程序设计基础及语言 —— C++ 大学教程

第 1 章 计算机和 C++ 简介

邱 远

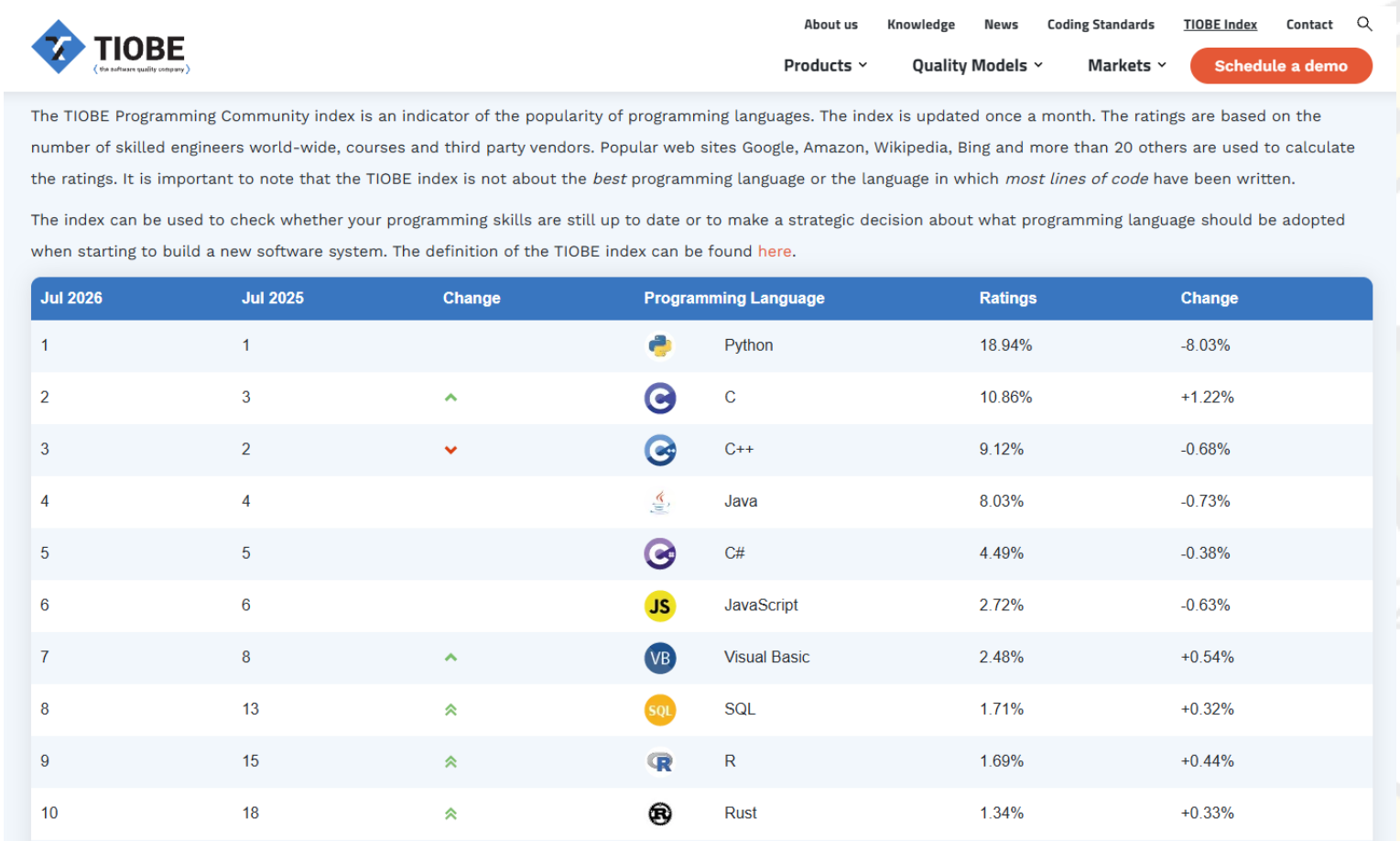
东南大学 网络空间安全学院
yuanqiu@seu.edu.cn

2026 年 9 月

目录

为什么要学 C++？

C++ 依然是极受欢迎的编程语言



TIOBE

About us | Knowledge | News | Coding Standards | TIOBE Index | Contact

Products | Quality Models | Markets | Schedule a demo

The TIOBE Programming Community index is an indicator of the popularity of programming languages. The index is updated once a month. The ratings are based on the number of skilled engineers world-wide, courses and third party vendors. Popular web sites Google, Amazon, Wikipedia, Bing and more than 20 others are used to calculate the ratings. It is important to note that the TIOBE index is not about the *best* programming language or the language in which *most lines of code* have been written.

The index can be used to check whether your programming skills are still up to date or to make a strategic decision about what programming language should be adopted when starting to build a new software system. The definition of the TIOBE index can be found here.

Jul 2026	Jul 2025	Change	Programming Language	Ratings	Change
1	1		Python	18.94%	-8.03%
2	3	↑	C	10.86%	+1.22%
3	2	↓	C++	9.12%	-0.68%
4	4		Java	8.03%	-0.73%
5	5		C#	4.49%	-0.38%
6	6		JavaScript	2.72%	-0.63%
7	8	↑	Visual Basic	2.48%	+0.54%
8	13	⇈	SQL	1.71%	+0.32%
9	15	⇈	R	1.69%	+0.44%
10	18	⇈	Rust	1.34%	+0.33%

图 1: 在 2026 年 7 月的 TIOBE 指数[1]中，C++ 排名第 3 [7]

[1] TIOBE 编程社区指数是编程语言流行度的指标，由搜索引擎对该语言的查询结果数统计。

C++ 用于构建深度学习框架

main pytorch / aten / src / ATen / native / LossCTC.cpp ↑ Top

Code Blame 595 lines (534 loc) · 27.8 KB Raw

```cpp
// This kernel is a relatively straightforward implementation of the alpha calculation in the forward backward algorithm (section 4.1).
// A (minor) twist is that we are using log-calculations to enhance numerical stability (log_probs and log_alpha).
// The function returns the loss and the alphas, the alphas are kept for the backward step. The wrapper (ctc_loss below) hides
// the alphas from the user by only returning the loss.
template<typename scalar_t, ScalarType target_scalar_type>
std::tuple<Tensor, Tensor> ctc_loss_cpu_template(const Tensor& log_probs, const Tensor& targets, IntArrayRef input_lengths, IntArrayRef target_lengths, int64_t BLANK) {
  TORCH_CHECK(log_probs.numel() > 0, "log_probs tensor must not be empty");
  // log_probs: input_len x batch_size x num_labels
  // targets [int64]: batch_size x target_length OR sum(target_lengths)
  constexpr scalar_t neginf = -std::numeric_limits<scalar_t>::infinity();
  using target_t = typename std::conditional_t<target_scalar_type == kInt, int, int64_t>;

  Tensor neg_log_likelihood, log_alpha;
  size_t tg_target_stride;
  std::vector<int64_t> tg_batch_offsets;

  if (targets.scalar_type() == kLong) {
    std::tie(neg_log_likelihood, log_alpha,tg_target_stride, tg_batch_offsets) =
        ctc_loss_allocate_outputs<scalar_t, kLong>(
            log_probs, targets, input_lengths, target_lengths, BLANK);
  } else {
    std::tie(neg_log_likelihood, log_alpha, tg_target_stride, tg_batch_offsets) =
        ctc_loss_allocate_outputs<scalar_t, kInt>(
            log_probs, targets, input_lengths, target_lengths, BLANK);
  }

  int64_t batch_size = log_probs.size(1);
  auto lpp  = log_probs.permute({1,0,2});
  auto log_probs_a_global = lpp.accessor<const scalar_t, 3>();
  auto log_alpha_a_global = log_alpha.accessor<scalar_t, 3>();
  auto targets_data = targets.const_data_ptr<target_t>();
  auto neg_log_likelihood_a = neg_log_likelihood.accessor<scalar_t, 1>();
```

