Lab Checkpoint 5: down the stack (the network interface)

Due: Sunday, Nov. 9, 11:59 p.m.

Collaboration Policy: Same as checkpoint 0. Please do not look at other students' code or solutions to past versions of these assignments. Please fully disclose any collaborators or any gray areas in your writeup—disclosure is the best policy.

0 Overview

In this week's checkpoint, you'll go down the stack and implement a network interface: the bridge between Internet datagrams that travel the world, and link-layer Ethernet frames that travel one hop. This component can fit "underneath" your TCP/IP implementation from the earlier labs, but it will also be used in a different setting: when you build a router in checkpoint 6, it will route datagrams *between* network interfaces. Figure 1 shows how the network interface fits into both settings.

In past labs, you wrote a TCP implementation that can exchange **TCP segments** with any other computer that speaks TCP. How are these segments actually conveyed to the peer's TCP implementation? As we've discussed, there are a few options:

- **TCP-in-UDP-in-IP.** The TCP segments can be carried in the payload of a user datagram. When working in a normal (user-space) setting, this is the easiest to implement: Linux provides an interface (a "datagram socket", `UDPSocket`) that lets applications supply *only the payload* of a user datagram and the target address, and the kernel takes care of constructing the UDP header, IP header, and Ethernet header, then sending the packet to the appropriate next hop. The kernel makes sure that each socket has an exclusive combination of local and remote addresses and port numbers, and since the kernel is the one writing these into the UDP and IP headers, it can guarantee isolation between different applications.

- **TCP-in-IP.** In common usage, TCP segments are almost always placed directly inside an Internet datagram, without a UDP header between the IP and TCP headers. This is what people mean by "TCP/IP." This is a little more difficult to implement. Linux provides an interface, called a TUN device, that lets application supply an *entire* Internet datagram, and the kernel takes care of the rest (writing the Ethernet header, and actually sending via the physical Ethernet card, etc.). But now the application has to construct the full IP header itself, not just the payload.

- **TCP-in-IP-in-Ethernet.** In the above approach, we're still relying on the Linux kernel for part of the networking stack. Each time your code writes an IP datagram to the TUN device, Linux has to construct an appropriate link-layer (Ethernet) frame with the IP datagram as its payload. This means Linux has to figure out the next hop's Ethernet destination address, given the IP address of the next hop. If it doesn't know

实验检查点5：向下层（网络接口）

截止日期：11月9日，晚上11:59

合作政策：与检查点0相同。请不要查看其他学生的代码或这些作业过去版本的解决方案。请在您的报告中充分披露任何合作者或任何灰色地带——披露是最好的政策。

0 概述

在本周的检查点中，你将深入栈层并实现一个网络接口：这是在全世界旅行的互联网数据报和在一个跳段内旅行的链路层以太网帧之间的桥梁。这个组件可以嵌入早期实验中的TCP/IP实现“下方”，但它也将用于不同的环境：当你在检查点6中构建路由器时，它将在网络接口之间路由数据报。图1显示了网络接口如何适用于这两种环境。

在之前的实验中，你编写了一个TCP实现，它能够与任何会说TCP的计算机交换TCP段。这些段实际上是如何传递到对端的TCP实现的？正如我们所讨论的，有几个选项：

• TCP在UDP在IP中。TCP段可以承载在用户数据报的有效载荷中。在正常（用户空间）环境中，这是最容易实现的：Linux提供了一个接口（一个“数据报套接字”，`UDPSocket`），它允许应用程序仅提供用户数据报的有效载荷和目标地址，而内核负责构建UDP报头、IP报头和以太网报头，然后将数据包发送到适当的下一跳。内核确保每个套接字都有一个本地和远程地址以及端口号的独占组合，并且由于内核是将这些写入UDP和IP报头中的，因此它可以保证不同应用程序之间的隔离。

• TCP-in-IP。在常见用法中，TCP片段几乎总是直接放置在互联网数据报内部，IP和TCP头之间没有UDP头。这就是人们所说的“TCP/IP”。这稍微难实现一些。Linux提供了一个接口，称为TUN设备，它允许应用程序提供整个互联网数据报，而内核负责其余部分（写入以太网头，以及实际通过物理以太网卡发送等）。但现在应用程序必须自己构建完整的IP头，而不仅仅是有效载荷。

