Lab Checkpoint 6: building an IP router

Due: Sunday, November 16, 11:59 p.m.

0 Collaboration Policy

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.

1 Overview

In this week's lab checkpoint, you'll implement an IP router on top of your existing `NetworkInterface`. A router has *several* network interfaces, and can receive Internet datagrams on any of them. The router's job is to forward the datagrams it gets according to the **routing table**: a list of rules that tells the router, for any given datagram,

- What interface to send it out
- The IP address of the next hop

Your job is to implement a router that can figure out these two things for any given datagram. (You will not need to implement the algorithms that *make* the routing table, e.g. RIP, OSPF, BGP, or an SDN controller—just the algorithm that *follows* the routing table.)

Your implementation of the router will use the Minnow library with a new `Router` class, and tests that will check your router's functionality in a simulated network. Checkpoint 6 builds on your implementation of `NetworkInterface` from Checkpoint 5, but does *not* use the TCP stack you implemented previously. IP routers don't have to know anything about TCP, ARP, or Ethernet (only IP). We expect your implementation will require about **30–60 lines of code**. (The `scripts/lines-of-code` tool prints "Router: 38 lines of code" from the starter code, and "89 lines of code" for our example solutions.)

2 Getting started

1. Make sure you have committed all your solutions to Checkpoint 5. Please don't modify any files outside the top level of the `src` directory, or `webget.cc`. You may have trouble merging the Checkpoint 6 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 6 by running `git merge origin/check6-startercode`.

实验检查点6：构建IP路由器

截止日期：11月16日星期日，晚上11:59

0 协作政策

协作政策：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协作者或任何灰色地带——公开是最佳政策。

1 概述

在本周的实验检查点中，你将在现有的`NetworkInterface`之上实现一个IP路由器。路由器有多个网络接口，并且可以在任何一个接口上接收互联网数据报。路由器的工作是根据路由表转发它接收到的数据报：路由表是一系列规则，告诉路由器，对于任何给定的数据报，

- 应该通过哪个接口发送
- 下一跳的IP地址

你的任务是实现一个路由器，能够为任何给定的数据报确定这两件事。（你不需要实现生成路由表的算法，例如RIP、OSPF、BGP或SDN控制器——只需要实现遵循路由表的算法。）

你的路由器实现将使用Minnow库和一个新的 `Router` 类，以及将在模拟网络中测试你路由器功能的测试。检查点6基于你在检查点5中实现的 `NetworkInterface` ，但不需要使用你之前实现的TCP栈。IP路由器不需要了解TCP、ARP或以太网（只需要IP）。我们预计你的实现将需要大约30-60行代码。（从启动代码中， `scripts/lines-of-code` 工具打印“路由器：38行代码”，对于我们示例解决方案则打印“89行代码”。）

2 开始使用

1. 确保您已将所有解决方案提交到 Checkpoint 5。请不要修改 `src` 目录顶层以外的任何文件，或 `webget.cc`。否则，您可能会在合并 Checkpoint 6 起始代码时遇到问题。

2. 在实验作业的存储库内，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/check6-startercode` 下载 Checkpoint 6 的初始代码。

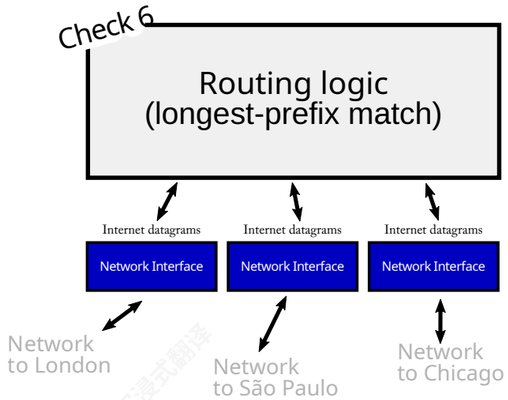

Figure 1: A router contains several network interfaces and can receive IP datagrams on any one of them. The router forwards any datagram it receives to the next hop, on the appropriate outbound interface. The routing table tells the router how to make this decision.

