Lab Checkpoint 0: networking warmup

Due: Sunday, September 28, 11:59 p.m.

Welcome to CS144: Introduction to Computer Networking. In this warmup, you will set up an installation of GNU/Linux on your computer, learn how to perform some tasks over the Internet by hand, write a small program in C++ that fetches a Web page over the Internet, and implement (in memory) one of the key abstractions of networking: a reliable stream of bytes between a writer and a reader. We expect this warmup to take you between 2 and 6 hours to complete (future checkpoints will take more of your time). Three quick points about the lab assignment:

- It's a good idea to read the whole document before diving in!
- Over the course of this 8-part lab assignment, you'll be building up your own implementation of a significant portion of the Internet—a router, a network interface, and the TCP protocol (which transforms unreliable datagrams into a reliable byte stream). *Most weeks will build on work you have done previously,* i.e., you are building up your own implementation gradually over the course of the quarter, and you'll continue to use your work in future weeks. This makes it hard to "skip" a checkpoint.
- If you don't meet the CS144 prerequisites, please don't take this class yet—our teaching staff's resources are limited. And please use checkpoints 0 and 1 as a gauge: if you find yourself uncomfortable with the programming in the first two checkpoints, **please consider taking CS144 in a later year** after you've attained more comfort with this kind of programming (perhaps after taking CS 106L, embarking on a self-directed programming project, or otherwise building up your comfort and experience level).
- The lab documents aren't "specifications"—meaning they're not intended to be consumed in a one-way fashion. They're written closer to the level of detail that a software engineer will get from a boss or client. We expect that you'll benefit from attending the lab sessions and asking clarifying questions if you find something to be ambiguous and you think the answer matters. We'll update the "lab FAQ" document on the course website in response to late questions that need clarification.

0 Collaboration Policy

The programming assignments must be your own work: You must write all the code you hand in for the programming assignments, except for the code that we give you as part of the assignment. Please do not copy-and-paste code from Stack Overflow, GitHub, ChatGPT, or other sources. If you base your own code on examples you find on the Web or elsewhere, cite the URL in a comment in your submitted source code.

Working with others: You may not show your code to anyone else, look at anyone else's code for these assignments (whether human- or AI-authored), or look at solutions from previous years. You may discuss the assignments with other students, but do not copy anybody's code. If you discuss an assignment with another student, please name them in a

实验检查点0：网络热身

截止日期：9月28日星期日，晚上11:59

欢迎来到CS144：计算机网络导论。在这个热身练习中，你将在你的计算机上设置GNU/Linux的安装，学习如何手动通过互联网执行一些任务，用C++ 编写一个小程序来通过互联网获取网页，并实现（在内存中）网络的一个关键抽象：一个在写入者和读取者之间的可靠字节流。我们预计这个热身练习需要你花费2到6小时完成（未来的检查点将需要更多时间）。关于实验作业有三点需要注意：

C• 在开始之前，最好先通读整个文档！

• 在这个8部分的实验作业中，你将逐步构建自己的互联网重要部分的实现——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）。大多数周的工作都基于你之前完成的内容，也就是说，你将逐步构建自己的实现，并在后续几周继续使用你的工作。这使得“跳过”某个检查点变得困难。

• 如果你没有满足CS144的先修条件，请不要现在就选修这门课——我们的教学资源有限。请将检查点0和1作为参考：如果你在最初的两个检查点中发现自己对编程感到不适，请考虑在更熟悉这种编程（也许在完成CS 106L、开展一个自我导向的编程项目或以其他方式提高你的舒适度和经验水平后）的年份再选修CS144。

• 实验文档并非“规范”——这意味着它们并非设计为单向消费的。它们所写的详细程度更接近软件工程师从老板或客户那里获得的水平。我们预计，如果你发现某些内容含糊不清且你认为答案很重要，参加实验课并提出澄清问题将对你有所帮助。我们将根据需要澄清的延迟问题，在课程网站上更新“实验常见问题解答”文档。

0 协作政策

编程作业必须是你自己的工作：你必须编写所有提交的编程作业代码，除了我们作为作业一部分提供的代码。请不要从 Stack Overflow、GitHub、ChatGPT 或其他来源复制粘贴代码。如果你基于在网络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的注释中注明网址。

与他人合作：你不得向任何人展示你的代码，也不得查看其他人（无论是人类还是AI编写）的代码，或查看往年解决方案。你可以与其他学生讨论作业，但不得抄袭任何人的代码。如果你与另一名学生讨论了作业，请在提交的源代码中的注释里注明他们的名字。请参考课程管理手册的更多细节，如有不清楚的地方请在EdStem上提问。GitHub Copilot或ChatGPT等服务应被视为等同于“之前修过CS144的学生。”

comment in your submitted source code. Please refer to the course administrative handout for more details, and ask on EdStem if anything is unclear. Services like GitHub Copilot or ChatGPT should be considered to be equivalent to "a student that took CS144 in a prior year."

EdStem: Please feel free to ask questions on EdStem, but please don't post any source code.

1 Set up GNU/Linux on your computer

CS144's assignments require the GNU/Linux operating system and a recent C++ compiler that supports the C++ 2023 standard. Please choose one of these three options:

1. **Recommended:** Install the CS144 VirtualBox virtual-machine image (instructions at https://stanford.edu/class/cs144/vm_howto/vm-howto-image.html).
2. Use a Google Cloud virtual machine (instructions at https://stanford.edu/class/cs144/vm_howto). We have applied for $50 coupons from Google and will distribute them if/when they arrive.
3. Run Ubuntu version 25.04.
4. Use another GNU/Linux distribution "at your own risk," but be aware that you may hit roadblocks along the way and will need to be comfortable debugging them. Your code will be tested on Ubuntu 25.04 LTS with `g++` 14.2.0 and must compile and run properly under those conditions.
5. If you have a 2020–24 MacBook (with the ARM64 M-series chips), VirtualBox will not successfully run. Instead, please install the UTM virtual machine software and our ARM64 virtual machine image from https://stanford.edu/class/cs144/vm_howto/.