• TCP-IP-以太网。在上述方法中，我们仍然依赖Linux内核来处理部分网络栈。每次你的代码将一个IP数据报写入TUN设备时，Linux都必须构造一个合适的链路层（以太网）帧，并将IP数据报作为其有效载荷。这意味着Linux必须根据下一跳的IP地址来确定下一跳的以太网目标地址。如果它不知道

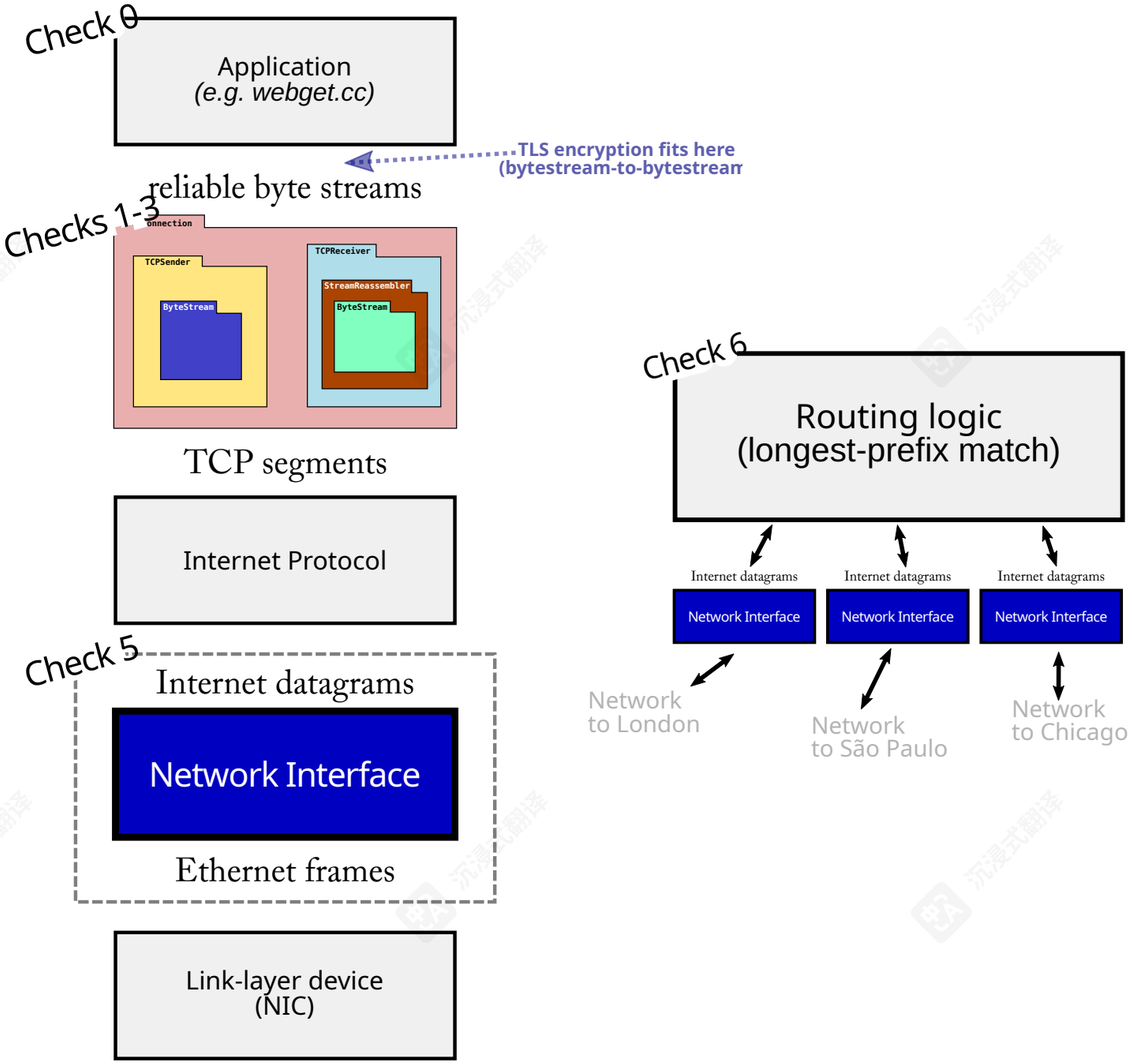

Figure 1: The network interface bridges the worlds of Internet datagrams and of link-layer frames. This component is useful as part of a host's TCP/IP stack (left side), and also as part of an IP router (right side).

