CS144：计算机网络导论

2025 年秋季

实验检查点 0：网络热身

截止时间：9 月 28 日（周日），晚上 11:59

欢迎来到 CS144：计算机网络导论。在这个热身练习中，你将在自己的电脑上安装 GNU/Linux 系统，学习如何手动完成一些网络任务，用 C++ 编写一个获取网页的小程序，并在内存中实现网络的一个关键抽象：写入方和读取方之间的可靠字节流。我们预计这个热身练习需要你 2 到 6 小时完成（后续的检查点会占用更多时间）。关于实验作业有三点需要快速说明：

- 在开始之前先通读整个文档是个好主意！
- 在这个分为 8 部分的实验作业中，你将逐步构建自己对互联网重要组成部分的实现——一个路由器、一个网络接口和 TCP 协议（它将不可靠的数据报转换为可靠的字节流）。大多数周的任务都建立在之前的工作之上，也就是说，你在整个学期中逐步构建自己的实现，并且在未来的几周中将继续使用你的代码。这使得很难"跳过"某个检查点。
- 如果你不满足 CS144 的先修课程要求，请不要选修这门课——我们的教学资源有限。请将检查点 0 和 1 作为一个衡量标准：如果你在前两个检查点的编程中感到吃力，请考虑在积累了更多编程经验后再选修 CS144（也许可以在修完 CS 106L、开始一个自主编程项目或以其他方式提高你的编程能力之后）。
- 实验文档不是"规范"——也就是说它们不是用来单向阅读的。它们的详细程度更接近于软件工程师从老板或客户那里得到的需求描述。我们希望你通过参加实验课并在发现模糊之处时提出澄清性问题来从中受益。我们会在课程网站上更新"实验常见问题"文档，以回答需要澄清的后续问题。

0 合作政策

编程作业必须是你自己的作品：你必须编写你提交的所有编程作业代码，除了我们作为作业一部分提供给你的代码。请勿从 Stack Overflow、GitHub、ChatGPT 或其他来源复制粘贴代码。如果你基于在网上或其他地方找到的示例编写自己的代码，请在提交的源代码中以注释引用 URL。

与他人合作：你不能向任何人展示你的代码，查看任何人的代码（无论是人类还是 AI 编写的），或查看往年的解答。你可以与其他学生讨论作业，但不要复制任何人的代码。如果你与另一位同学讨论了作业，请在提交的源代码中以注释注明他们的名字。更多详情请参阅课程管理讲义，如有疑问请在 EdStem 上提问。像 GitHub Copilot 或 ChatGPT 这样的服务应被视为等同于"往年修过 CS144 的学生"。

`EdStem`：请随时在 EdStem 上提问，但请勿发布任何源代码。

1 在你的电脑上设置 GNU/Linux

CS144 的作业需要 GNU/Linux 操作系统和支持 C++ 2023 标准的最新 C++ 编译器。请选择以下选项之一：

1. 推荐：安装 CS144 VirtualBox 虚拟机镜像（说明见 https://stanford.edu/class/cs144/vm_howto/vm-howto-image.html）。
2. 使用 Google Cloud 虚拟机（说明见 https://stanford.edu/class/cs144/vm_howto）。我们已向 Google 申请了 50 美元的优惠券，收到后将分发。

3. 运行 Ubuntu 25.04 版本。

4. "自担风险"使用其他 GNU/Linux 发行版，但请注意你可能会遇到障碍，并且需要有能力自行调试。你的代码将在 Ubuntu 25.04 LTS 上使用 g++ 14.2.0 进行测试，必须在这些条件下正确编译和运行。

5. 如果你使用的是 2020–24 年的 MacBook（搭载 ARM64 M 系列芯片），VirtualBox 将无法正常运行。请安装 UTM 虚拟机软件和我们的 ARM64 虚拟机镜像，详见 https://stanford.edu/class/cs144/vm_howto/。

2 安装所需软件包

以下命令将安装运行课程软件所需的 Ubuntu（Debian）软件包：

```
sudo apt update && sudo apt install git cmake gdb build-essential clang \
    clang-tidy clang-format gcc-doc pkg-config glibc-doc tcpdump tshark
```

3 手动使用网络

让我们开始使用网络。你将手动完成两个任务：获取网页（就像 Web 浏览器一样）和发送电子邮件（就像电子邮件客户端一样）。这两个任务都依赖于一种称为可靠双向字节流的网络抽象：你将在终端中输入一系列字节，相同的字节序列最终将以相同的顺序被传递到另一台计算机上运行的程序（服务器）。服务器以自己的字节序列进行响应，传回你的终端。