(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/check6-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/check6.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.

3 Implementing the Router

In this lab, you will implement a `Router` class that can:

- keep track of a routing table (the list of forwarding rules, or routes), and
- forward each datagram it receives:
  - to the correct next hop
  - on the correct outgoing `NetworkInterface`.

Your implementation will be added to the `router.hh` and `router.cc` skeleton files. Before you get to coding, please review the documentation for the new `Router` class in `router.hh`.

Here are the two methods you'll implement, and what we're expecting in each:

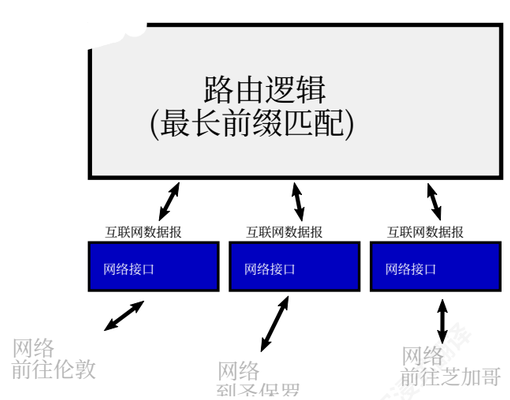

图1：路由器包含多个网络接口，可以在任何一个接口上接收IP数据报。路由器会将接收到的任何数据报转发到合适的出站接口上的下一跳。路由表告诉路由器如何做出这个决定。

(如果你将"origin"远程仓库名重命名为其他名称，这里可能需要使用不同的名称，例如 `git merge upstream/check6-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/check6.md` 文件。这是你的实验报告模板，并将包含在你的提交中。

7. 提醒：请在你工作时，在你的本地 Git 仓库中频繁提交小的更改。如果你需要帮助以确保你这样做是正确的，请向同学或教学人员寻求帮助。你可以使用 `git log` 命令来查看你的 Git 历史记录。

3 实现路由器

在这个实验中，你将实现一个 `Router` 类，它可以：

- 跟踪一个路由表（转发规则或路由的列表），和
- 转发它接收到的每个数据报：
  - 到正确的下一跳
  - 在正确的出站 `NetworkInterface` 上。

您的实现将被添加到 `router.hh` 和 `router.cc` 骨架文件中。在您开始编码之前，请查阅 `router.hh` 中新 `Router` 类的文档。

以下是您将实现的两个方法，以及我们对每个方法的期望：

```
void add_route(uint32_t route_prefix,
               uint8_t prefix_length,
               optional<Address> next_hop,
               size_t interface_num);
```

This method adds a route to the routing table. You'll want to add a data structure as a private member in the `Router` class to store this information. All this method needs to do is save the route for later use.

What do the parts of a route mean?

A route is a "match-action" rule: it tells the router that *if* a datagram is headed for a particular network (a range of IP addresses), and *if* the route is chosen as the most specific matching route, *then* the router should forward the datagram to a particular next hop on a particular interface.

The "match": is the datagram headed for this network? The `route_prefix` and `prefix_length` together specify a range of IP addresses (a network) that might include the datagram's destination. The `route_prefix` is a 32-bit numeric IP address. The `prefix_length` is a number between 0 and 32 (inclusive); it tells the router *how many most-significant bits* of the `route_prefix` are significant. For example, to express a route to the network "18.47.0.0/16" (this matches *any* 32-bit IP address where the first two bytes are 18 and 47), the `route_prefix` would be 305070080 ($18 \times 2^{24} + 47 \times 2^{16}$), and the `prefix_length` would be 16. Any datagram destined for "18.47.x.y" will match.