图 2: PyTorch 仓库 [5] 中的 C++ 代码

使用 C++ 的公司


IBM


intel


Google

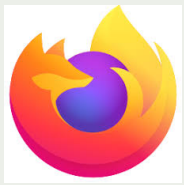

ByteDance
字节跳动

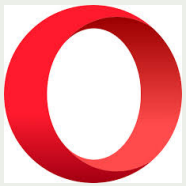

Alibaba Group
阿里巴巴集团


Tencent
腾讯

用 C++ 开发的程序

浏览器

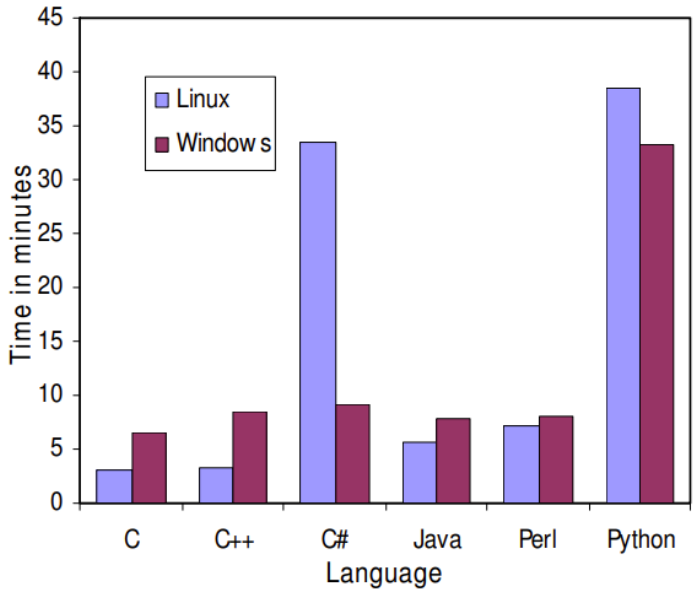


软件

游戏

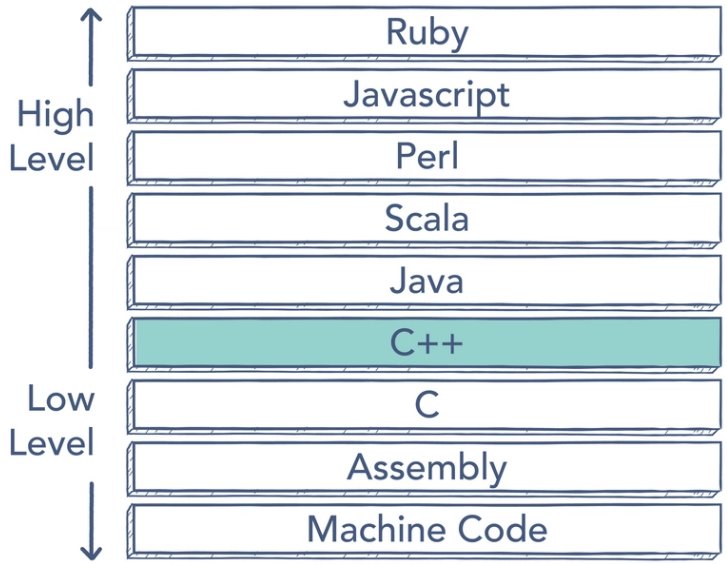
为什么选择 C++?

更快



同一程序用不同语言实现，采用 C/C++ 的版本执行效率更高 [2]。

更底层



C++ 能访问内存地址与进行位运算等硬件底层操作，代码质量更高。

目录

教学大纲

- 课程代码：BJSL0022 (57S261, JS126C)
- 课程名称：程序设计基础及语言（I+II）
- 本学期学时：48 讲课学时 +32 实验学时
  - 讲课时间：16周 × 3节/周，星期四，3-5 节 (9:50-12:15)
  - 上机时间：11周 × 3节/周，星期四，6-8 节 (14:00-16:35)
- 课程目标：学习语言 → 掌握方法 → 提高动手实践能力
  - 支撑数据结构、算法、操作系统等后续课程
- 成绩构成：平时考核（40%）+ 期末考核（60%）
  - 签到与课堂表现 (10%)
  - 上机实验 (20%)：2，4，6，8，9，10，11，12，13，14，16 周
  - 过程性评价 (10%)：笔试
  - 期末考试 (60%)

$$\begin{aligned}\text{期末考试总分}(100) &= \text{笔试}(40) + \text{机试}(60)\\ &= \text{代码阅读}(20) + \text{代码填空}(20) + \text{2-4 题应用编程}(60)\end{aligned}$$

主要参考书

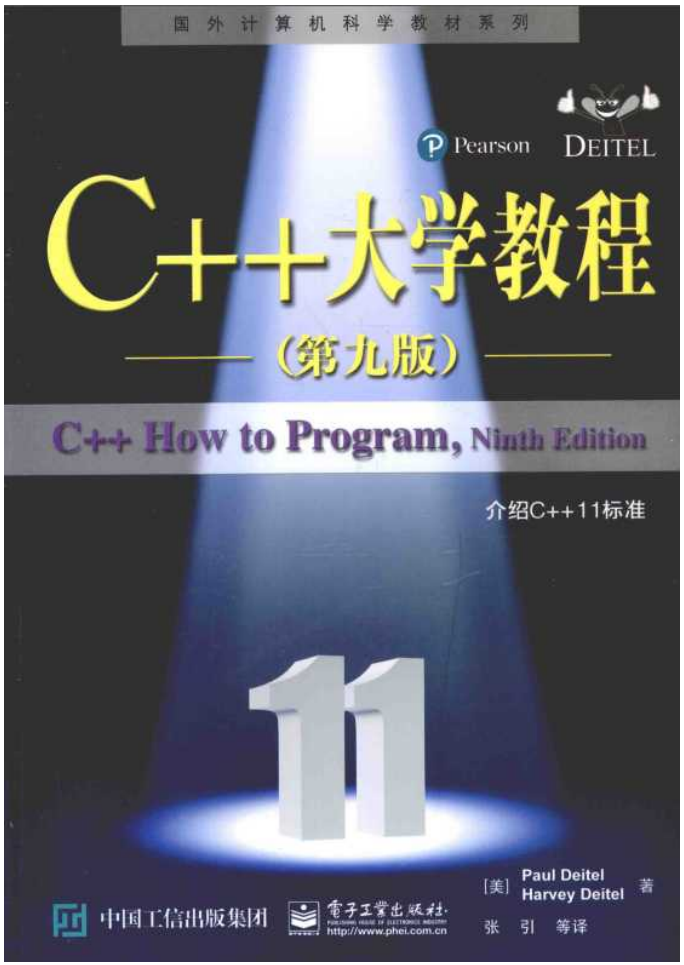


图 3: C++ 大学教程（第九版）[1]