3.1 获取网页

1. 在 Web 浏览器中访问 http://cs144.keithw.org/hello 并观察结果。

2. 现在，你将手动完成浏览器所做的相同事情。

   (a) 在你的虚拟机上（或在你自己的电脑上——例如 macOS 上的终端程序），运行 `telnet cs144.keithw.org http`。这告诉 telnet 程序在你的计算机和另一台计算机（名为 cs144.keithw.org）之间打开一个可靠的字节流，并连接到该计算机上运行的特定服务："http"服务，即超文本传输协议，万维网所使用的协议。[1]

   如果你的计算机已正确设置并连接到互联网，你将看到：

```
user@computer:~$ telnet cs144.keithw.org http
Trying 104.196.238.229...
Connected to cs144.keithw.org.
Escape character is '^]'.
```

   如果需要退出，按住 `ctrl` 并按 `]`，然后输入 `close`。

   (b) 输入 `GET /hello HTTP/1.1`。这告诉服务器 URL 的路径部分。（从第三个斜杠开始的部分。）

   (c) 输入 `Host: cs144.keithw.org`。这告诉服务器 URL 的主机部分。（`http://` 和第三个斜杠之间的部分。）

   (d) 输入 `Connection: close`。这告诉服务器你已经完成请求，它应该在回复完毕后立即关闭连接。

   (e) 再按一次 Enter 键。这会发送一个空行，告诉服务器你的 HTTP 请求已经完成。

   (f) 如果一切顺利，你将看到与浏览器相同的响应，前面还有 HTTP 头部，告诉浏览器如何解释该响应。

3. 作业：现在你知道如何手动获取网页了，向我们展示你能做到！使用上述技术获取 URL http://cs144.keithw.org/lab0/`sunetid`，将 `sunetid` 替换为你自己的主 SUNet ID。你将在

`X-Your-Code-Is`：头部收到一个密钥。保存你的 SUNet ID 和密钥，以便写入实验报告。

[1] 计算机的名称有一个数字等价物（104.196.238.229，一个 IPv4 地址），服务的名称也是如此（80，一个 TCP 端口号）。我们稍后会详细讨论这些。

3.2 给自己发一封电子邮件

既然你知道如何获取网页，现在是时候发送电子邮件了，同样使用可靠字节流连接到另一台计算机上运行的服务。

1. SSH 登录 `sunetid@cardinal.stanford.edu`（确保你在斯坦福的网络上），然后运行 `telnet 67.231.149.169 smtp`[2]。"smtp"服务是指简单邮件传输协议，用于发送电子邮件。如果一切顺利，你将看到：

```
user@computer:~$ telnet 67.231.149.169 smtp
Trying 67.231.149.169...
Connected to 67.231.149.169.
Escape character is '^]'.
220 mx0a-00000d07.pphosted.com ESMTP mfa-m0342459
```

2. 第一步：向邮件服务器标识你的计算机。输入 `HELO mycomputer.stanford.edu`。等待看到类似 "250 ... Hello cardinal3.stanford.edu [171.67.24.75], pleased to meet you" 的内容。

3. 下一步：谁在发送电子邮件？输入 `MAIL FROM: sunetid@stanford.edu`。将 `sunetid` 替换为你的 SUNet ID[3]。如果一切顺利，你将看到 "250 2.1.0 Sender ok"。

4. 接下来：收件人是谁？先试着给自己发一封邮件。输入 `RCPT TO: sunetid@stanford.edu`。将 `sunetid` 替换为你自己的 SUNet ID。如果一切顺利，你将看到 "250 2.1.5 Recipient ok."。

5. 现在是上传邮件内容的时候了。输入 `DATA` 告诉服务器你已准备好开始。如果一切顺利，你将看到 "354 End data with <CR><LF>.<CR><LF>"。

6. 现在你正在给自己写一封电子邮件。首先，输入你在邮件客户端中会看到的邮件头部。在头部末尾留一个空行。

```
354 End data with <CR><LF>.<CR><LF>
From: sunetid@stanford.edu
To: sunetid@stanford.edu
Subject: Hello from CS144 Lab 0!
```

7. 输入邮件正文——随便写什么都行。完成后，在单独一行输入一个句号：`.`。预计会看到类似："250 2.0.0 33h24dpdsr-1 Message accepted for delivery"。