The "action": what to do if the route matches and is chosen. If the router is *directly* attached to the network in question, the `next_hop` will be an empty `optional`. In that case, the `next_hop` is the datagram's destination address. But if the router is connected to the network in question through some other router, the `next_hop` will contain the IP address of the next router along the path. The `interface_num` gives the index of the router's `NetworkInterface` that should use to send the datagram to the next hop. You can access this interface with the `interface(`*interface_num*`)` method.


```
void add_route(uint32_t route_prefix,
uint8_t prefix_length,
optional<Address> next_hop,
size_t interface_num);
```

此方法向路由表添加路由。您需要在 `Router` 类中添加一个数据结构作为私有成员来存储这些信息。该方法只需保存路由以供后续使用即可。

路由的各个部分代表什么?

路由是一种"匹配-动作"规则：它告诉路由器，如果数据报的目标是特定网络（一组IP地址），并且如果该路由被选为最具体的匹配路由，则路由器应将数据报转发到特定接口上的特定下一跳。

"匹配"：数据报的目标是此网络吗？ `route prefix`和`prefix length` 共同指定了一组IP地址（一个网络），该网络可能包含数据报的目标。 `route prefix` 是一个32位数字IP地址。`prefix length` 是一个介于0到32（含）之间的数字；它告诉路由器 `route prefix` 的哪些最高有效位是重要的。例如，要表示路由到"18.47.0.0/16"网络（这匹配任何前两个字节为18和47的32位IP地址），`route prefix` 将是305070080 ($18 \times 2^{24} + 47 \times 2^{16}$)，而 `prefix length` 将是16。任何目标为"18.47.x.y"的数据报都将匹配。

"动作"：如果路由匹配且被选择时该做什么。如果路由器直接连接到相关网络，则 `next hop` 将是空的 `optional`。在这种情况下， `next hop` 是数据报的目的地址。但如果路由器通过其他路由器连接到相关网络，则 `next hop` 将包含路径上下一路由器的IP地址。 `interface num` 给出路由器应使用其 `NetworkInterface` 将数据报发送到下一跳的索引。您可以通过 `interface(`接口num`)` 方法访问此接口。

```
void route();
```

Here's where the rubber meets the road. This method needs to route each incoming datagram to the next hop, out the appropriate interface. It needs to implement the "longest-prefix match" logic of an IP router to find the *best* route to follow. That means:

- The `Router` searches the routing table to find the routes that match the datagram's destination address. By "match," we mean the most-significant `prefix_length` bits of the destination address are identical to the most-significant `prefix_length` bits of the `route_prefix`.
- Among the matching routes, the router chooses the route with the *biggest* value of `prefix_length`. This is the **longest-prefix-match** route.
- If no routes matched, the router drops the datagram.
- The router decrements the datagram's TTL (time to live). If the TTL was zero already, or hits zero after the decrement, the router should drop the datagram.
- Otherwise, the router sends the modified datagram on the appropriate interface (`interface(`*interface_num*`)->send_datagram()`) to the appropriate next hop.

There's a beauty (or at least a successful abstraction) in the Internet's design here: the router never thinks about TCP, about ARP, or about Ethernet frames. The router doesn't even know what the link layer looks like. The router only thinks about Internet datagrams, and only interacts with the link layer through the `NetworkInterface` abstraction. When it comes to questions like, "How are link-layer addresses resolved?" or "Does the link layer even have its own addressing scheme distinct from IP?" or "What's the format of the link-layer frames?" or "What's the meaning of the datagram's payload?", the router just doesn't care.

4 Testing

You can test your implementation by running `cmake --build build --target check5`. This will test your router in a particular simulated network, shown in Figure 2.