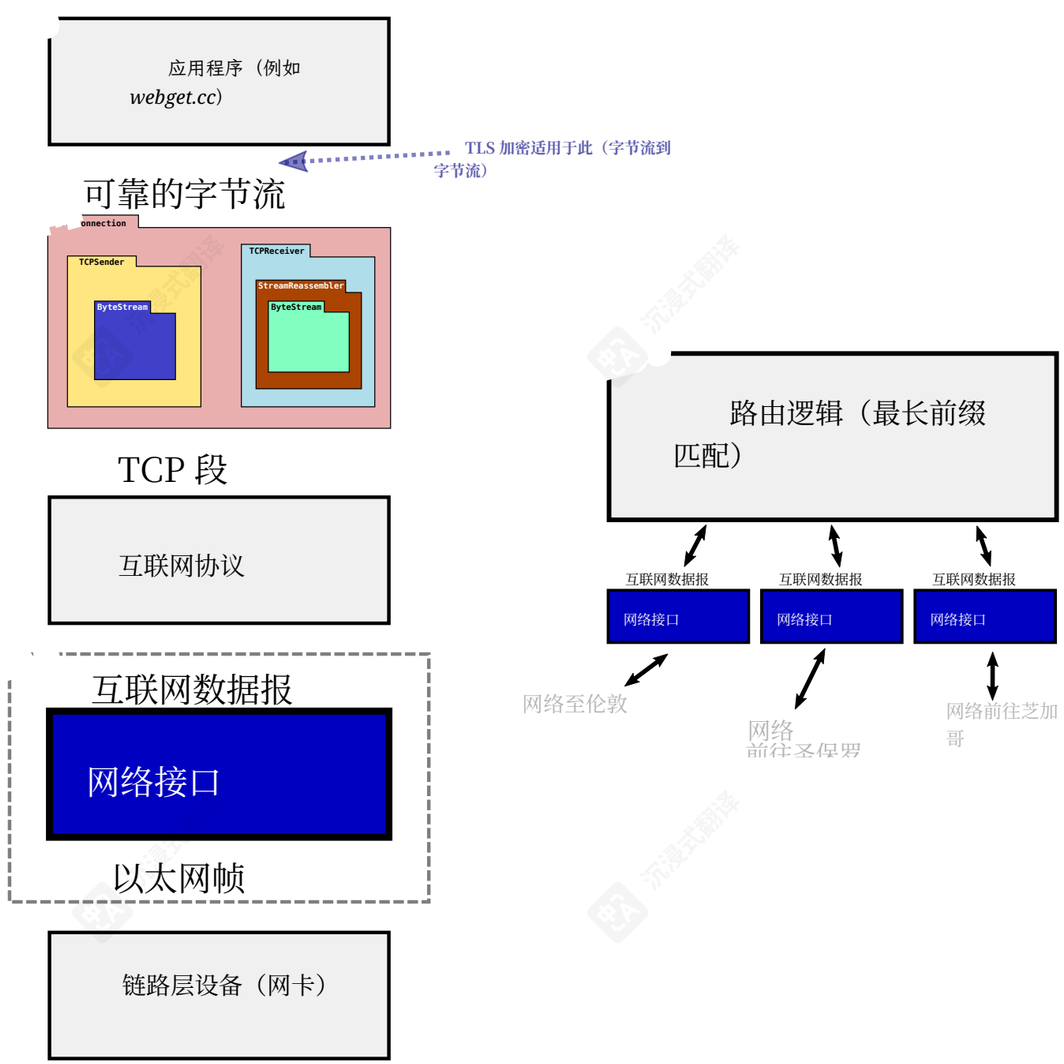

图1：网络接口桥接了互联网数据报和链路层帧的世界。该组件作为主机TCP/IP协议栈的一部分（左侧）和IP路由器的一部分（右侧）都很有用。

this mapping already, Linux broadcasts a query that asks, "Who claims the following IP address? What's your Ethernet address?" and waits for a response.