8. 输入 `QUIT` 结束与邮件服务器的会话。检查你的收件箱和垃圾邮件文件夹，确保你收到了这封邮件。

[2] 这些说明也可能在斯坦福网络之外有效，但我们无法保证。

[3] 是的，可以提供虚假的"发件人"地址。电子邮件有点像邮局的真实邮件——回复地址的准确性（基本上）靠自觉。你可以在明信片上随意写回复地址，电子邮件也大致如此。请勿滥用此功能——说真的。工程知识带来责任！使用虚假"发件人"地址发送电子邮件通常由垃圾邮件发送者和犯罪分子所为，以便冒充他人。玩玩这个假装是 santaclaus@northpole.gov 很有趣，但请确保你没有欺骗任何收件人。而且：即使收件人知道这是个玩笑，也不要冒充任何斯坦福员工发送电子邮件（否则你可能触发大学的 IT 安全警报）。

3.3 监听与连接

你已经看到了 telnet 能做什么：一个向其他计算机上运行的程序发起出站连接的客户端程序。现在是时候尝试做一个简单的服务器了：那种等待客户端来连接的程序。

1. 在一个终端窗口中，在你的虚拟机上运行 `netcat -v -l -p 9090`。你应该看到：

```
user@computer:~$ netcat -v -l -p 9090
Listening on [0.0.0.0] (family 0, port 9090)
```

2. 让 netcat 保持运行。在另一个终端窗口中，运行 `telnet localhost 9090`（也在你的虚拟机上）。

3. 如果一切顺利，netcat 将打印类似 "Connection from localhost 53500 received!" 的内容。

4. 现在尝试在任一终端窗口中输入——netcat（服务器）或 telnet（客户端）。注意你在一个窗口中输入的任何内容都会出现在另一个窗口中，反之亦然。你需要按回车键才能传输字节。

5. 在 netcat 窗口中，按 `ctrl-C` 退出程序。注意 telnet 程序也会立即退出。

4 使用操作系统流套接字编写网络程序

在这个热身实验的下一部分中，你将编写一个通过互联网获取网页的短程序。你将利用 Linux 内核（以及大多数其他操作系统）提供的一项功能：在两个程序之间创建可靠的双向字节流——一个运行在你的计算机上，另一个运行在互联网另一端的计算机上（例如 Apache 或 nginx 等 Web 服务器，或 netcat 程序）。

此功能称为流套接字。对你的程序和 Web 服务器来说，套接字看起来就像一个普通的文件描述符（类似于磁盘上的文件，或 stdin/stdout I/O 流）。当两个流套接字连接时，写入一个套接字的任何字节最终将以相同的顺序从另一台计算机上的另一个套接字中出来。

然而，实际上互联网并不提供可靠的字节流服务。相反，互联网真正做的只是"尽最大努力"将短数据片段（称为互联网数据报）传递到目的地。每个数据报包含一些元数据（头部），指定源地址和目的地地址等信息——它来自哪台计算机，以及它要去哪台计算机——以及一些有效载荷数据（最多约 1,500 字节）要传递到目的地计算机。

虽然网络会尝试传递每个数据报，但在实践中数据报可能会 (1) 丢失、(2) 乱序到达、(3) 内容被篡改，甚至 (4) 被重复传递多次。通常由连接两端的操作系统将"尽最大努力的数据报"（互联网提供的抽象）转化为"可靠的字节流"（应用程序通常想要的抽象）。

两台计算机必须合作以确保流中的每个字节最终都能按正确顺序传递到另一端的流套接字。它们还必须告诉对方自己准备接受多少数据，并确保发送的数据不超过对方愿意接受的量。所有这些都使用 1981 年制定的协议来完成，称为传输控制协议（TCP）。

在这个实验中，你将简单地使用操作系统现有的 TCP 支持。你将编写一个名为"webget"的程序，它创建一个 TCP 流套接字，连接到 Web 服务器并获取页面——就像你在本实验前面手动做的那样。在未来的实验中，你将通过自己实现传输控制协议来实现这个抽象的另一端，从不太可靠的数据报中创建可靠的字节流。

4.1 开始——在虚拟机和 GitHub 上设置代码仓库

1. 实验作业将使用一个名为"Minnow"的初始代码库。在你的虚拟机上，运行 `git clone https://github.com/cs144/minnow` 获取实验的源代码。