5 Q & A

- *What data structure should I use to record the routing table?*

  Up to you! But please don't get crazy. It's perfectly acceptable for each datagram to require $O(N)$ work, where N is the number of entries in the routing table. If you'd like to do something more efficient, we'd encourage you to get a working implementation


```
void route();
```

事情的关键就在这里。这种方法需要将每个传入的数据报路由到下一个跳点，通过合适的接口出去。它需要实现IP路由器的“最长前缀匹配”逻辑，以找到最佳路由路径。这意味着：

- 它会搜索路由表，以找到与数据报目标地址匹配的路由。我们所说的“匹配”，是指目标地址的最高位 `prefix length` 与`route prefix`的最高位 `prefix length` 相同。
- 在所有匹配的路由中，路由器会选择`prefix length`值最大的路由。这就是最长前缀匹配路由。
- 如果没有路由匹配，路由器会丢弃数据报。
- 路由器会减少数据报的TTL（生存时间）。如果TTL已经是零，或者减去后变为零，路由器应该丢弃该数据报。
- 否则，路由器会将修改后的数据报通过适当的接口（ `interface(`接口编号`)->send datagram()`）发送到适当的下一跳。

互联网的设计中有一个优点（或者至少是一个成功的抽象）：路由器从不考虑TCP、ARP或以太网帧。路由器甚至不知道链路层的样子。路由器只关心互联网数据报，并且仅通过 `NetworkInterface`抽象与链路层交互。当遇到诸如“如何解析链路层地址？”、“链路层是否拥有独立于IP的地址方案？”、“链路层帧的格式是什么？”或“数据报的有效载荷是什么含义？”等问题时，路由器根本不在乎。

4 测试

您可以通过运行 `cmake --build build --target check5` 来测试您的实现。这将测试您的路由器在一个特定的模拟网络中，如图 2 所示。

5 Q & A

- 我应该使用什么数据结构来记录路由表？

  由您决定！但请不要发疯。对于每个数据报都需要 $O(N)$ 工作是完全可以接受的，其中 N 是路由表中的条目数。如果您想做得更高效，我们鼓励您先实现一个可以工作的版本