These functions are performed by the *network interface*: a component that translates outbound IP datagrams into link-layer (e.g., Ethernet) frames and vice versa. (In a real system, network interfaces typically have names like `eth0`, `eth1`, `wlan0`, etc.) **In this week's lab**, you'll implement a network interface, and stick it at the very bottom of your TCP/IP stack. Your code will produce raw Ethernet frames, which will be handed over to Linux through an interface called a TAP device—similar to a TUN device, but more low-level, in that it exchanges raw link-layer frames instead of IP datagrams.

Most of the work will be in looking up (and caching) the Ethernet address for each next-hop IP address. The protocol for this is called the **Address Resolution Protocol**, or **ARP**.

We've given you unit tests that put your network interface through its paces. In checkpoint 6, you'll use the same network interface outside the context of TCP, as a part of an IP router.

1 Getting started

1. Make sure you have committed all your solutions. Please don't modify any files outside the top level of the `src` directory, or `webget.cc`. You may have trouble merging the Checkpoint 5 starter code otherwise.
2. While inside the repository for the lab assignments, run `git fetch --all` to retrieve the most recent version of the lab assignment.
3. Download the starter code for Checkpoint 5 by running `git merge origin/check5-startercode`. (If you have renamed the "origin" remote to be something else, you might need to use a different name here, e.g. `git merge upstream/check5-startercode`.)
4. Make sure your build system is properly set up: `cmake -S . -B build`
5. Compile the source code: `cmake --build build`
6. Open and start editing the `writeups/check5.md` file. This is the template for your lab writeup and will be included in your submission.
7. Reminder: please make frequent **small commits** in your local Git repository as you work. If you need help to make sure you're doing this right, please ask a classmate or the teaching staff for help. You can use the `git log` command to see your Git history.

2 Checkpoint 5: The Address Resolution Protocol

Your main task in this lab will be to implement the three main methods of `NetworkInterface` (in the `network_interface.cc` file), maintaining a mapping from IP addresses to Ethernet addresses. The mapping is a cache, or "soft state": the NetworkInterface keeps it around

这个映射已经，Linux会广播一个查询，询问“谁声称以下IP地址？你的以太网地址是什么？”并等待响应。
这些功能由网络接口执行：一个组件，将出站IP数据报转换为链路层（例如以太网）帧，反之亦然。（在一个真实系统中，网络接口通常有名称如 `eth0`、 `eth1`、 `wlan0` 等。）在本周的实验中，你将实现一个网络接口，并将其放在你TCP/IP栈的最底部。你的代码将生成原始以太网帧，这些帧将通过一个称为TAP设备的接口传递给Linux——类似于TUN设备，但更底层，因为它交换原始链路层帧而不是IP数据报。

大部分工作将是在查找（并缓存）每个下一跳IP地址的以太网地址。这个协议称为地址解析协议，或ARP。

我们已经为你提供了单元测试，让网络接口得到充分测试。在检查点6中，你将使用同一个网络接口，在TCP的上下文之外，作为IP路由器的一部分。

1 开始使用

1. 确保你已经提交了所有解决方案。请不要修改 `src` 目录或 `webget.cc`的顶层文件。否则，否则你可能会在合并检查点5的启动代码时遇到问题。
2. 在实验作业的存储库中，运行 `git fetch --all` 以获取实验作业的最新版本。
3. 通过运行 `git merge origin/check5-startercode` 下载检查点5的启动代码。（如果你将“origin”远程仓库重命名为其他名称，你可能需要在这里使用不同的名称，例如 `git merge upstream/check5-startercode`。）
4. 确保您的构建系统已正确设置： `cmake -S . -B build`
5. 编译源代码： `cmake --build build`
6. 打开并开始编辑 `writeups/check5.md`文件。这是您的实验报告模板，并将包含在您的提交中。
7. 提醒：请随着工作进度，在您的本地Git仓库中频繁提交小改动。如果您需要帮助以确保正确操作，请向同学或教学人员寻求帮助。您可以使用 `git log` 命令查看您的Git历史记录。

2 检查点5：地址解析协议

本次实验的主要任务将是实现 `NetworkInterface` 的三个主要方法（在 `network interface.cc` 文件中），维护一个从 IP 地址到以太网地址的映射。该映射是一个缓存，或称为“软状态”：NetworkInterface 会保留它

for efficiency's sake, but if it has to restart from scratch, the mapping will naturally be regenerated without causing a problem.