2 Install the required packages

These commands will install the required Ubuntu (Debian) packages to run the course software:

```
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 Networking by hand

Let's get started with using the network. You are going to do two tasks by hand: retrieving a Web page (just like a Web browser) and sending an email message (like an email client). Both of these tasks rely on a networking abstraction called a *reliable bidirectional byte stream*: you'll type a sequence of bytes into the terminal, and the same sequence of bytes will eventually

请参考课程管理手册的更多细节，如有不清楚的地方请在EdStem上提问。GitHub Copilot或ChatGPT等服务应被视为等同于“之前修过CS144的学生。”

EdStem：请随时在EdStem上提问，但请不要发布任何源代码。

1 在你的计算机上设置GNU/Linux

CS144的作业需要GNU/Linux操作系统和最近支持C++ 标准的C++ 2023 编译器。请选择以下三个选项中的一个：

1. 推荐：安装CS144 VirtualBox虚拟机镜像（说明请见 https://stanford.edu/class/cs144/vm howto/vm-howto-image.html）。
2. 使用 Google Cloud 虚拟机（说明文档请参考 https://stanford.edu/class/cs144/ v m howto）。我们已经申请了 Google 提供的 50 美元优惠券，如果收到将会分发给大家。
3. 运行 Ubuntu 25.04 版本。
4. 可以使用其他 GNU/Linux 发行版“自行承担风险”，但需注意可能会遇到各种问题，并且需要熟悉调试。你的代码将在 Ubuntu 25.04 LTS 环境下使用 `g++` 14.2.0 进行测试，必须在这些条件下正确编译和运行。
5. 如果你有 2020–24 年的 MacBook（搭载 ARM64 M 系列芯片），VirtualBox 将无法成功运行。请改为安装 UTM 虚拟机软件以及我们从 https://stanford.edu/class/cs144/vm howto/ 提供的 ARM64 虚拟机镜像。

2 安装所需的软件包

这些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 手工网络

让我们开始使用网络。你需要手动完成两个任务：获取一个网页（就像使用网页浏览器一样）和发送电子邮件消息（就像使用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：你会在终端中输入一系列字节，而同一系列字节最终会以相同的顺序被发送到另一台计算机上运行的程序（服务器）。服务器会用自己的字节序列回应，并将其发送回你的终端。

be delivered, in the same order, to a program running on another computer (a server). The server responds with its own sequence of bytes, delivered back to your terminal.

3.1 Fetch a Web page

1. In a Web browser, visit http://cs144.keithw.org/hello and observe the result.
2. Now, you'll do the same thing the browser does, by hand.
   (a) **On your VM** (or on your own computer—e.g. the Terminal program on macOS), run `telnet cs144.keithw.org http`. This tells the `telnet` program to open a reliable byte stream between your computer and another computer (named `cs144.keithw.org`), and with a particular *service* running on that computer: the "`http`" service, for the Hyper-Text Transfer Protocol, used by the World Wide Web.[1]

   If your computer has been set up properly and is on the Internet, you will see:

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

   If you need to quit, hold down ctrl and press], and then type `close ↵`.

   (b) Type `GET /hello HTTP/1.1 ↵`. This tells the server the *path* part of the URL. (The part starting with the third slash.)

   (c) Type `Host: cs144.keithw.org ↵`. This tells the server the *host* part of the URL. (The part between http:// and the third slash.)

   (d) Type `Connection: close ↵`. This tells the server that you are finished making requests, and it should close the connection as soon as it finishes replying.

   (e) Hit the Enter key one more time: `↵`. This sends an empty line and tells the server that you are done with your HTTP request.

   (f) If all went well, you will see the same response that your browser saw, preceded by HTTP *headers* that tell the browser how to interpret the response.
3. **Assignment:** Now that you know how to fetch a Web page by hand, show us you can! Use the above technique to fetch the URL http://cs144.keithw.org/lab0/*sunetid*, replacing *sunetid* with your own primary SUNet ID. You will receive a secret code in the `X-Your-Code-Is:` header. Save your SUNet ID and the code for inclusion in your writeup.

[1]The computer's name has a numerical equivalent (`104.196.238.229`, an *Internet Protocol v4 address*), and so does the service's name (`80`, a *TCP port number*). We'll talk more about these later.

服务器会用自己的字节序列回应，并将其发送回你的终端。

3.1 获取网页

1. 在网页浏览器中，访问 http://cs144.keithw.org/hello 并观察结果。
2. 现在，你需要手动完成浏览器所做的事情。
   (a) 在你的虚拟机（或你自己的计算机——例如 macOS 上的终端程序）上运行 `telnet cs144.keithw.org http`。这告诉 `telnet` 程序在您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，并在该计算机上运行特定的服务："`http`" 服务，即用于万维网的"超文本传输协议"。[1]

   如果您的计算机已正确设置并已连接到互联网，您将看到：

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

   如果您需要退出，请按住ctrl键并按下]，，然后输入 `close` 。

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

   (c) 输入 `Host: cs144.keithw.org ↵`. 这会告诉服务器网址的主机部分。(位于 http:// 和第三个斜杠之间的部分。)

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

   (e) 再按一次 Enter 键: `↵`. 这会发送一个空行并告知服务器会告诉你已完成 HTTP 请求。

   (f) 如果一切顺利，你会看到与浏览器看到相同的响应，该响应以 HTTP 头部开头这些头部会告诉浏览器如何解析响应。
3. 作业：现在你已经知道如何手动获取网页，来展示你的能力吧！使用上述方法获取网址 http://cs144.keithw.org/lab0/sunetid，将 sunetid 替换为你自己的主 SUNetID。你将在 `X-Your-Code-Is:` 头部收到一个秘密代码。保存你的 SUNetID 和代码，以便包含在你的报告里。

[1]计算机的名称有一个数值等效（104.196.238.229，一个 IPv4地址），服务的名称也有一个数值等效（80，一个 TCP端口号）。我们之后会更多地讨论这些。

3.2 Send yourself an email

Now that you know how to fetch a Web page, it's time to send an email message, again using a reliable byte stream to a service running on another computer.

1. SSH to *sunetid*@`cardinal.stanford.edu` (to make sure you are on Stanford's network), then run `telnet 67.231.149.169 smtp`.[2] The "smtp" service refers to the Simple Mail Transfer Protocol, used to send email messages. If all goes well, you will see:

```
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. First step: identify your computer to the email server. Type `HELO mycomputer.stanford.edu ↵`. Wait to see something like "`250 ... Hello cardinal3.stanford.edu [171.67.24.75], pleased to meet you`".

3. Next step: who is sending the email? Type `MAIL FROM: sunetid@stanford.edu ↵`. Replace *sunetid* with your SUNet ID.[3] If all goes well, you will see "`250 2.1.0 Sender ok`".

4. Next: who is the recipient? For starters, try sending an email message to yourself. Type `RCPT TO: sunetid@stanford.edu ↵`. Replace *sunetid* with your own SUNet ID. If all goes well, you will see "`250 2.1.5 Recipient ok.`"

5. It's time to upload the email message itself. Type `DATA ↵` to tell the server you're ready to start. If all goes well, you will see "`354 End data with <CR><LF>.<CR><LF>`".

6. Now you are typing an email message to yourself. First, start by typing the *headers* that you will see in your email client. Leave a blank line at the end of the headers.

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

[2]These instructions might also work from outside Stanford's network, but we can't guarantee it.

[3]Yes, it's possible to give a phony "from" address. Electronic mail is a bit like real mail from the postal service, in that the accuracy of the return address is (mostly) on the honor system. You can write anything you like as the return address on a postcard, and the same is largely true of email. Please do not abuse this—seriously. With engineering knowledge comes responsibility! Sending email with a phony "from" address is commonly done by spammers and criminals so they can pretend to be somebody else. It's fun to play around with this and pretend to be `santaclaus@northpole.gov`, but **make sure you don't deceive any recipient**. And: even if the recipient is in on the joke, **do not send email pretending to be any Stanford employee** (otherwise you may set off the university's IT security alerts).