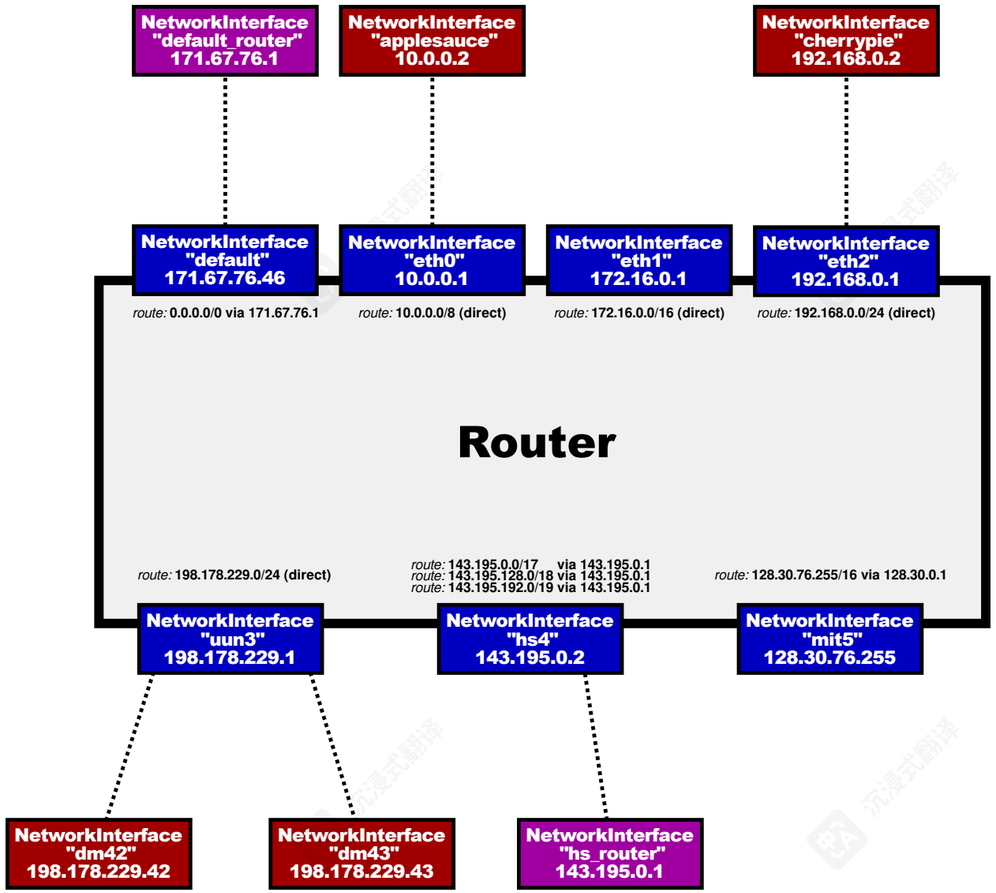

Figure 2: The simulated test network used in the `router` test, also run by `cmake --build build --target check5`. (Fun fact: the UUN network is David Mazières's slice of the Internet, allocated in 1993. The `whois` tool, or the linked website, can be used to look up who controls each IP address allocation.)

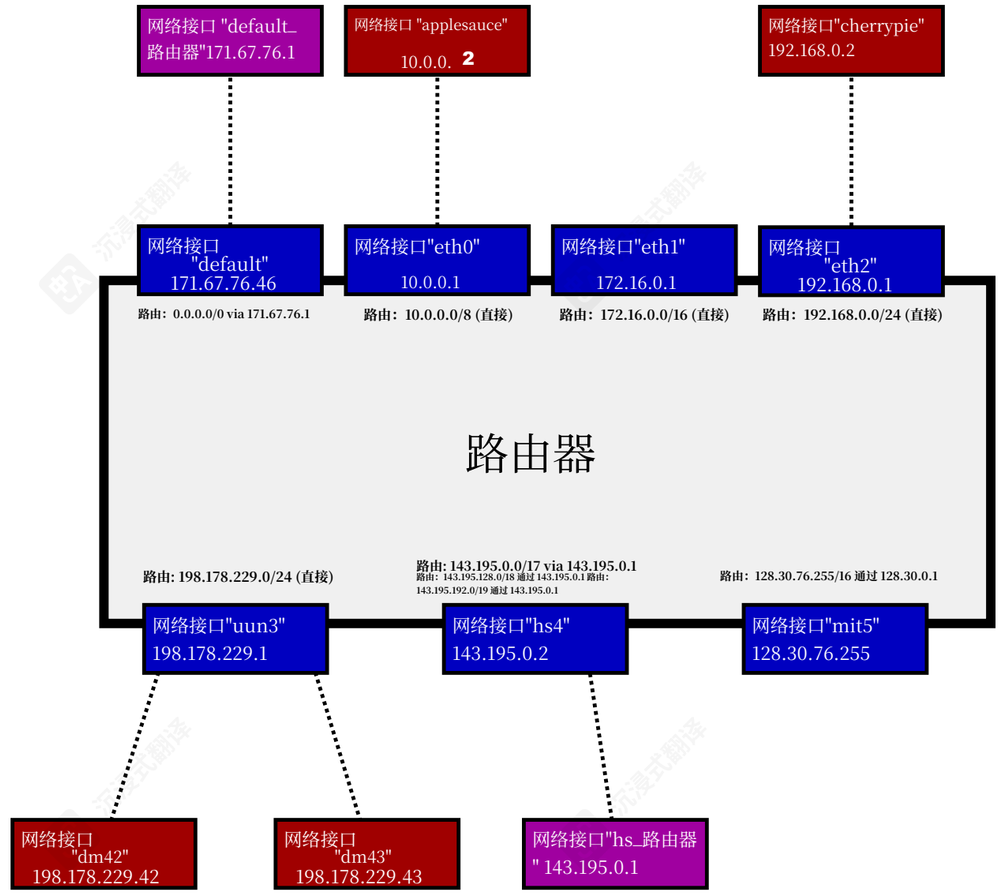

图2：`router` 测试中使用的模拟测试网络，也由`cmake --build build --target check5`运行。（趣闻：该 UUN 网络是David Mazi`eres的互联网切片，于1993年分配。可以使用 `whois` 工具或链接网站查询每个IP地址分配的归属方。）

first before optimizing, and carefully document and comment whatever you choose to implement.