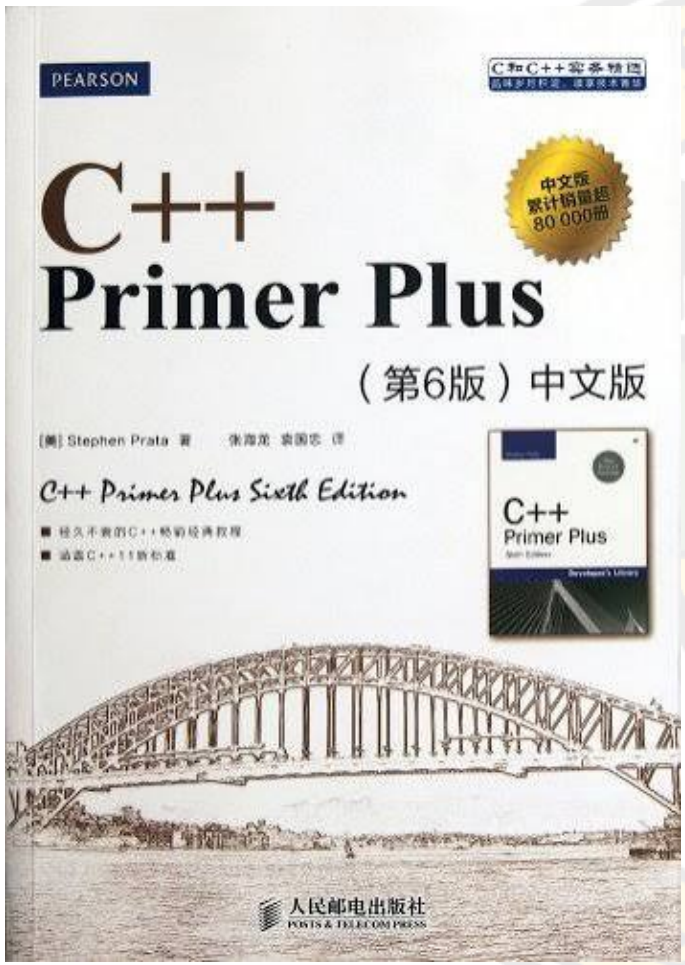


图 4: C++ Primer Plus [4]

授课老师

- 邱远，网络空间安全学院，东南大学青年首席教授
- 邮箱：yuanqiu@seu.edu.cn
- 课程主页：https://QiuTedYuan.github.io/cpp.html
- 办公室：九龙湖校区李文正图书馆（北门）434 室
- 相关经历
  - 初中开始学习编程（Pascal 语言），曾获 NOI 竞赛普及组一等奖
  - 本、硕、博毕业于香港科技大学，曾在阿里数据库（Java）团队实习
  - 2022 年 7 月-2023 年 9 月于字节跳动进行 C++ 相关开发工作[2]

[2] https://github.com/ByConity/ByConity/tree/master/src/Advisor

目录

计算机简史

- 理论探索

1936 年	Alan Turing	图灵机 [9]
1945 年	John von Neumann	冯 · 诺依曼架构 [10]
1948 年	Claude Shannon	信息论 [6]
1950 年	Alan Turing	图灵测试、人工智能 [8]

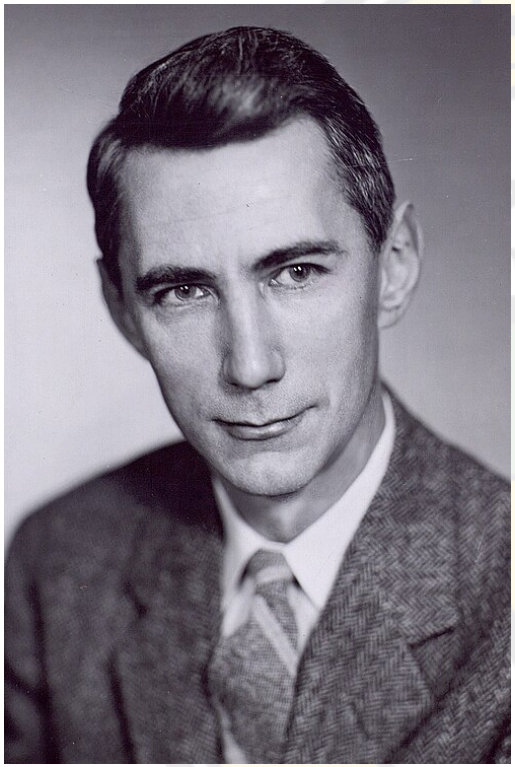
- 硬件发展

1946 年	第一代：电子管	ENIAC
1950 年代	第二代：晶体管	IBM 7090
1964 年	第三代：集成电路	IBM 360
1971 年	第四代：微处理器	Intel 4004
21 世纪	多核、GPU 等	NVIDIA

计算机名人

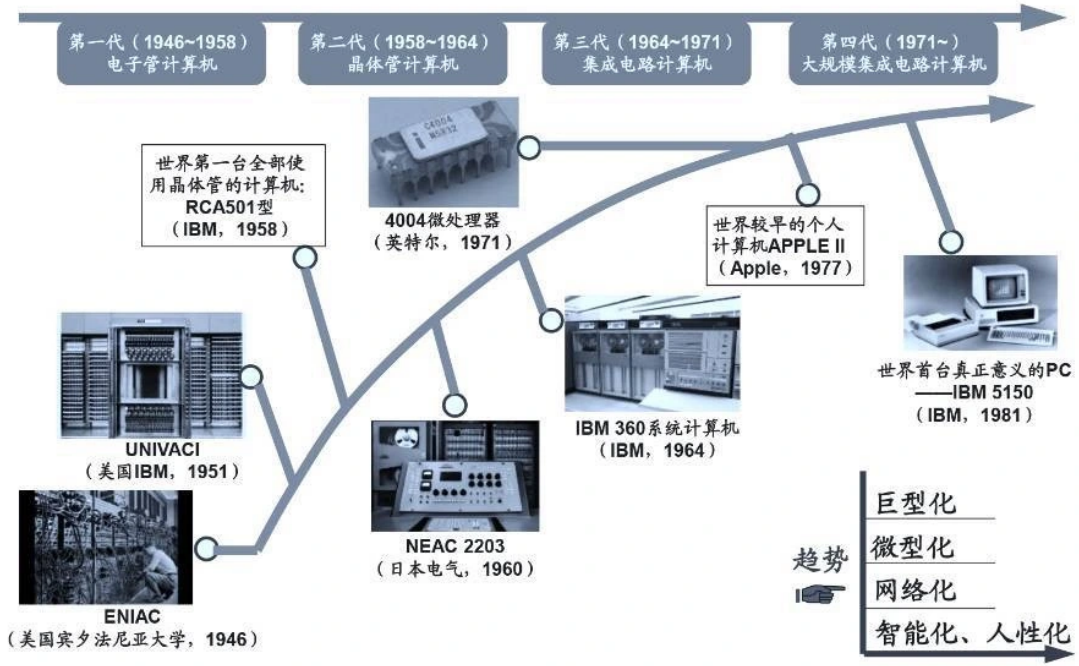
(a) 艾伦 · 图灵 (1912-1954)
计算机科学之父、人工智能之父
1936 年提出图灵机
1950 年提出图灵测试
ACM 于 1966 年设立图灵奖