3.2 发送一封电子邮件给自己

现在你已经知道如何获取网页了，是时候发送电子邮件消息了，再次使用可靠的字节流发送到另一台计算机上运行的服务。

1. SSH到sunetid@`cardinal.stanford.edu`（确保您连接到斯坦福的网络），然后运行 `telnet 67.231.149.169 smtp`.[2] "smtp"服务指的是简单邮件传输协议，用于发送电子邮件消息。如果一切顺利，您将看到：

```
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：让电子邮件服务器识别您的计算机。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：谁在发送邮件？输入 `MAIL FROM: sunetid@stanford.edu`。将 sunetid 替换为您的 SUNetID.[3] 如果一切顺利，您将看到"`250 2.1.0 Senderok`"。

4. 下一步：收件人是谁？首先，尝试将电子邮件消息发送给自己。输入 `RCPT TO: sunetid@stanford.edu`。将sunetid替换为你自己的SUNetID。如果一切顺利，你会看到"`250 2.1.5 Recipient ok。`"

5. 现在，请上传电子邮件消息本身。输入 `DATA` 以告知服务器您已准备好开始。如果一切顺利，您将看到"`354 End data with <CR><LF>.<CR><LF>`"。

6. 现在，您正在给自己写电子邮件消息。首先，开始输入标题 你将在你的电子邮件客户端中看到的内容。在标题的末尾留一个空行。

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

[2]这些说明可能也适用于斯坦福大学网络之外的环境，但我们不能保证这一点。[3]是的，可以设置虚假的"发件人"地址。电子邮件有点像邮政服务的普通邮件，在这一点上，回复地址的准确性（基本上）依赖于信誉。你可以在明信片上写任何你想要的内容作为回复地址，电子邮件在很大程度上也是如此。请勿滥用这一点——认真对待。拥有工程知识就意味着有责任！使用虚假的"发件人"地址发送电子邮件是垃圾邮件发送者和犯罪分子常用的手段，这样他们就可以冒充其他人。虽然可以假装自己是northpole.gov的santaclaus@northpole.gov来玩这个游戏，但要确保不要欺骗任何收件人。而且：即使收件人知道这是个玩笑，也请不要冒充任何斯坦福大学的员工（否则可能会触发大学的IT安全警报）。

7. Type the *body* of the email message—anything you like. When finished, end with a dot on a line by itself: `. ⏎`. Expect to see something like: "`250 2.0.0 33h24dpdsr-1 Message accepted for delivery`".

8. Type `QUIT ⏎` to end the conversation with the email server. Check your inbox and spam folder to make sure you got the email.

3.3 Listening and connecting

You've seen what you can do with `telnet`: a **client** program that makes outgoing connections to programs running on other computers. Now it's time to experiment with being a simple **server**: the kind of program that waits around for clients to connect to it.

1. In one terminal window, run `netcat -v -l -p 9090` **on your VM**. You should see:

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

2. Leave `netcat` running. In another terminal window, run `telnet localhost 9090` (also on your VM).

3. If all goes well, the `netcat` will have printed something like "`Connection from localhost 53500 received!`".

4. Now try typing in either terminal window—the `netcat` (server) or the `telnet` (client). Notice that anything you type in one window appears in the other, and vice versa. You'll have to hit ⏎ for bytes to be transfered.

5. In the `netcat` window, quit the program by typing `ctrl-C`. Notice that the `telnet` program immediately quits as well.

4 Writing a network program using an OS stream socket

In the next part of this warmup lab, you will write a short program that fetches a Web page over the Internet. You will make use of a feature provided by the Linux kernel, and by most other operating systems: the ability to create a *reliable bidirectional byte stream* between two programs, one running on your computer, and the other on a different computer across the Internet (e.g., a Web server such as Apache or nginx, or the `netcat` program).

This feature is known as a *stream socket*. To your program and to the Web server, the socket looks like an ordinary file descriptor (similar to a file on disk, or to the `stdin` or `stdout`

7. 输入电子邮件消息正文——任何你喜欢的文本。完成后，在单独的一行末尾输入一个点：。。预期会看到类似 "`250 2.0.0 33h24dpdsr-1Message accepted for delivery`" 的内容。

8. 输入 `QUIT ⏎` 以结束与电子邮件服务器的对话。检查你的收件箱和垃圾邮件文件夹，以确保你收到了邮件。

3.3 倾听与连接

你已经看到了 `telnet` 能做什么：一个向运行在其他计算机上的程序发起连接的客户程序。现在，是时候尝试做一个简单的服务器了：那种等待客户端连接到它的程序。

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

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

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

3. 如果一切顺利，`netcat` 将会打印类似以下内容："`Connection from localhost 53500 received!`"

4. 现在试着在任一终端窗口中输入—— `netcat` (服务器)或 `telnet` (客户端)。请注意，你在其中一个窗口中输入的任何内容都会在另一个窗口中显示，反之亦然。你将不得不按键以进行字节传输。

5. 在 `netcat` 窗口中，通过键入来退出程序 `ctrl-C`。请注意， `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

在本次热身实验的下一部分，你将编写一个简短的程序，通过互联网获取一个网页。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一个可靠的双向字节流，其中一个程序运行在你的计算机上，另一个程序运行在互联网上另一台计算机上（例如，一个 Apache 或 nginx 的网页服务器，或 `netcat` 程序）。

此功能称为流套接字。对您的程序和 Web 服务器而言，套接字看起来像是一个普通的文件描述符（类似于磁盘上的文件，或类似于 `stdin` 或 `stdout`）

I/O streams). When two stream sockets are *connected*, any bytes written to one socket will eventually come out in the same order from the other socket on the other computer.

In reality, however, the Internet doesn't provide a service of reliable byte-streams. Instead, the only thing the Internet really does is to give its "best effort" to deliver short pieces of data, called *Internet datagrams*, to their destination. Each datagram contains some metadata (headers) that specifies things like the source and destination addresses—what computer it came from, and what computer it's headed towards—as well as some *payload* data (up to about 1,500 bytes) to be delivered to the destination computer.

Although the network tries to deliver every datagram, in practice datagrams can be (1) lost, (2) delivered out of order, (3) delivered with the contents altered, or even (4) duplicated and delivered more than once. It's normally the job of the operating systems on either end of the connection to turn "best-effort datagrams" (the abstraction the Internet provides) into "reliable byte streams" (the abstraction that applications usually want).

The two computers have to cooperate to make sure that each byte in the stream eventually gets delivered, in its proper place in line, to the stream socket on the other side. They also have to tell each other how much data they are prepared to accept from the other computer, and make sure not to send more than the other side is willing to accept. All this is done using an agreed-upon scheme that was set down in 1981, called the Transmission Control Protocol, or TCP.

In this lab, you will simply use the operating system's pre-existing support for the Transmission Control Protocol. You'll write a program called "`webget`" that creates a TCP stream socket, connects to a Web server, and fetches a page—much as you did earlier in this lab. In future labs, you'll implement the other side of this abstraction, by implementing the Transmission Control Protocol yourself to create a reliable byte-stream out of not-so-reliable datagrams.

4.1 Let's get started—setting up the repository on your VM and on GitHub