1. `void NetworkInterface::send_datagram(InternetDatagram dgram, const Address &next_hop);`

   This method is called when the caller (e.g., your `TCPConnection` or a router) wants to send an outbound Internet (IP) datagram to the next hop.[1] It's your interface's job to translate this datagram into an Ethernet frame and (eventually) send it.

   - *If the destination Ethernet address is already known*, send it right away. Create an Ethernet frame (with `type = EthernetHeader::TYPE_IPv4`), set the payload to be the serialized datagram, and set the source and destination addresses.
   - *If the destination Ethernet address is unknown*, broadcast an ARP request for the next hop's Ethernet address, and queue the IP datagram so it can be sent after the ARP reply is received.

   Except: You don't want to flood the network with ARP requests. If the network interface already sent an ARP request about the same IP address in the last five seconds, don't send a second request—just wait for a reply to the first one. Again, queue the datagram until you learn the destination Ethernet address.

2. `void NetworkInterface::recv_frame(const EthernetFrame &frame);`

   This method is called when an Ethernet frame arrives from the network. The code should ignore any frames not destined for the network interface (meaning, the Ethernet destination is either the broadcast address or the interface's own Ethernet address stored in the `_ethernet_address` member variable).

   - *If the inbound frame is IPv4*, parse the payload as an `InternetDatagram` using the free `parse()` function in `helpers.hh` and, if successful (meaning the `parse()` function returned true), push the resulting datagram on to the `datagrams_received_` queue.
   - *If the inbound frame is ARP*, parse the payload as an `ARPMessage` and, if successful, remember the mapping between the sender's IP address and Ethernet address for 30 seconds. (Learn mappings from both requests and replies.) In addition, if it's an ARP request asking for our IP address, send an appropriate ARP reply.

3. `void NetworkInterface::tick(const size_t ms_since_last_tick);`

   This is called as time passes. Expire any IP-to-Ethernet mappings that have expired.

You can test your implementation by running `cmake --build build --target check5` This test does not rely on your TCP implementation.

[1] Please don't confuse the *ultimate* destination of the datagram, which is in the datagram's own header as the destination address, with the next hop. In this lab you're *only* going to care about the next hop's address.

为了提高效率，但如果它必须从头开始重新启动，映射自然会重新生成，而不会引起问题。

1. `void NetworkInterface::send datagram(InternetDatagram dgram, const Address &next hop);`

   当调用者（例如您的 `TCPConnection` 或路由器）想要将外出的互联网（IP）数据报发送到下一跳时，就会调用此方法。[1] 您的接口的工作是将此数据报转换为以太网帧，并（最终）发送它。

   - 如果目标以太网地址已知，立即发送。创建一个以太网帧（类型为 = 以太网头部类型IPv4），将有效载荷设置为序列化的数据报，并设置源和目标地址。
   - 如果目标以太网地址未知，则广播下一跳的以太网地址的ARP请求，并将IP数据报排队，以便在收到ARP回复后发送。

   除非性能：您不希望用ARP请求淹没网络。如果网络接口在过去的五秒内已经就同一个IP地址发送了ARP请求，则不要发送第二个请求——只需等待第一个请求的回复。同样，直到您知道目标以太网地址之前，请将数据报排队。

2. `void NetworkInterface::recv frame(const EthernetFrame &frame);`

   此方法在从网络到达以太网帧时被调用。代码应忽略任何不发送至网络接口的帧（这意味着，以太网目标地址是广播地址或接口自身存储在 `ethernet address` 成员变量中的以太网地址）。

   - 如果入站帧是IPv4，将有效载荷作为 `InternetDatagram` 使用 `parse()` 中的自由 `helpers.hh` 函数进行解析，如果成功（即 `parse()` 如果返回值为 true），将生成的数据报推送到 `datagrams received` 队列上。
   - 如果入站帧是ARP，将有效载荷解析为一个 `ARPMessage`，如果成功，则记住发送者的IP地址和以太网地址之间的映射，持续30秒。（从请求和回复中学习映射。）此外，如果它是一个请求我们IP地址的ARP请求，则发送一个适当的ARP回复。

3. `void NetworkInterface::tick(const size t ms since last tick);`

   随着时间推移，会过期任何已过期的IP到以太网映射。

您可以通过运行 `cmake --build build --target check5` 来测试您的实现。此测试不依赖于您的 TCP 实现。

[1] 请勿将数据报的最终目的地（它位于数据报自己的头部作为目标地址）与下一跳混淆。在本实验中，你只需要关心下一跳的地址。

3 Q & A

- *How much code are you expecting?*

  Overall, we expect the implementation (in `network_interface.cc`) will require about 100–150 lines of code in total.