- *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.

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

  Use the `Address::from_ipv4_numeric()` method.

- *How do I compare the most-significant N bits (where $0 \leq N \leq 32$) of one 32-bit IP address with the most-significant N bits of another 32-bit IP address?*

  This is probably the "trickiest" part of this assignment—getting that logic right. It may be worth writing a small test program in C++ (a short standalone program) or adding a test to Minnow to verify your understanding of the relevant C++ operators and double-check your logic.

  Recall that in C and C++, it can produce *undefined behavior* to shift a 32-bit integer by 32 bits. The tests run your code under sanitizers that try to detect this. You can run the router test directly by running `./build/tests/router` from the minnow directory.

- *If the router has no route to the destination, or if the TTL hits zero, shouldn't it send an ICMP error message back to the datagram's source?*

  In real life, yes, that would be helpful. But not necessary in this lab—dropping the datagram is sufficient. (Even in the real world, not every router will send an ICMP message back to the source in these situations.)

- *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.

6 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)

先完成基本功能，再进行优化，并仔细记录和注释你选择实现的内容。

- 如何将一个以*Address*对象形式传入的*IP*地址，转换成一个我可以写入*ARP*消息的原始*32*位整数？

  使用 `Address::ipv4 numeric()` 方法。

- 如何将一个以原始*32*位整型形式出现的*IP*地址转换成地址对象？

  使用 `Address::from ipv4 numeric()` 方法。

- 如何比较一个*32*位*IP*地址的最高有效 N 位（其中 $0 \leq N \leq 32$）与另一个*32*位*IP*地址的最高有效 N 位？如何比较一个*32*位*IP*地址的最高有效 N 位与另一个*32*位*IP*地址的最高有效 N 位？

  这部分可能是这个作业中最“棘手”的部分——正确实现该逻辑。值得用C++ （一个简短的独立程序）编写一个小测试程序，或向Minnow添加一个测试用例来验证你对相关C++ 运算符的理解，并再次检查你的逻辑。

  请记住，在 C 和 C++ 中，将 32 位整数左移 32 位可能会产生未定义行为。测试会在尝试检测此问题的卫生器下运行你的代码。你可以通过从 minnow 目录运行 `./build/tests/router` 直接运行路由器测试。

- 如果路由器没有到达目的地的路由，或者如果 *TTL* 达到零，它不应该向数据报的源发送 *ICMP* 错误消息吗？

  在现实生活中，是的，那会很有帮助。但在本实验中不是必要的——丢弃数据报就足够了。（即使在真实世界中，并非所有路由器在这种情况下都会向源发送 ICMP 消息。）

- 我可以在哪里阅读，如果在这个 *PDF* 发布后还有更多常见问题解答？

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

6 提交

1. 在您的提交中，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，请根据需要添加私有成员，但请勿更改任何类的公共接口。

2. 提交任何作业前，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。您可以通过运行 `git status` 来确保没有未完成的更改。记住：边编写代码边进行小规模提交。

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

(c) `cmake --build build --target check6` (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/check6.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?

   (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 the lab sessions, or by posting a question on EdStem.

(c) `cmake --build build --target check6` (确保自动测试通过)

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

3. 用 `writeups/check6.md` 这个文件应为一个约20到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) 程序结构和设计。描述你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TA提供更多细节以供理解。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) 实施挑战。描述你发现代码中最令人头疼的部分，并解释原因。反思你是如何克服这些挑战的，以及是什么最终帮助你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) 剩余的bug。指出并尽可能解释代码中仍然存在的任何bug（或未处理的边缘情况）。

4. 请填写完成作业所花费的小时数以及任何其他评论。

5. 请尽快告知课程工作人员实验课期间遇到的问题，或在EdStem上发布问题。