1. The lab assignments will use a starter codebase called "Minnow." **On your VM**, run `git clone https://github.com/cs144/minnow` to fetch the source code for the lab.
2. Enter the minnow directory by typing: `cd minnow`
3. In a Web browser, you'll make a repository within your own GitHub account to hold your solutions to the lab assignment.
   (a) If you don't already have a GitHub account, please make one at https://github.com.
   (b) Navigate to https://github.com/new to create a new repository.
   (c) Name the repository "minnow" within your GitHub account.
   (d) *Make sure to set the repository to "Private" so your solutions are not public.*

I/O 流）。当两个流套接字连接时，写入其中一个套接字的任何字节最终都会以相同的顺序从另一台计算机上的另一个套接字中输出。

然而，在现实中，互联网并不提供可靠的字节流服务。相反，互联网真正做的是尽其所能将短数据片段（称为互联网数据报）传输到目的地。每个数据报包含一些元数据（头部），这些头部指定了诸如源地址和目标地址等信息——数据报来自哪台计算机，以及它将传输到哪台计算机——以及一些要传输到目标计算机的有效载荷数据（最多约1,500字节）。

尽管网络尝试传输每个数据报，但在实际中，数据报可能会（1）丢失，（2）顺序错乱，（3）内容被篡改，甚至（4）重复传输多次。通常，连接两端的操作系统的任务是"尽力而为的数据报"（互联网提供的抽象）转换为"可靠字节流"（应用程序通常需要的抽象）。

这两台计算机必须协同工作，以确保流中的每个字节最终都能按正确的顺序传递到另一边的流套接字。它们还需要相互告知对方愿意从另一台计算机接收多少数据，并确保发送的数据量不超过对方愿意接受的范围。所有这一切都是通过一个在1981年确定的、约定的方案来完成的，该方案称为传输控制协议，或TCP。

在本实验中，你将简单地使用操作系统对传输控制协议的预置支持。你将编写一个名为"`webget`"的程序，该程序创建一个TCP流套接字，连接到Web服务器并获取页面——就像你在这之前在本实验中所做的那样。在未来的实验中，你将实现这一抽象的另一端，通过自己实现传输控制协议，将不可靠的数据报转化为可靠的字节流。

4.1 让我们开始——在你的虚拟机（VM）上和GitHub上设置仓库

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（VM）上，运行 `git clone https://github.com/cs144/minnow` 来获取实验的源代码。
2. 通过输入： `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，用于存放实验作业的解决方案。
   (a) 如果你还没有GitHub账户，请访问https://github.com创建一个。
   (b) 访问https://github.com/new来创建一个新的仓库。
   (c) 在你的GitHub账户中，将仓库命名为"minnow"。
   (d) 确保将仓库设置为 "私有"模式，这样你的解决方案就不会公开。

(e) Click “Create Repository”.

(f) On the next screen, click “Invite collaborators”, then “Add people”.

(g) Add “cs144-grader” as a collaborator (this will let us see and grade your code, while keeping it private).

4. Back on your VM, register the GitHub repository as a target by running the command: `git remote add github https://github.com/username/minnow` (replacing “username” with your actual GitHub username). This creates an association between your local copy of the lab assignment (on your VM) and your copy on GitHub (which you'll use to back up your local copy and be graded).

5. Run `git push github` to send the starter code to your GitHub repository. If all goes well, you will see a few lines of text printed, ending in: `* [new branch] main -> main`. If you see an error message, double-check that you have executed the above steps correctly. This command uploads *your* code to your private copy of the repository on GitHub and lets us grade your submissions.

4.2 Compiling the starter code

1. Still in the “minnow” directory, create a directory to compile the lab software: `cmake -S . -B build`

2. Compile the source code: `cmake --build build`

3. Using your favorite text editor (many students prefer VS Code editing files over SSH, but you can use whatever you want): open and start editing the `writeups/check0.md` file. This is the template for your lab checkpoint writeup and will be included in your submission.

4.3 Modern C++: mostly safe but still fast and low-level

CS144 is a programming-heavy class. The lab assignment is done in a contemporary C++ style that uses recent (2023) features to program as safely as possible. This might be different from how you have been asked to write C++ in the past. For references to this style, please see the C++ Core Guidelines (http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines).

The basic idea is to make sure that every object is designed to have the smallest possible public interface, has a lot of internal safety checks and is hard to use improperly, and knows how to clean up after itself. We want to avoid “paired” operations (e.g. malloc/free, or new/delete), where it might be possible for the second half of the pair not to happen (e.g., if a function returns early or throws an exception). Instead, operations happen in the constructor to an object, and the opposite operation happens in the destructor. This style is called “Resource acquisition is initialization,” or RAII.

(e) 点击“创建仓库”。

(f) 在下一个界面中，点击“邀请协作者”，然后“添加人员”。

(g) 将“cs144-grader”添加为协作者（这将让我们查看和评分你的代码，同时保持其私密）。

4. 在你的虚拟机（VM）上，通过运行命令将 GitHub 仓库注册为目标（将 “user-name” 替换为你的实际 GitHub 用户名）。这会在你的本地实验作业副本（位于虚拟机上）和你的 GitHub 副本（用于备份本地副本和评分）之间建立关联。

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，你会看到几行文本打印出来，以 “ `* [new branch] main -> main` ” 结尾。如果你看到错误消息，请仔细检查你是否正确执行了上述步骤。该命令将你的代码上传到 GitHub 上的私有仓库副本，并允许我们评分你的作业。

4.2 编译初始代码

1. 仍然在“小鱼”目录中，创建一个用于编译实验软件的目录： `cmake -S . -B build`

2. 编译源代码： `cmake --build build`

3. 使用你最喜欢的文本编辑器（许多学生更喜欢通过 SSH 编辑 VS Code 文件，但你可以使用任何你想要的）：打开并开始编辑 `writeups/check0.md` 文件。这是你的实验检查点报告的模板，并将包含在你的提交中。

4.3 现代C++：基本安全但依然快速且底层

CS144是一门编程密集型课程。实验作业采用当代C++风格编写，使用最新的（2023年）特性尽可能安全地编程。这可能与过去要求你编写C++ 的方式有所不同。关于这种风格的参考，请参阅C ++ 核心指南（http://isocpp.github.io/CppCoreGuidelines/CppCoreGuidelines）。

基本思想是确保每个对象都设计为具有尽可能小的公共接口，包含大量内部安全检查且难以误用，并且知道如何清理自身。我们希望避免“配对”操作（例如 malloc/free 或 new/delete ），其中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，而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”，或 RAII。

In particular, we would like you to:

- Use the language documentation at https://en.cppreference.com as a resource. (We'd recommend you avoid `cplusplus.com` which is more likely to be out-of-date.)
- Never use `malloc()` or `free()`.
- Never use **new** or **delete**.
- Essentially never use raw pointers (`*`), and use "smart" pointers (`unique_ptr` or `shared_ptr`) only when necessary. (You will not need to use these in CS144.)
- Avoid templates, threads, locks, and virtual functions. (You will not need to use these in CS144.)
- Avoid C-style strings (`char *str`) or string functions (`strlen()`, `strcpy()`). These are pretty error-prone. Use a `std::string` instead.
- Never use C-style casts (e.g., `(FILE *)x`). Use a C++ `static_cast` if you have to (you generally will not need this in CS144).
- Prefer passing function arguments by `const` reference (e.g.: `const Address & address`).
- Make every variable `const` unless it needs to be mutated.
- Make every method `const` unless it needs to mutate the object.
- Avoid global variables, and give every variable the smallest scope possible.