2. 输入 `cd minnow` 进入 minnow 目录。

3. 在 Web 浏览器中，你将在自己的 GitHub 账户中创建一个仓库来存放你的实验解答。

   (a) 如果你还没有 GitHub 账户，请在 https://github.com 创建一个。

   (b) 访问 https://github.com/new 创建新仓库。

   (c) 在你的 GitHub 账户中将仓库命名为"minnow"。

   (d) 确保将仓库设置为"Private"（私有），以便你的解答不会公开。

   (e) 点击"Create Repository"。

   (f) 在下一个页面上，点击"Invite collaborators"，然后点击"Add people"。

   (g) 添加"cs144-grader"作为协作者（这将允许我们查看和评分你的代码，同时保持其私密性）。

4. 回到你的虚拟机，运行命令 `git remote add github https://github.com/■■■/minnow`（将"用户名"替换为你的实际 GitHub 用户名）将 GitHub

仓库注册为目标。这将在你的本地实验代码副本（在虚拟机上）和 GitHub 上的副本之间建立关联（你将用它来备份本地副本并接受评分）。

5. 运行 `git push github` 将初始代码发送到你的 GitHub 仓库。如果一切顺利，你将看到几行文本输出，以 `* [new branch] main -> main` 结尾。如果看到错误消息，请仔细检查你是否正确执行了上述步骤。

4.2 编译初始代码

1. 仍在"minnow"目录中，创建一个目录来编译实验软件：`cmake -S . -B build`

2. 编译源代码：`cmake --build build`

3. 使用你喜欢的文本编辑器（许多学生喜欢通过 SSH 使用 VS Code 编辑文件，但你可以使用任何你喜欢的）：打开并编辑 `writeups/check0.md` 文件。这是你的实验报告模板，将包含在你的提交中。

4.3 现代 C++：大部分安全但仍然快速和底层

CS144 是一门编程密集的课程。实验作业使用当代 C++ 风格完成，利用最新的（2023 年）特性来尽可能安全地编程。这可能与你以前被要求编写 C++ 的方式不同。有关此风格的参考，请参见 C++ Core Guidelines（http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines）。

基本思想是确保每个对象都被设计为具有尽可能小的公共接口，具有大量内部安全检查且难以被误用，并且知道如何自行清理。我们希望避免"配对"操作（例如 malloc/free 或 new/delete），因为配对的后半部分可能不会发生（例如，如果函数提前返回或抛出异常）。相反，操作在对象的构造函数中发生，相反的操作在析构函数中发生。这种风格称为"资源获取即初始化"，即 RAII。

具体来说，我们希望你：

- 使用 https://en.cppreference.com 上的语言文档作为资源。（我们建议你避免使用 cplusplus.com，因为它更可能过时。）
- 永远不要使用 `malloc()` 或 `free()`。
- 永远不要使用 `new` 或 `delete`。
- 基本上不要使用原始指针（`*`），只在必要时使用"智能"指针（`unique_ptr` 或 `shared_ptr`）。（在 CS144 中你不需要使用这些。）
- 避免使用模板、线程、锁和虚函数。（在 CS144 中你不需要使用这些。）
- 避免 C 风格字符串（`char *str`）或字符串函数（`strlen()`、`strcpy()`）。这些非常容易出错。请改用 `std::string`。
- 不要使用 C 风格的类型转换（例如 `(FILE *)x`）。如果必须的话使用 C++ 的 `static_cast`（在 CS144 中你通常不需要这个）。
- 优先使用 const 引用传递函数参数（例如：`const Address & address`）。
- 除非变量需要被修改，否则将每个变量都设为 `const`。
- 除非方法需要修改对象，否则将每个方法都设为 `const`。
- 避免使用全局变量，并给每个变量尽可能小的作用域。
- 在提交作业之前，运行 `cmake --build build --target tidy` 获取关于改善 C++ 编程实践的建议，并运行 `cmake --build build --target format` 来统一代码格式。

关于 Git 的使用：实验以 Git（版本控制）仓库的形式分发——一种记录更改、检查点版本以帮助调试和跟踪源代码来源的方式。请在工作过程中频繁做小的提交，并使用标识更改内容和原因的提交信息。理想的情况是每次提交都应该能编译通过，并稳步让越来越多的测试通过。做小的"语义"提交有助于调试（如果每次提交都能编译通过