- *How do I "send" an Ethernet frame?*

  Call `transmit()` on it.

- *What data structure should I use to record the mapping between next-hop IP address and Ethernet addresses?*

  Up to you!

- *How do I convert an IP address that comes in the form of an Address object, into a raw 32-bit integer that I can write into the ARP message?*

  Use the `Address::ipv4_numeric()` method.

- *What should I do if the NetworkInterface sends an ARP request but never gets a reply? Should I resend it after some timeout? Signal an error to the original sender using ICMP?*

  In real life, yes, both of those things, but don't worry about that in this lab. (In real life, an interface will eventually send an ICMP "host unreachable" back across the Internet to the original sender if it can't get a reply to its ARP requests.)

- *What should I do if an InternetDatagram is queued waiting to learn the Ethernet address of the next hop, and that information never comes? Should I drop the datagram after some timeout?*

  Yes, the network interface shouldn't store a datagram beyond the time when it would be willing to send another ARP request for the same IP address.

- *Where can I read if there are more FAQs after this PDF comes out?*

  Please check the website (https://cs144.github.io/lab_faq.html) and EdStem regularly.

4 Development and debugging advice

1. Implement the `NetworkInterface`'s public interface (and any private methods or functions you'd like) in the file `network_interface.cc`. You may add any private members you like to the `NetworkInterface` class in `network_interface.hh`.
2. You can test your code with `cmake --build build --target check5`.
3. Please re-read the section on "using Git" in the Checkpoint 0 document, and remember to keep the code in the Git repository it was distributed in on the `main` branch. **Make small commits, using good commit messages that identify what changed and why.**

3个问答

- 你期望多少代码？

  总的来说，我们预计（在 `network interface.cc`中的）实现将需要大约总共100-150行代码。

- 我该如何"发送"一个以太网帧？

  给 `transmit()` 打电话。

- 我应该使用什么数据结构来记录下一跳 *IP* 地址和以太网地址之间的映射？

  由你决定！

- 我该如何将一个以地址对象形式传入的 *IP* 地址，转换成一个原始的 *32* 位整数，以便我能写入 *ARP* 消息？

  使用`Address::ipv4 numeric()` 方法。

- 如果网络接口发送*ARP*请求但从未收到回复，我应该怎么做？是否在超时后重新发送？使用*ICMP*向原始发送者发送错误信号？

  在现实生活中，是的，这两件事，但在这个实验中不用担心这些。（在现实生活中，如果接口无法收到ARP请求的回复，最终会向原始发送者通过互联网发送ICMP "主机不可达" 消息。如果它无法收到对ARP请求的回复，则将数据包转发回原始发送者。）

- 如果互联网数据报正在排队等待学习以太网地址，我应该怎么做？下一跳的信息，而且这些信息永远不会到来？我应该超时后丢弃数据报吗？

  是的，网络接口不应在它愿意为同一个IP地址发送另一个ARP请求之前，存储数据报。

- 在*PDF*发布后，我可以在哪里阅读更多常见问题解答？

  请定期查看网站（https://cs144.github.io/lab faq.html）和EdStem。

4 开发和调试建议

1. 在文件 `network interface.cc`中实现 `NetworkInterface`的公共接口（以及你希望添加的任何私有方法或函数）。你可以将任何你喜欢的私有成员添加到 `network interface.hh`中的 `NetworkInterface` 类。

2. 你可以用 `cmake --build build --target check5` 测试你的代码.