- Before handing in an assignment, run `cmake --build build --target tidy` for suggestions on how to improve the code related to C++ programming practices, and `cmake --build build --target format` to format the code consistently.

On using Git: The labs are distributed as Git (version control) repositories—a way of documenting changes, checkpointing versions to help with debugging, and tracking the provenance of source code. **Please make frequent small commits as you work, and use commit messages that identify what changed and why.** The Platonic ideal is that each commit should compile and should move steadily towards more and more tests passing. Making small "semantic" commits helps with debugging (it's much easier to debug if each commit compiles and the message describes one clear thing that the commit does) and protects you against claims of cheating by documenting your steady progress over time—and it's a useful skill that will help in any career that includes software development. The graders will be reading your commit messages to understand how you developed your solutions to the labs. If you haven't learned how to use Git, please do ask for help at the CS144 office hours or consult a tutorial (e.g., https://guides.github.com/introduction/git-handbook). Finally, while we ask you to back your code up and submit your code to us by using a **private** repository on GitHub, please **make sure your code is not publicly accessible.**

To repeat (because we have taught this class before): make frequent small commits as you work, and use commit messages that identify what changed and why.

特别是，我们希望你能：

- 以 https://en.cppreference.com 的语言文档作为参考资源。(我们建议你避免 `cplusplus.com`，因为它更有可能已经过时。)
- 永远不要使用 `malloc()` 或 `free()`。
- 永远不要使用 new 或 delete。
- 基本上永远不要使用原始指针 (*)，并且仅在必要时使用“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。)
- 避免使用 C 风格字符串 (`char *str`) 或字符串函数 (`strlen()`, `strcpy()`)。这些很容易出错。使用一个 `std::string` 代替。
- 永远不要使用 C 风格强制转换 (例如， `(FILE *)x`)。如果你必须使用 C++ `static cast` (通常在 CS144 中不需要这个)，请使用它。
- 优先通过 `const`引用传递函数参数 (例如： `const Address & address`)。
- 让每个变量 `const`，除非它需要被修改。
- 让每个方法 `const`，除非它需要修改对象。
- 避免使用全局变量，并给每个变量最小的作用域。
- 在提交作业前，运行 `cmake --build build --target tidy` 获取关于改进与 C++ 编程实践相关的代码的建议，以及 `cmake --build build --target format` 统一格式化代码。

关于使用 Git：实验资料以 Git（版本控制）仓库形式分发——这是一种记录变更、检查点版本以帮助调试以及追踪源代码来源的方式。请边工作边频繁提交小改动，并使用能标识变更内容和原因的提交信息。柏拉图理想是每个提交都应该能编译，并且稳步朝着更多测试通过的方向前进。提交小的“语义”变更有助于调试（如果每个提交都能编译且信息描述一个清晰的功能点，调试会容易得多），并且通过记录随时间稳步的进展来保护你免受作弊指控——这是一种包含软件开发职业的任何领域都很有用的技能。评分员会阅读你的提交信息以了解你如何开发实验解决方案。如果你还没学会使用 Git，请务必在 CS144 办公时间寻求帮助或查阅教程（例如，https://guides.github.com/introduction/git-handbook）。最后，虽然我们要求你使用 GitHub 上的私有仓库备份并提交代码，但请确保你的代码不是公开访问的。

要重复（因为我们之前教过这门课）：工作时频繁提交小改动，并使用能说明改动内容和原因的提交信息。

4.4 Reading the Minnow support code

To support this style of programming, Minnow's classes wrap operating-system functions (which can be called from C) in "modern" C++. We have provided you with C++ wrappers for concepts we hope you're broadly familiar with from CS 111, especially sockets and file descriptors.

Please read over the public interfaces (the part that comes after "`public:`" in the files `util/socket.hh` and `util/file_descriptor.hh`. (Please note that a `Socket` is a type of `FileDescriptor`, and a `TCPSocket` is a type of `Socket`.)

4.5 Writing `webget`

It's time to implement `webget`, a program to fetch Web pages over the Internet using the operating system's TCP support and stream-socket abstraction—just like you did by hand earlier in this lab.

1. From the `build` directory, open the file `../apps/webget.cc` in a text editor or IDE.
2. In the **get_URL** function, implement the simple Web client as described in this file, using the format of an HTTP (Web) request that you used earlier. Use the `TCPSocket` and `Address` classes.
3. Hints:
   - Please note that in HTTP, each line must be ended with "\r\n" (it's not sufficient to use just "\n" or `endl`).
   - Don't forget to include the "Connection: close" line in your client's request. This tells the server that it shouldn't wait around for your client to send any more requests after this one. Instead, the server will send one reply and then will immediately end its outgoing bytestream (the one *from* the server's socket *to* your socket). You'll discover that your incoming byte stream has ended because your socket will reach "EOF" (end of file) when you have read the entire byte stream coming from the server. That's how your client will know that the server has finished its reply.
   - Make sure to read and print *all* the output from the server until the socket reaches "EOF" (end of file)—**a single call to `read` is not enough.**
   - We expect you'll need to write about eight lines of code.
4. Compile your program by running `cmake --build build`. If you see an error message, you will need to fix it before continuing.
5. Test your program by running `./apps/webget cs144.keithw.org /hello`. How does this compare to what you see when visiting http://cs144.keithw.org/hello in a

4.4 阅读Minnowsupport代码

为了支持这种编程风格，Minnow 的课程将操作系统函数（可以从 C 语言调用）封装在“现代”C++中。我们已经为你提供了 C++ 封装，用于我们希望你在 CS 111 中广泛熟悉的那些概念，特别是套接字和文件描述符。

请阅读公共接口（文件中“`public:`”之后的部分`util/socket.hh` 和 `util/file descriptor.hh`）。（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，这是一个使用操作系统提供的TCP支持和流套接字抽象来从互联网获取网页的程序——就像你之前手动完成这个实验时做的那样。

1. 从 `build` 目录中，使用文本编辑器或 IDE 打开文件 `../apps/webget.cc` 。
2. 在 get URL 函数中，按照本文件中描述的方式实现简单的 Web 客户端，使用之前使用的 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）。
   - 别忘了在客户端请求中包含“Connection: close”行。这告诉服务器它不需要等待客户端在此请求之后发送任何更多请求。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您的套接字）。您会发现您的入站字节流已结束，因为当您读取完从服务器发送的整个字节流时，您的套接字将达到“EOF”（文件结束）。这就是您的客户端如何知道服务器已完成其回复的方式。
   - 确保读取并打印服务器所有输出，直到套接字达到“EOF”（文件结束）——仅调用一次 `read` 是不够的。
   - 我们预计你需要编写大约八行代码。
4. 通过运行 `cmake --build build` 编译你的程序。如果你看到错误消息，你需要先修复它才能继续。
5. 通过运行`./apps/webget cs144.keithw.org /hello` 来测试你的程序。这与你访问 http://cs144.keithw.org/hello 时看到的内容相比如何?

Web browser? How does it compare to the results from Section 3.1? Feel free to experiment—test it with any `http` URL you like!

6. When it seems to be working properly, run `cmake --build build --target check_webget` to run the automated test. Before implementing the **get_URL** function, you should expect to see the following:

```
$ 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
```

After completing the assignment, you will see:

```
$ 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. The graders will run your `webget` program with a different hostname and path than `make check_webget` runs—so make sure it doesn't *only* work with the hostname and path used by the unit tests.

5 An in-memory reliable byte stream

By now, you've seen how the abstraction of a *reliable byte stream* can be useful in communicating across the Internet, even though the Internet itself only provides the service of "best-effort" (unreliable) datagrams.

To finish off this week's lab, you will implement, in memory on a single computer, an object that provides this abstraction. (You may have done something similar in CS 110/111.) Bytes are written on the "input" side and can be read, in the same sequence, from the "output" side. The byte stream is finite: the writer can end the input, and then no more bytes can be written. When the reader has read to the end of the stream, it will reach "EOF" (end of file) and no more bytes can be read.

网页浏览器？它与第3.1节的结果相比如何？请随意尝试——用你喜欢的任何 `http` 网址来测试它！

6. 当它似乎运行正常时，运行 `cmake --build build --target check webget` 来运行自动测试。在实现获取网址功能之前，你应该预期看到以下内容：

```
$ 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，你会看到：

```
$ 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. 评分员将使用与 `make check webget` 运行不同的主机名和路径来运行你的 `webget` 程序——因此确保它不仅能在单元测试使用的主机名和路径下工作。

5 内存可靠字节流

到目前为止，你已经看到了可靠字节流抽象在跨互联网通信中的用途，尽管互联网本身只提供"尽力而为"（不可靠）数据报的服务。

为了完成本周的实验，你将实现一个对象，该对象在单台计算机的内存中提供这种抽象。（你可能已经在 CS 110/111 中做过类似的事情。）字节在"输入"端写入，并且可以按相同顺序从"输出"端读取。字节流是有限的：写入者可以结束输入，然后就不能再写入更多字节。当读取者读取到流的末尾时，它将到达"EOF"（文件结束）并且不能再读取更多字节。

Your byte stream will also be *flow-controlled* to limit its memory consumption at any given time. The object is initialized with a particular "capacity": the maximum number of bytes it's willing to store in its own memory at any given point. The byte stream will limit the writer in how much it can write at any given moment, to make sure that the stream doesn't exceed its storage capacity. As the reader reads bytes and drains them from the stream, the writer is allowed to write more. Your byte stream is for use in a *single* thread—you don't have to worry about concurrent writers/readers, locking, or race conditions.