(b) 约翰 · 冯 · 诺依曼 (1903-1957)
现代计算机之父
1945 年提出冯诺依曼架构



(c) 克劳德 · 艾尔伍德 · 香农 (1916-2001)
信息论之父
1948 年创立信息论

计算机硬件发展



摩尔定律 (Moore's law [3])

每 18 个月，集成电路上可容纳的晶体管数目翻倍（或同样价格的计算机性能翻倍，或同样性能的处理器价格对半）。

计算机组成

- 输入 (Input unit)：键盘、鼠标、触屏、U 盘、...
- 输出 (Output unit)：显示器、打印机、扬声器、机械臂、...
- 内存 (Memory unit)：快速存取，断电即丢
- 二级存储 (Secondary storage unit)：速度较慢，永久保存
- 中央处理器 (CPU)：控制器 + 运算器（算数逻辑单元 + 寄存器）
- 图形处理器 (GPU)：擅长矩阵和向量运算

二进制编码

十进制

- 每一位取值为 0-9
- 逢 10 进 1
- 表示方法

$$(114)_{10} = 1\times10^2+1\times10^1+4\times10^0$$

- 十进制转二进制：不断对 2 求余数，将余数倒读

$$\begin{aligned}(14)_{10} &= 2\times7+0\\ &= 2\times(2\times3+1)+0\\ &= 2\times(2\times(2\times1+1)+1)+0\\ &= (1110)_2\end{aligned}$$

二进制

- 每一位取值为 0 或 1
- 逢 2 进 1
- 表示方法

$$(101)_2 = 1\times2^2+0\times2^1+1\times2^0$$

- 二进制转十进制：直接计算

$$(101)_2 = 4+0+1 = 5$$

- 二进制转十六进制：每 4 位二进制对应 1 位十六进制

$$(1001\,1100)_2 = 0\text{x}9\text{C}$$

计算机中的信息存储

- 比特 (bit)：信息储存的最小单位，值为 0 或 1。也称位或二进制位 (binary digit)
- 字节 (Byte)：1 Byte = 8 bits，约定俗成的基本计量单位

b_4	b_3	b_2	b_1	Row ↓ / Column →	0	1	2	3	4	5	6	7
					$0\,0\,0$	$0\,0\,1$	$0\,1\,0$	$0\,1\,1$	$1\,0\,0$	$1\,0\,1$	$1\,1\,0$	$1\,1\,1$
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	¦
1	1	0	1	13	CR	GS	—	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

- 字 (Word)：寄存器大小，通常为 32 位（4 字节）或 64 位（8 字节）

	千	兆	吉	太
十进制 (SI)	kilo, 1 kB $= 10^3$ B	mega/MB	giga/GB	tera/TB
二进制 (微软)	kibi, 1 KiB $= 1024$ B	mebi/MiB	gibi/GiB	tebi/TiB

表 1: 倍数存储单位

目录

一段 C++ 代码

这是一段 C++ 的 Hello world 代码

```
#include <iostream>

int main() {
  std::cout << "Hello, world!" << std::endl;
  return 0;
}
```

又一段 C++ 代码

这也是一段同样功能的代码，但同时也是 C 的代码！

```
#include "stdio.h"
#include "stdlib.h"

int main(int argc, char **argv) {
  printf("%s", "Hello, world!\n"); // a C function!
  return EXIT_SUCCESS;
}
```

又一段 C++ 代码……吗？

理论上，这也是一段 C++ 的代码（编译需要-arch x86_64)

```
#include "stdio.h"
#include "stdlib.h"

int main(int argc, char **argv) {
  asm( "sub $0x20,%rsp\n\t" // assembly code
    "movabs $0x77202c6f6c6c6548,%rax\n\t"
    "mov %rax,(%rsp)\n\t"
    "movl $0x646c726f, 0x8(%rsp)\n\t"
    "movw $0x0a21, 0xc(%rsp)\n\t"
    "movb $0x0,0xe(%rsp)\n\t"
    "leaq (%rsp),%rax\n\t"
    "mov %rax,%rdi\n\t"
    "call _printf\n\t"
    "add $0x20, %rsp\n\t"
  );
  return EXIT_SUCCESS;
}
```

机器语言、汇编语言和高级语言

机器语言 (Machine Language)

计算机能够直接识别的指令，通常由 0 和 1 组成。

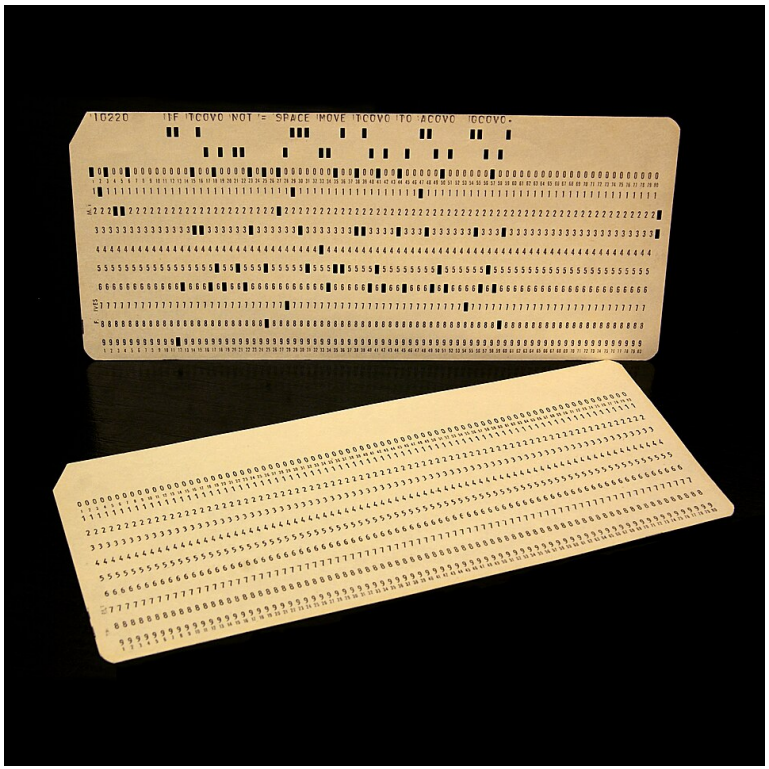


图 6: 机器语言编程用的打孔卡 (Punched card)，现进化为答题卡

机器语言、汇编语言和高级语言

汇编语言 (Assembly Language)