3. 请重新阅读"使用 Git"部分，并记住将代码保持在它最初分发的 Git 仓库的 `main` 分支上。进行小规模提交，使用清晰的提交信息，说明变更内容及其原因。

4. Please work to make your code readable to the CA who will be grading it for style. Use reasonable and clear naming conventions for variables. Use comments to explain complex or subtle pieces of code. Use "defensive programming"—explicitly check preconditions of functions or invariants, and throw an exception if anything is ever wrong. Use modularity in your design—identify common abstractions and behaviors and factor them out when possible. Blocks of repeated code and enormous functions will make it hard to follow your code.

5 Submit

1. In your submission, please only make changes to the `.hh` and `.cc` files in the `src` directory. Within these files, please feel free to add private members as necessary, but please don't change the *public* interface of any of the classes.

2. Before handing in any assignment, please run these in order:

   (a) Make sure you have committed all of your changes to the Git repository. You can run `git status` to make sure there are no outstanding changes. Remember: make small commits as you code.

   (b) `cmake --build build --target format` (to normalize the coding style)

   (c) `cmake --build build --target check5` (to make sure the automated tests pass)

   (d) Optional: `cmake --build build --target tidy` (suggests improvements to follow good C++ programming practices)

3. Write a report in `writeups/check5.md`. This file should be a roughly 20-to-50-line document with no more than 80 characters per line to make it easier to read. The report should contain the following sections:

   (a) **Program Structure and Design.** Describe the high-level structure and design choices embodied in your code. You do not need to discuss in detail what you inherited from the starter code. Use this as an opportunity to highlight important design aspects and provide greater detail on those areas for your grading TA to understand. You are strongly encouraged to make this writeup as readable as possible by using subheadings and outlines. Please do not simply translate your program into an paragraph of English.

   (b) **Implementation Challenges.** Describe the parts of code that you found most troublesome and explain why. Reflect on how you overcame those challenges and what helped you finally understand the concept that was giving you trouble. How did you attempt to ensure that your code maintained your assumptions, invariants, and preconditions, and in what ways did you find this easy or difficult? How did you debug and test your code?

4. 请努力让你的代码对评阅的助教（CA）来说易于阅读。为变量使用合理且清晰的命名规范。使用注释来解释复杂或微妙的代码部分。使用“防御式编程”——明确检查函数的前置条件或不变式，并在任何情况下出错时抛出异常。在设计中使用模块化——识别常见的抽象和行为，并在可能的情况下将它们提取出来。重复的代码块和巨大的函数这会使你的代码难以理解。

5 提交

1. 在你的提交中，请仅修改 `.hh` 和 `.cc` 文件，这些文件位于 `src` 目录中。在这些文件中，请根据需要自由添加私有成员，但请不要更改任何类的公共接口。

2. 在提交任何作业之前，请按顺序运行这些命令：

   (a) 确保您已将所有更改提交到 Git 仓库。您可以使用 `git status` 来确保没有未解决的更改。记住：边编码边提交小批次的更改。

   (b) `cmake --build build --target format`（以规范化编码风格）

   (c) `cmake --build build --target check5`（以确保自动测试通过）

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

3. 使用该文件应为一个大约 20–50 行的文档，每行不超过80个字符，以便于阅读。报告应包含以下部分：

   (a) 程序结构与设计。描述代码中体现的高级结构和设计选择。你不需要详细讨论从启动代码中继承的内容。利用这个机会突出重要的设计方面，并为你的评分助教提供更详细的说明。强烈建议你通过使用子标题和提纲使这份写稿尽可能易读。请不要简单地将你的程序翻译成一段英文。

   (b) 实现挑战。描述你发现最麻烦的代码部分，并解释原因。反思你是如何克服这些挑战的，以及是什么帮助你最终理解了那个困扰你的概念。你是如何尝试确保你的代码保持你的假设、不变量和前提条件的，以及你发现这容易还是困难？你是如何调试和测试你的代码的?

(c) **Remaining Bugs.** Point out and explain as best you can any bugs (or unhandled edge cases) that remain in the code.

4. Please also fill in the number of hours the assignment took you and any other comments.

5. Please let the course staff know ASAP of any problems at a lab session, or by posting a question on Ed. Good luck!

(c) 剩余的Bug。尽可能指出并解释代码中仍然存在的任何Bug（或未处理的边缘情况）。

4. 请同时填写完成作业所花费的小时数以及任何其他意见。

5. 请尽快告知课程工作人员在实验课中遇到的问题，或在Ed上发布问题。祝你好运！