To be clear: the byte stream is finite, but it can be *almost arbitrarily long*[4] before the writer ends the input and finishes the stream. Your implementation must be able to handle streams that are much longer than the capacity. The capacity limits the number of bytes that are held in memory (written but not yet read) at a given point, but does not limit the length of the stream. An object with a capacity of only one byte could still carry a stream that is terabytes and terabytes long, as long as the writer keeps writing one byte at a time and the reader reads each byte before the writer is allowed to write the next byte.

Here's what the interface looks like for the writer:

```
void push( std::string data ); // Push data to stream, but only as much as available capacity allows.
void close();     // Signal that the stream has reached its ending. Nothing more will be written.

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
```

And here is the interface for the reader:

```
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
```

Please open the `src/byte_stream.hh` and `src/byte_stream.cc` files, and implement an object that provides this interface. As you develop your byte stream implementation, you can run the automated tests with `cmake --build build --target check0`.

If all tests pass, the `check0` test will then run a speed benchmark of your implementation. Anything **faster than** `0.1 Gbit/s` (in other words, 100 million bits per second) is acceptable for purposes of this class, for the three pop lengths tested. (It is possible for an implementation to perform faster than 10 Gbit/s, but this depends on the speed of your computer and is not required.)

For any late-breaking questions, please check out the lab FAQ on the course website or ask your classmates or the teaching staff in the lab session (or on EdStem).

[4]At least up to 2^{64} bytes, which in this class we will regard as essentially arbitrarily long

你的字节流还将进行流量控制，以限制其在任何给定时间点的内存消耗。该对象使用特定的“容量”初始化：即它在任何给定时间点愿意存储在自身内存中的最大字节数。字节流将限制写入者在任何给定时刻可以写入的量，以确保流不会超过其存储容量。随着读取者读取字节并将它们从流中消耗掉，写入者被允许写入更多字节。你的字节流用于单线程——你不必担心并发写入者/读取者、锁定或竞态条件。

要明确：字节流是有限的，但在写入者结束输入并完成流之前，它可以几乎任意长[4]。您的实现必须能够处理远超容量的流。容量限制了在特定时刻内存中持有的字节数量（已写入但尚未读取），但并不限制流的长度。一个容量仅为1个字节的对象仍然可以承载一个长达TB级别的流，只要写入者一次只写入1个字节，并且读取者在允许写入者写入下一个字节之前读取每个字节。

这是为作者设计的界面看起来是这样的：

```
void push( std::string data ); // Push data to stream, but only as much as available capacity allows.
void close();     // Signal that the stream has reached its ending. Nothing more will be written.

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
```

并且这里是读者的界面：

```
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，并实现一个提供此接口的对象。在您开发字节流实现时，您可以使用 `cmake --build build --target check0` 运行自动化测试。

如果所有测试都通过， `check0` 测试将随后对你的实现运行速度基准测试 . 任何比 `0.1 Gbit/s` 更快（换句话说，每秒1亿比特）的速度对于本课程的目的都是可接受的，针对测试的三个pop长度。（实现可能比10 Gbit/s运行得更快，但这取决于你的计算机速度，并且不是必需的。）

对于任何突发问题，请查阅课程网站上的实验室常见问题解答，或向实验室的同学或教学人员（或在 EdStem 上）咨询。

[4]至少长达 2^{64} 字节，在本课程中我们将将其视为本质上任意长

What's next? Over the next four weeks, you'll implement a system to provide the same interface, no longer in memory, but instead over an unreliable network. This is the Transmission Control Protocol—and its implementations are arguably the **most prevalent computer program in the world**.

6 Submit

1. In your submission, please only make changes to `webget.cc` and the source code in the top level of `src` (`byte_stream.hh` and `byte_stream.cc`). Please don't modify any of the tests or the helpers in `util`.
2. Remember to make small commits as you code, with good commit messages. After making a commit, back up your VM's repository to your private GitHub repository often by running `git push github`. Your code needs to be committed and pushed to GitHub for it to be gradable.
3. 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 check0` (to make sure the automated tests pass)
   (d) Optional: `cmake --build build --target tidy` (suggests improvements to follow good C++ programming practices)
4. Finish editing `writeups/check0.md`, filling in the number of hours this assignment took you and any other comments.
5. Make sure your code is committed and pushed to your private GitHub repository (`git push github`).
6. There will be a Gradescope assignment due Sunday 11:59 p.m. for you to submit the commit ID of your submission.
7. Please let the course staff know ASAP of any problems at the Wednesday lab session, or by posting a question on EdStem. Good luck and welcome to CS144!

接下来是什么？在未来四周内，你将实现一个系统，提供相同的界面，不再是在内存中，而是在不可靠的网络之上。这是传输控制协议——其实现可以说是世界上最常见的计算机程序。

6 提交

1. 在你的提交中，请仅修改 `webget.cc` 和位于 `src` 顶层目录中的源代码 (`byte stream.hh` 和 `byte stream.cc`)。请不要修改 `util` 中的任何测试或辅助文件。
2. 请记住在编码时提交小的更改，并使用良好的提交信息。提交后，请通过运行 `git push github` 经常将你的 VM 的仓库备份到你的私有 GitHub 仓库。你的代码需要提交并推送到 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 check0` (以确保自动测试通过)
   (d) 可选：`cmake --build build --target tidy` (建议对遵循良好的 C++ 编程实践)
4. 完成 `writeups/check0.md` 的编辑，填写本次作业的已将你和其他任何评论都包含在内。
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. 将有一个 Gradescope 作业于周日 23:59 截止，你需要提交你的提交的 commit ID。
7. 请在周三的实验课或通过 EdStem 发布问题的方式尽快通知课程工作人员任何问题。祝你好运，欢迎加入 CS144！