且信息描述了提交所做的一件明确的事情，调试会容易得多），并通过记录你的持续进展来防止作弊指控——这是一项在任何包含软件开发的职业中都有用的技能。评分者将阅读你的提交信息以了解你是如何开发实验解答的。如果你还没学会如何使用 Git，请在 CS144 答疑时间寻求帮助或参考教程（例如 https://guides.github.com/introduction/git-handbook）。最后，虽然我们要求你使用 GitHub 上的私有仓库备份和提交代码，请确保你的代码不被公开访问。

再次强调（因为我们以前教过这门课）：在工作过程中频繁做小的提交，并使用标识更改内容和原因的提交信息。

4.4 阅读 Minnow 支持代码

为了支持这种编程风格，Minnow 的类将操作系统函数（可以从 C 调用的函数）封装在"现代"C++ 中。我们为你提供了 C++ 封装，涵盖了希望你从 CS 111 中大致熟悉的概念，特别是套接字和文件描述符。

请阅读 `util/socket.hh` 和 `util/file_descriptor.hh` 文件中的公共接口（文件中 "public:" 之后的部分）。（请注意 Socket 是一种 FileDescriptor，而 TCPSocket 是一种 Socket。）

4.5 编写 webget

是时候实现 webget 了，一个使用操作系统的 TCP 支持和流套接字抽象通过互联网获取网页的程序——就像你在本实验前面手动做的那样。

1. 从 build 目录，在文本编辑器或 IDE 中打开 `../apps/webget.cc` 文件。

2. 在 `get_URL` 函数中，按照此文件中描述的简单 Web 客户端进行实现，使用你之前使用的 HTTP（Web）请求格式。使用 `TCPSocket` 和 `Address` 类。

3. 提示：

• 请注意在 HTTP 中，每行必须以 `\r\n` 结尾（仅使用 `\n` 或 `endl` 是不够的）。

• 不要忘记在客户端请求中包含 `Connection: close` 行。这告诉服务器不应该等待你的客户端在此之后发送更多请求。相反，服务器将发送一个回复然后立即结束其出站字节流（从服务器的套接字到你的套接字的那个）。你会发现你的入站字节流已经结束，因为当你读取完来自服务器的整个字节流时，你的套接字将到达"EOF"（文件末尾）。这就是你的客户端知道服务器已完成回复的方式。

• 确保读取并打印来自服务器的所有输出，直到套接字到达"EOF"（文件末尾）——单次调用 `read` 是不够的。

• 我们预计你大约需要写八行代码。

4. 运行 `cmake --build build` 编译你的程序。如果看到错误消息，你需要在继续之前修复它。

5. 运行 `./apps/webget cs144.keithw.org /hello` 测试你的程序。这与在 Web 浏览器中访问 http://cs144.keithw.org/hello 相比如何？与第 3.1 节的结果相比如何？随意实验——用你喜欢的任何 http URL 进行测试！

6. 当它似乎正常工作后，运行 `cmake --build build --target check_webget` 来运行自动化测试。在实现 `get_URL` 函数之前，你应该看到以下内容：

```
$ cmake --build build --target check_webget
Test project /home/cs144/minnow/build
    Start 1: compile with bug-checkers
1/2 Test #1: compile with bug-checkers ........ Passed 1.02 sec
    Start 2: t_webget
2/2 Test #2: t_webget .......................***Failed 0.01 sec
DEBUG: Function called: get_URL( "cs144.keithw.org", "/nph-hasher/xyzzy" )
DEBUG: get_URL() function not yet implemented
ERROR: webget returned output that did not match the test's expectations
```

完成作业后，你将看到：

```
$ cmake --build build --target check_webget
Test project /home/cs144/minnow/build
    Start 1: compile with bug-checkers
1/2 Test #1: compile with bug-checkers ........ Passed 1.09 sec
    Start 2: t_webget
2/2 Test #2: t_webget ......................... Passed 0.72 sec
100% tests passed, 0 tests failed out of 2
```

7. 评分者将使用与 `make check_webget` 不同的主机名和路径来运行你的 webget 程序——所以请确保它不是只能在单元测试使用的主机名和路径上工作。

5 内存中的可靠字节流

到现在为止，你已经看到了可靠字节流的抽象在互联网通信中是多么有用，即使互联网本身只提供"尽最大努力"（不可靠）的数据报服务。

为了完成本周的实验，你将在单台计算机的内存中实现一个提供此抽象的对象。（你可能在 CS 110/111 中做过类似的事情。）字节在"输入"端写入，可以按相同顺序从"输出"端读取。字节流是有限的：写入方可以结束输入，之后不能再写入字节。当读取方读到流的末尾时，它将到达"EOF"（文件末尾），不能再读取字节。