使用英语简写的机器语言指令，需要通过汇编器 (assembler) 汇编后才能转换为可执行的计算机代码。

图 7: 模拟汇编语言的小游戏《程序员升职记》[3]

图 8: 使用汇编语言进行代码注入[4]

[3] https://store.steampowered.com/app/375820/Human_Resource_Machine/
[4] https://www.bilibili.com/video/BV1Ks411z7KS/

机器语言、汇编语言和高级语言

高级语言 (High-level Language)

高度封装的程序语言，让程序员用日常用语 + 数学符号进行编程。

1956 年，第一个高级语言 Fortran

```
PROGRAM sum
INTEGER :: x, y
READ(*,*) x, y
PRINT *, 'Sum is ', x+y
END PROGRAM sum
```

1960 年，源于数学的语言 Lisp

```
(define input (read))
(define x (car input))
(define y (car (cdr input)))
(display (+ x y))
```

机器无法理解，需要编译器 (Compiler) 将高级语言编译为机器语言[5]。更“高级”的语言（如 Python、Scala）用解释器 (Interpreter) 代替编译器，消除了编译延迟，但代码执行更慢。

[5] https://godbolt.org/

机器语言、汇编语言和高级语言

表 2: 机器语言、汇编语言、高级语言对比

	机器语言	汇编语言	高级语言
机器理解	✓直接	—需要汇编	×需要编译（耗时）
人类阅读	×无法阅读	—可阅读难理解	✓接近自然语言
代码长度	×极长	×极长	✓高度集成
程序控制	✓完全	✓完全	—越底层，控制越强
执行效率	✓优秀	✓优秀	—越底层，效率越高
可移植性	×机器限定	×架构限定	✓可移植性好

第四代？超高级？语言

Vibe Coding

写个scala的快排

以下是Scala实现的快速排序，包含多种风格：

1. 函数式风格（最简洁）

scala 复制 下载

```scala
def quickSort(list: List[Int]): List[Int] = list match {
  case Nil => Nil
  case pivot :: tail =>
    val (left, right) = tail.partition(_ < pivot)
    quickSort(left) ::: pivot :: quickSort(right)
}
```

C++ 的历史

- 1972 年，Dennis Ritchie[6]发明 C 语言
  - 高速、简单、跨平台
- 1979 年，Bjarne Stroustrup[7]发明 C with Classes
  - 从面向过程的编程，进化为面向对象的编程
  - 开发大型程序更友好，且具有高级特性
- 1983 年，更名为 C++
- 1998 年，C++98
- 2003 年，C++03
- 2011 年，C++11（本课程采用）
- 2014 年，C++14（`auto` `func`, `deprecated`, `exchange`, ...）
- 2017 年，C++17（`inline`, `optional`, `string_view`, ...）
- 2020 年，C++20（`typename`, `range`, `nodiscard`, ...）
- 2023 年，C++23（`consteval`, `throw`, `print`, ...）
- 202? 年，C++26（`contracts`, `execution`, `_`, ...）

[6] https://www.nokia.com/bell-labs/about/dennis-m-ritchie/

[7] https://www.stroustrup.com/

C++ 设计理念

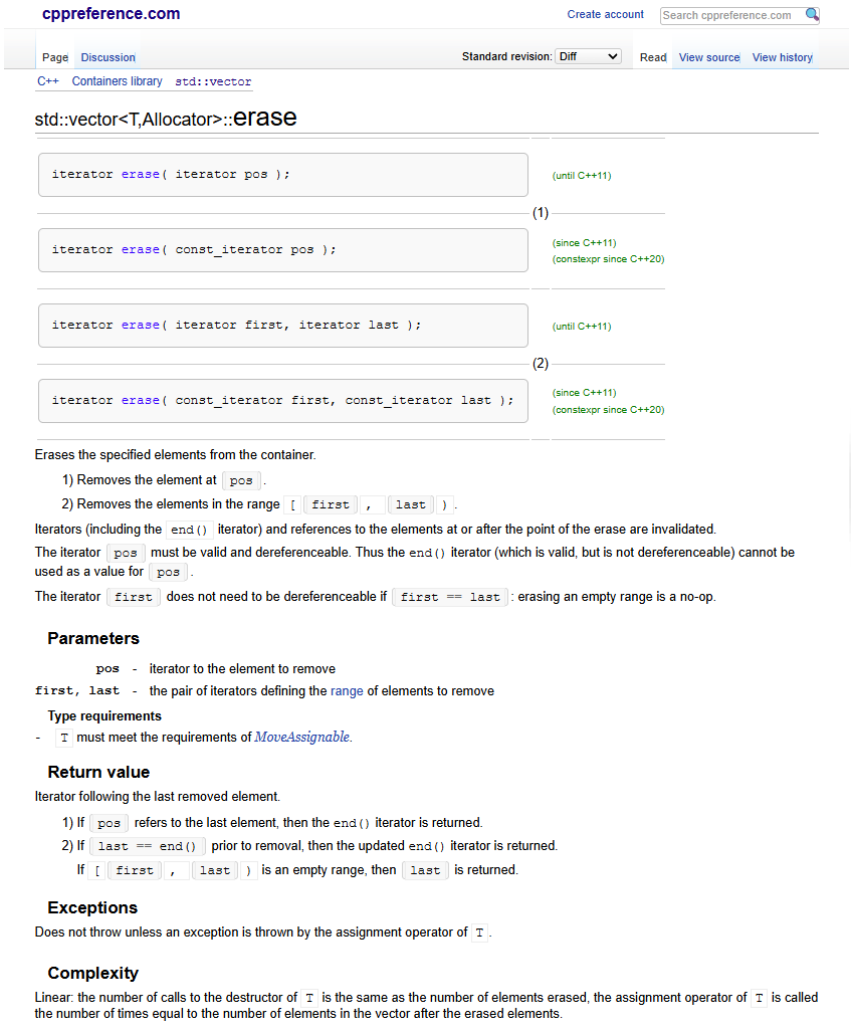
- 给予程序员更多选择（能力越大，责任越大）
  - 例如：C++ 允许程序员直接操纵内存
- 在编译时捕获错误（代价是编译报错更繁琐）
  - 例如：C++ 是强类型语言
- 鼓励将复杂代码隔离
  - 例如：命名空间、智能指针

表 3: 不同语言的特色

高级语言	主要特征
C	接近硬件，效率极高
C++	工具齐全，向后兼容，可上可下
Java	JVM、中央库、风格统一（所有人写出来的 Java 程序都一样）
JavaScript	灵活性（你觉得一个东西该怎么写，它就是这么写）
Python	社区极其丰富
Scala	函数式编程

C++ 有极为新手友好的注释

https://cppreference.com/



cppreference.com

Create account | Search cppreference.com

Page | Discussion | Standard revision: Diff | Read | View source | View history

C++ Containers library std::vector

std::vector<T,Allocator>::erase

```
iterator erase( iterator pos );
```
(until C++11)

(1)

```
iterator erase( const_iterator pos );
```
(since C++11)
(constexpr since C++20)

```
iterator erase( iterator first, iterator last );
```
(until C++11)

(2)

```
iterator erase( const_iterator first, const_iterator last );
```
(since C++11)
(constexpr since C++20)

Erases the specified elements from the container.

