio** (MSVC). Most popular IDE for Windows
- **Clion**. Powerful IDE with a lot of options
- **QT-Creator**. Fast (written in C++), simple

Free C++ IDE:

- **Microsoft Visual Studio Code** (VS Code)
- XCode. Default on Mac OS
- Cevelop
- Eclipse

Not suggested: Notepad, Gedit, and other similar editors - lack of support for programming



How to Compile?

Compile C++11, C++14, C++17, C++20, C++23 programs:

```
g++ -std=c++11 <program.cpp> -o program
g++ -std=c++14 <program.cpp> -o program
g++ -std=c++17 <program.cpp> -o program
g++ -std=c++20 <program.cpp> -o program
g++ -std=c++23 <program.cpp> -o program
```

Any C++ standard is backward compatible

C++ is also backward compatible with C, even for very old code, except if it contains C++ keywords (new, template, class, typename, etc.)

We can potentially compile a pure C program in C++23

Compiler	C++11		C++14		C++17		C++20/C++ 23	
	Core	Library	Core	Library	Core	Library	Core	Library
g++	4.8.1	5.1	5.1	5.1	7.1	9.0	11+	11+
clang++	3.3	3.3	3.4	3.5	5.0	11.0	12+	14+
MSVC	19.0	19.0	19.10	19.0	19.15	19.15	19.29+	19.29

Hello World

C code with `printf` :

```
#include <stdio.h>

int main() {
    printf("Hello World!\n");
}
```

`printf`

prints on standard output

C++ code with `streams` :

```
#include <iostream>

int main() {
    std::cout << "Hello World!\n";
}
```

`cout`

represent the standard output stream

The previous example can be written with the global `std` namespace:

```
#include <iostream>
using namespace std;

int main() {
    cout << "Hello World!\n";
}
```

Note: For sake of space and for improving the readability, we intentionally omit the `std` namespace in the next slides

`std::cout` is an example of *output* stream. Data is redirected to a destination, in this case the destination is the standard output

C:

```
#include <stdio.h>
int main() {
    int    a    = 4;
    double b    = 3.0;
    char   c[] = "hello";
    printf("%d %f %s\n", a, b, c);
}
```

C++:

```
#include <iostream>
int main() {
    int    a    = 4;
    double b    = 3.0;
    char   c[] = "hello";
    std::cout << a << " " << b << " " << c << "\n";
}
```

- **Type-safe:** The type of object provided to the I/O stream is known statically by the compiler. In contrast, `printf` uses `%` fields to figure out the types dynamically
- **Less error prone:** With IO Stream, there are no redundant `%` tokens that have to be consistent with the actual objects pass to I/O stream. Removing redundancy removes a class of errors
- **Extensible:** The C++ IO Stream mechanism allows new user-defined types to be pass to I/O stream without breaking existing code
- **Comparable performance:** If used correctly may be faster than C I/O (`printf` , `scanf` , etc.) .

- Forget the number of parameters:

```
printf("long phrase %d long phrase %d", 3);
```

- Use the wrong format:

```
int a = 3;  
...many lines of code...  
printf(" %f", a);
```

- The `%c` conversion specifier does not automatically skip any leading white space:

```
scanf("%d", &var1);  
scanf(" %c", &var2);
```

- codewars.com
- Create a free account
- Add a Clan: CCS