你的字节流还将进行流量控制以限制其在任何给定时间的内存消耗。对象在初始化时有一个特定的"容量"：它在任何给定时刻愿意在自己内存中存储的最大字节数。字节流将限制写入方在某一时刻可以写入的量，以确保流不超过其存储容量。随着读取方读取并消耗字节，写入方被允许写入更多。你的字节流用于单线程——你不必担心并发写入方/读取方、锁或竞态条件。

要明确的是：字节流是有限的，但在写入方结束输入之前它可以几乎任意地长[4]。你的实现必须能够处理远超容量的流。容量限制的是某一时刻内存中持有的字节数（已写入但尚未读取），但不限制流的长度。一个容量仅为一字节的对象仍然可以承载数 TB 长的流，只要写入方每次写入一个字节，且读取方在写入方被允许写入下一字节之前读取每个字节。

以下是写入方的接口：

```
void push( std::string data );         // Push data to stream, but only as much as available capacity a
void close();                          // Signal that the stream has reached its ending. Nothing more w
bool is_closed() const;                // Has the stream been closed?
uint64_t available_capacity() const;   // How many bytes can be pushed to the stream right now?
uint64_t bytes_pushed() const;         // Total number of bytes cumulatively pushed to the stream
```

以下是读取方的接口：

```
std::string_view peek() const;         // Peek at the next bytes in the buffer
void pop( uint64_t len );              // Remove `len` bytes from the buffer
bool is_finished() const;              // Is the stream finished (closed and fully popped)?
bool has_error() const;                // Has the stream had an error?
uint64_t bytes_buffered() const;       // Number of bytes currently buffered (pushed and not popped)
uint64_t bytes_popped() const;         // Total number of bytes cumulatively popped from stream
```

请打开 `src/byte_stream.hh` 和 `src/byte_stream.cc` 文件，实现一个提供此接口的对象。在开发字节流实现时，你可以运行 `cmake --build build --target check0` 来进行自动化测试。

如果所有测试通过，check0 测试将运行你的实现的速度基准测试。对于所测试的三种 pop 长度，任何快于 0.1 Gbit/s（即每秒 1 亿比特）的速度对于本课程的目的都是可接受的。（实现有可能快于 10 Gbit/s，但这取决于你计算机的速度，不作要求。）

如有任何最新问题，请查看课程网站上的实验常见问题，或在实验课上（或 EdStem 上）向你的同学或教学人员询问。

[4] 至少长达 2^{64} 字节，在本课程中我们将此视为基本上任意长。

接下来是什么？在接下来的四周里，你将实现一个系统来提供相同的接口，不再在内存中，而是在不可靠的网络上。这就是传输控制协议——它的实现可以说是世界上最普遍的计算机程序。

6 提交

1. 在你的提交中，请仅修改 `webget.cc` 和 `src` 顶层的源代码（`byte_stream.hh` 和 `byte_stream.cc`）。请勿修改任何测试或 `util` 中的辅助代码。

2. 记住在编码过程中做小的提交，并写好提交信息。提交后，通过运行 `git push github` 经常将虚拟机的仓库备份到你的私有 GitHub 仓库。你的代码需要被提交并推送到 GitHub 才能被评分。

3. 在提交任何作业之前，请按顺序运行以下命令：

   (a) 确保你已将所有更改提交到 Git 仓库。你可以运行 `git status` 确保没有未提交的更改。记住：在编码过程中做小的提交。

   (b) `cmake --build build --target format`（统一代码风格）

   (c) `cmake --build build --target check0`（确保自动化测试通过）

   (d) 可选：`cmake --build build --target tidy`（建议改进以遵循良好的 C++ 编程实践）

4. 完成编辑 `writeups/check0.md`，填写此作业花费的小时数和任何其他备注。

5. 确保你的代码已提交并推送到你的私有 GitHub 仓库（`git push github`）。

6. Gradescope 上将有一个作业，截止时间为周日晚 11:59，供你提交你的 commit ID。

7. 如有任何问题，请尽快在周三实验课上告知课程团队，或在 EdStem 上发帖提问。祝你好运，欢迎来到 CS144！