1) Removes the element at `pos`.

2) Removes the elements in the range `[ first , last )`.

Iterators (including the `end()` iterator) and references to the elements at or after the point of the erase are invalidated.

The iterator `pos` must be valid and dereferenceable. Thus the `end()` iterator (which is valid, but is not dereferenceable) cannot be used as a value for `pos`.

The iterator `first` does not need to be dereferenceable if `first == last`: erasing an empty range is a no-op.

Parameters

`pos` - iterator to the element to remove

`first, last` - the pair of iterators defining the range of elements to remove

Type requirements

- `T` must meet the requirements of *MoveAssignable*.

Return value

Iterator following the last removed element.

1) If `pos` refers to the last element, then the `end()` iterator is returned.

2) If `last == end()` prior to removal, then the updated `end()` iterator is returned.

If `[ first , last )` is an empty range, then `last` is returned.

Exceptions

Does not throw unless an exception is thrown by the assignment operator of `T`.

Complexity

Linear: the number of calls to the destructor of `T` is the same as the number of elements erased, the assignment operator of `T` is called the number of times equal to the number of elements in the vector after the erased elements.

目录

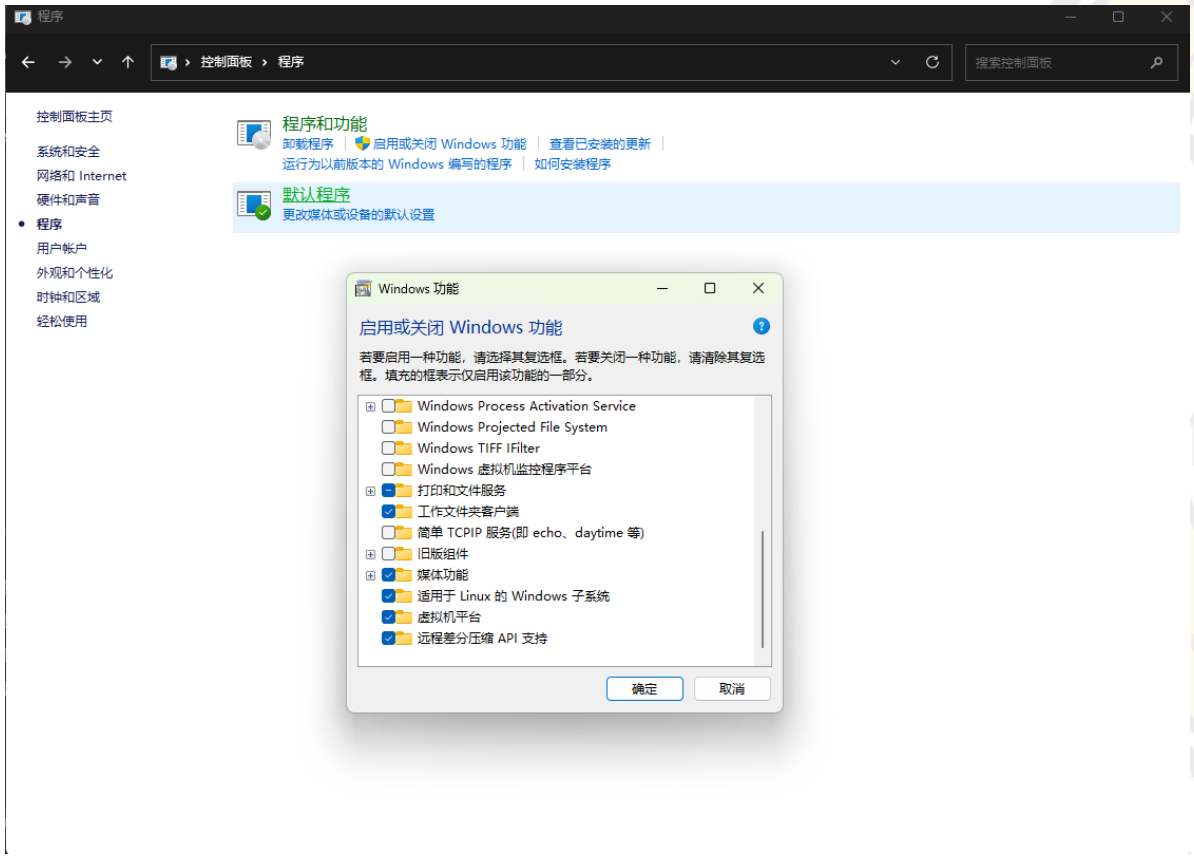
试着跑一个 C++ 程序

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>

int main() { // main.cpp
    std::cout << "Hello, world!" << std::endl;
    std::srand(std::time(0));
    int number = std::rand() % 101;
    std::cout << "Guess a random number between 0 and 100 (inclusive)" << std::endl;

    int guess = -1;
    while (guess != number) {
        std::cout << "Your guess: ";
        std::cin >> guess;
        if (guess == number) {
            std::cout << "Your got it! The number is exactly " << number << std::endl;
        } else if (guess > number) {
            std::cout << "Your guess is too high" << std::endl;
        } else {
            std::cout << "Your guess is too low" << std::endl;
        }
    }
    return 0;
}
```

如果你使用 Windows 10+

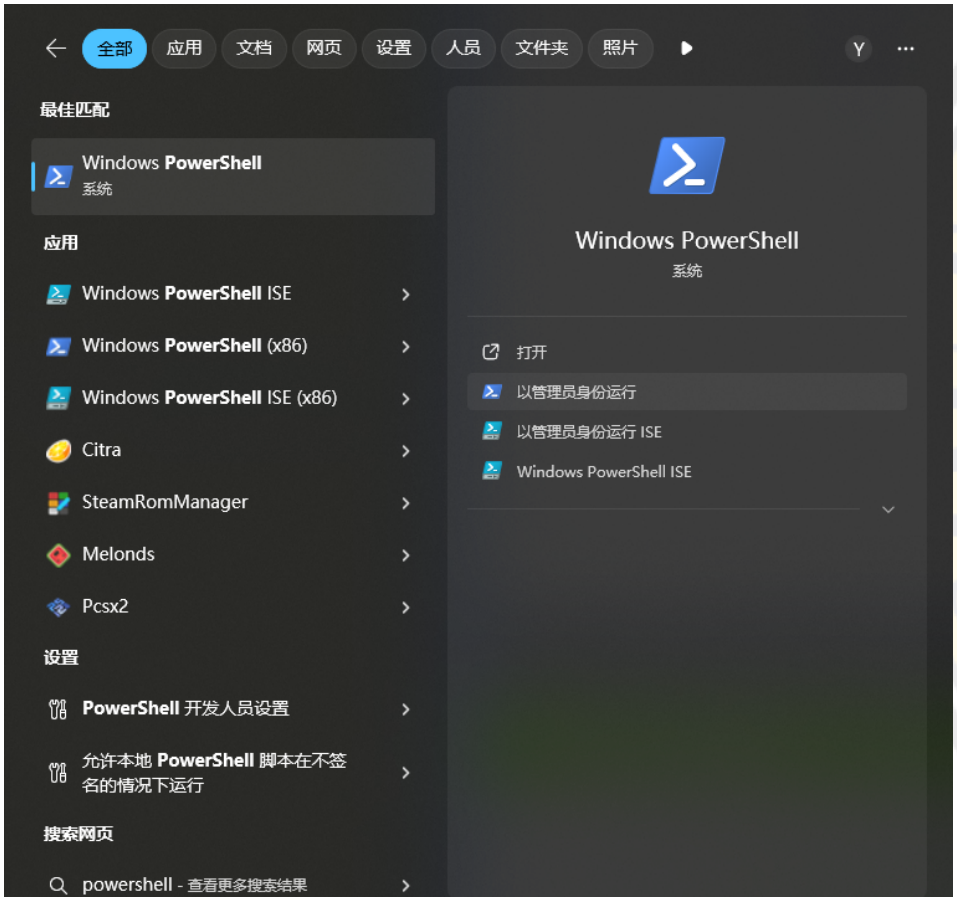
推荐使用 WSL(Windows Subsystem for Linux)

- 控制面板 > 程序 > 启用或关闭 Windows 功能，确认打开“适用于 Linux 的 Windows 子系统”



如果你使用 Windows 10+

- 以管理员身份运行 PowerShell，并执行`wsl --install`



安装 G++ 编译器

如果你使用 Linux 或 Windows 10+

- 打开命令行 (Windows 需要在标签栏中选择 Ubuntu)
- 在/etc/apt/sources.list 文件中添加清华源

```
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-updates main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-updates main restricted universe multiverse
deb https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-backports main restricted universe multiverse
# deb-src https://mirrors.tuna.tsinghua.edu.cn/ubuntu/ jammy-backports main restricted universe multiverse
```

- 安装 g++:

```
sudo apt-get update && sudo apt-get install g++
```

如果你使用 MacOS

- 打开命令行并安装 Homebrew[8]:

```
/bin/zsh -c "$(curl -fsSL https://gitee.com/cunkai/HomebrewCN/raw/master/Homebrew.sh)"
```

- 安装 g++:

```
brew install g++
```

[8] https://brew.sh/

编译并执行 C++ 代码

方案一：命令行

使用任意文本编辑器将代码保存为`test.cpp`文件，用`g++ test.cpp`进行编译。通过`./a.out`执行编译后的程序。

~/Documents/cpp_course/ch1/test.cpp

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime>
int main() { // guess.cpp
  std::cout << "Hello, world!" << std::endl;
  srand(time(0));
  int number = rand() % 101;
  std::cout << "Guess a random number between 0 and 100 (inclusive)" << std::endl;
  int guess = -1;
  while (guess != number) {
    std::cout << "Your guess: ";
    std::cin >> guess;
    if (guess == number) {
      std::cout << "Your got it! The number is exactly " << number << std::endl;
    } else if (guess > number) {
      std::cout << "Your guess is too high" << std::endl;
    } else {
      std::cout << "Your guess is too low" << std::endl;
    }
  }
  return 0;
}
```

U:**- test.cpp All (21,11) [Compiling] (C++//l Flymake[0 0] Proj[-] ElDoc Abbr

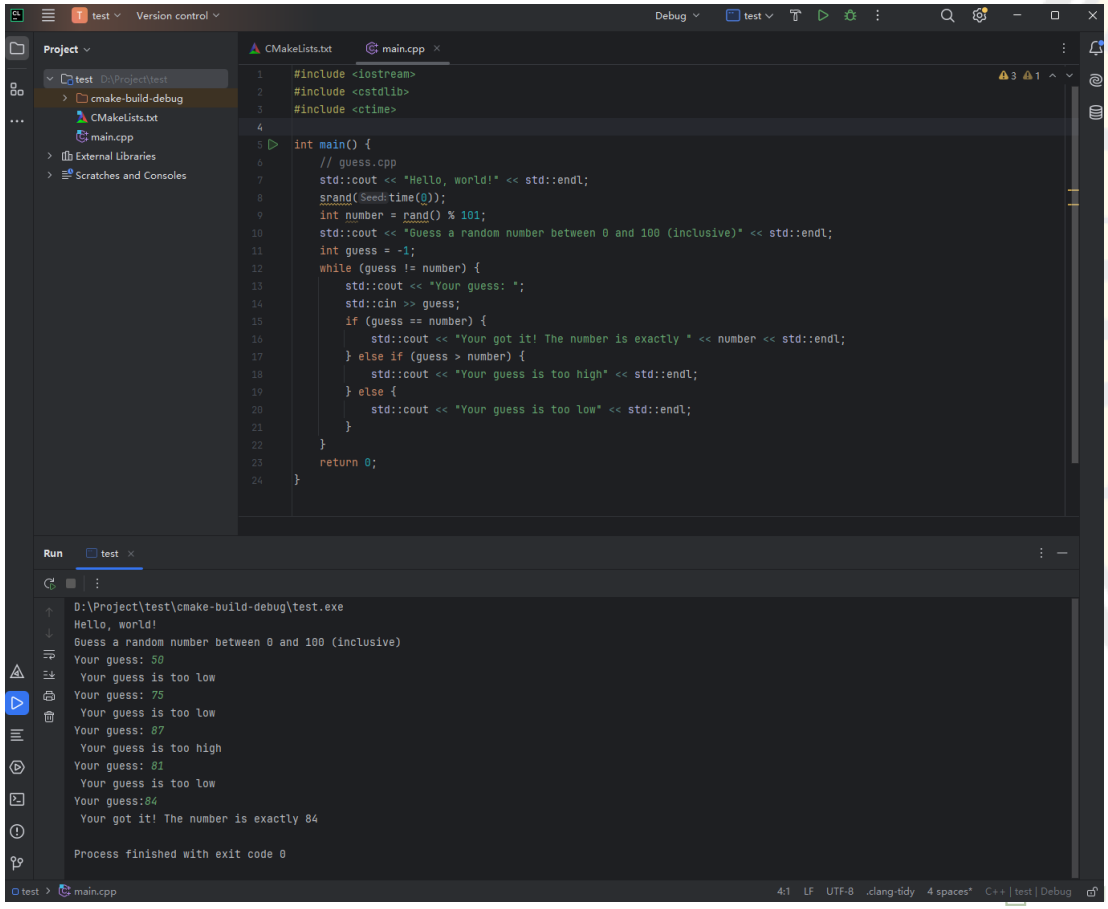
```
Welcome to the Emacs shell

~/Documents/cpp_course/ch1 $ g++ test.cpp
~/Documents/cpp_course/ch1 $ ./a.out
Hello, world!
Guess a random number between 0 and 100 (inclusive)
Your guess: 50
Your guess is too low
Your guess: 75
Your guess is too low
Your guess: 87
Your guess is too low
Your guess: 94
Your guess is too high
Your guess: 90
Your guess is too low
Your guess: 92
Your got it! The number is exactly 92
~/Documents/cpp_course/ch1 $
```

-- *eshell* All (19,29) [Compiling] (Eshell Proj[-])

编译并执行 C++ 代码

方案二：集成开发环境 (IDE)

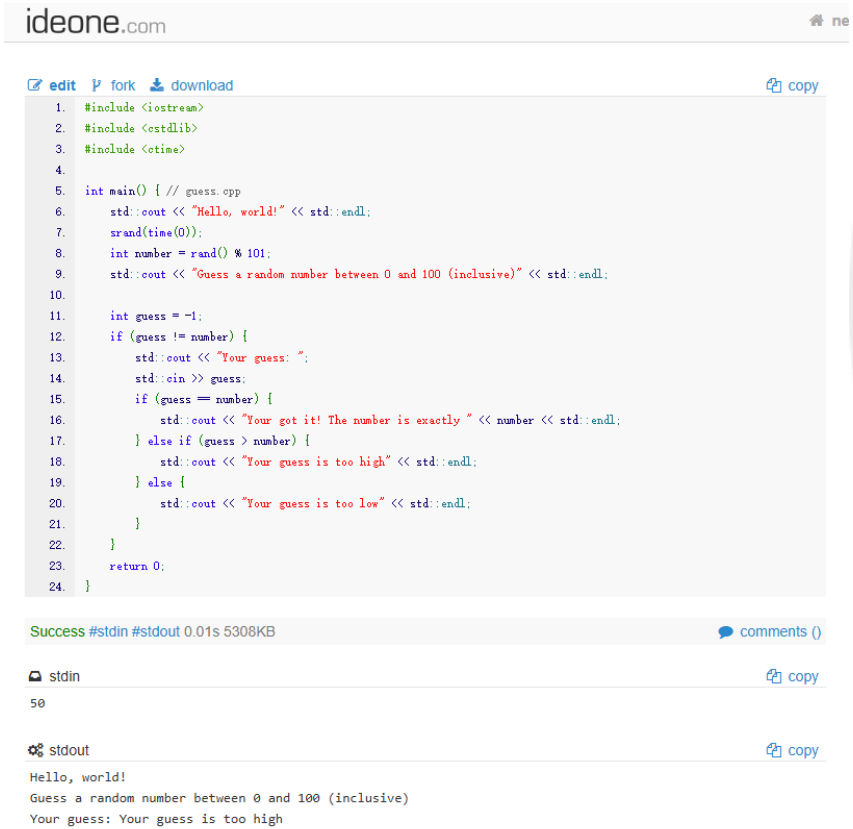
使用 VS Code/Clion 等 IDE 新建项目，将代码复制进`main.cpp`文件后，点击`main`函数或右上角处的运行。



编译并执行 C++ 代码

方案三：在线 IDE（不支持交互式代码）

将代码填入在线 IDE[9]，并编译执行。



[9] 例如https://www.ideone.com/

更优雅的编译：Makefile

当要编译的文件很多时，可以在目录下写一个 Makefile，这样可以直接通过make命令进行编译。目前而言，只需要用如下的 Makefile 即可

```
#Makefile
all: main

main: main.o
    g++ $^ -o $@

%.o: %.cpp
    g++ -c $< -o $@

clean:
    rm -rf *.o main
```

也可使用 CMake，格式如下

```
#CMakeLists.txt
cmake_minimum_required(VERSION 3.29)
project(test)

set(CMAKE_CXX_STANDARD 11)

add_executable(main main.cpp)
```

C++ 编译流程

1. 预处理 (源代码 → 源代码) `g++ -E main.cpp -o main.i`

```
extern __attribute__((__visibility__("default"))) wistream wcin;
extern __attribute__((__visibility__("default"))) wostream wcout;
extern __attribute__((__visibility__("default"))) wostream wcerr;
extern __attribute__((__visibility__("default"))) wostream wclog;


} }
# 2 "main.cpp" 2

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

2. 编译 (源代码 → 汇编码) `g++ -S main.i -o main.s`

```
_main:                                  ; @main
    .cfi_startproc
; %bb.0:
    stp x29, x30, [sp, #-16]!           ; 16-byte Folded Spill
    mov x29, sp
    .cfi_def_cfa w29, 16
    .cfi_offset w30, -8
    .cfi_offset w29, -16
    adrp x0, __ZNSt3__14coutE@GOTPAGE
    ldr  x0, [x0, __ZNSt3__14coutE@GOTPAGEOFF]
    adrp x1, l_.str@PAGE
    add  x1, x1, l_.str@PAGEOFF
```

3. 汇编 (汇编码 → 机器码) `g++ -c main.s -o main.o`
4. 链接 (机器码 → 可执行程序) `g++ main.o`

总结

- 理解计算机科学的相关基本概念✓
  - 计算机简史、二进制编码、比特与字节、...
- 了解计算的基本原理✓
  - 计算机器的基本结构、编程语言及其进化、...
- 理解一个典型的 C++ 程序开发环境✓

参考文献 I

[1] Paul Deitel and Harvey Deitel. *C++ 大学教程（第九版）*. Trans. by 张引 et al. 电子工业出版社, 2016. ISBN: 9787121290015.

[2] Mathieu Fourment and Michael R Gillings. "A comparison of common programming languages used in bioinformatics". In: *BMC bioinformatics* 9.1 (2008), p. 82.

[3] Gordon E Moore et al. *Cramming more components onto integrated circuits*. 1965.

[4] Stephen Prata. *C++ Primer Plus*. Trans. by 张海龙 and 袁国忠. 人民邮电出版社, 2012. ISBN: 9787115279460.

[5] *PyTorch*. 2026. URL: `https://github.com/pytorch/pytorch` (visited on 07/14/2026).

[6] Claude Elwood Shannon. "A mathematical theory of communication". In: *The Bell system technical journal* 27.3 (1948), pp. 379–423.

[7] TIOBE Software. *TIOBE Index for July 2026*. 2026. URL: `https://www.tiobe.com/tiobe-index/` (visited on 07/14/2026).

参考文献 II

[8] Alan M Turing. “Computing machinery and intelligence (1950)”. In: *Mind* 59.236 (1987), pp. 33–60.

[9] Alan M Turing et al. “On computable numbers, with an application to the Entscheidungsproblem”. In: *J. of Math* 58.345-363 (1936), p. 5.

[10] John Von Neumann. “First Draft of a Report on the EDVAC”. In: *IEEE Annals of the History of Computing* 15.4 (1993), pp. 27–75.