1. Autograding testscases for 85% credit to be added in the next week. The remaining 15% are from testcases for error handling I keep to myself.

I provide either parsers for the command lines you havee to handle in either C (old school lex/yacc) or C++ (boost spirit x3 parser) started. You have to use one of these two parsers.

Advice:

1. Start early and ask loits of questions
2. Commit early and often to github - one advantage of this is that if you run into problems, I can (and will!) check out your code, look at it, and provide you advice
3. If you've tried to solve a problem and are just stuck *ask me for help!*

Additional Tips:

1. Start by uUnderstanding what fork(), wait(), and exec() do: https://pages.cs.wisc.edu/~remzi/OSTEP/cpu-api.pdf
2. Consider carefully the difference between a shell 'builtin' and a shell command that starts a new process. A 'builtin' is somethign that *looks* like starting a new process but actually only modifies the internal state of the shell. E.G., 'quit' doesn't start a new process, it just directs the shell to call the 'exit()' function. Is 'alias' a builtin or does it start a new process? What about 'cd?
3. The UNIX environment is just an array (ponited to by char *envp[]) that is initialized by process creation using data passed to execve(). Your shell doesn't need to use the environment passed in, because it maintains its own environment ('setenv' is a builtin!), and manages the environment of the processes it creates by controlling the arguments to execve().
4. For file redirection, you're going to use the c system call file interface, not the C++ stl (iostream) or C library (fopen/printf) interface. Remember that stdin is by definition file descriptor 0, stdout is file descriptor 1, and stderr is file descriptor 2. So how can I change the file pointed to by stdin, stdout, and stderr in a child process I create?

Still working on sorting things out; holding off acceptingthe assignment for now is not a bad idea. Certainly don't craete a codespace yet.

1. **自动评分测试用例**将于下周新增，占总分的 85%。剩下的 15%来自我保留的错误处理测试用例。
   我提供了用于处理命令行的解析器，你可以选择使用 C 语言（传统的 lex/yacc）或 C++（boost spirit x3 解析器）。你必须使用这两种解析器之一。

   建议：

2. 尽早开始，并多问问题。
3. 早提交并频繁提交到 GitHub——这样做的一个好处是，如果你遇到问题，我可以（也会）查看你的代码，给你建议。
4. 如果你尝试解决问题但卡住了，直接来问我！

额外提示：

1. 先理解 fork()、wait()和 exec()的作用：参考资料
2. 仔细考虑 Shell **内建命令（builtin）** 与 **新进程启动命令** 的区别。**内建命令**看起来像是启动了新进程，但实际上只是修改了 Shell 的内部状态。例如，quit 并不会启动新进程，而只是让 Shell 调用 exit() 函数。那么，alias 是内建命令还是会启动新进程？cd 呢？
3. UNIX 环境变量只是一个数组（由 char *envp[]指向），它在进程创建时由 execve()传递的数据初始化。你的 Shell 不需要使用传入的环境变量，因为它会维护自己的环境（例如，setenv 是一个内建命令！）。你的 Shell 管理它创建的进程的环境变量，方法是控制传递给 execve() 的参数。
4. **对于文件重定向**，你需要使用 C 语言的**系统调用（system call）** 文件接口，而不是 C++ STL（iostream）或 C 标准库（fopen/printf）。记住，**标准输入**（stdin）默认是文件描述符 0，**标准输出**（stdout）是文件描述符 1，**标准错误**（stderr）是文件描述符 2。那么，在创建子进程时，如何更改 stdin、stdout 和 stderr 指向的文件？

目前仍在整理相关内容；**暂缓接受该作业是个不错的选择**。当然，**不要现在就创建 Codespace**。

Due Date

Overview

Your task is to implement---on your own---a basic Unix command shell. Your shell *must* use either the provided C (src-c/) or C++ parser (src/), and implement a subset of the features described in the included ISH manual page. Notably, you

do *not* need to implement job control or pipelines.

The basic features you need to implement are:

- Basic command execution
    - Correct shell prompt as described in the manual page
    - Run command with full command name
    - Run command with full command name and arguments
    - 'cd' builtin works (check with /bin/pwd)
    - quit builtin and EOF cause shell to exit properly and cleanly
- Envirionment and alias handling
    - Environment variables correctly passed to child
    - PATH searching works
    - PATH in wrong syntax handled well
    - printenv (setenv with no arguments) works
    - unsetenv works
    - Adding an alias works
    - Removing an alias works
- Miscellaneous
    - .ishrc executed properly
- File redirection
    - Redirect output to a simople file
    - Appending to a file works
    - Redirection of stdout and stderr works
    - Redirect input from a simple file
    - Ambiguous redirections are detected and reported
    - Redirection of both input and output works.
- Error handling
    - The shell should fail gracefully and report errors like csh does when

permissions or something else goes wrong doing thie things described above.

The features you do //not// need to implement that are described in the ISH man page are:

- Job control (the & separator, ^Z handling, and the bg/fg/jobs builtins)
- Pipelines

Starter Source Code and Programming Assignment Restrictions

Your shell should be written in C or C++, compile using cmake, and produce an executable named ish. I have provided a C++ parser for the subset of 'ish' assigned in this assignment in the src/ directory; the C++ starter code does *not* parse pipelines as they are not required for this assignment. In addition, I have also provided C starter code in the src-c director that you may use if you prefer. To use this code, change the CMakeLists.txt in the top-level directory to point to the src-c/ directory instead of the src/ to find its source code.

Your code must also be "clean" -- it should compile without warnings or unresolved references ion a standard UNIX system with the Boost C++ libraries installed. I will provide a GitHub Codespace with the appropriate setup for developing your program shortly after the assignment becomes available. I encourage you to use GitHub Codespaces for development, as if you have problems I can connect to your codespace and help you debug your program. Your programs are expected to be well organized and easy to read, as well as correct.

You //must//, use either the C++ or C parsers provided to implement your shell. You may modify these parsers if necessary (e.g. if you want to change the parser to detect builtin commands or add extra functionality to the command structure/class provided), but the general structure of the parser *must remain the same*. You may *not* use your own custom parser or one taken from elsewhere.

You must use the fork and execve C system calls to start new processes and the dup or dup2 command to handle file redirection. You may *not* use other variants of exec or the system call, their C++ equivalents, or any other mechanism for creating processes or redirecting file I/O.

Testing

Testcases that test correctness and constitute 85% of the grade of the shell will be provided approximately one week after the assignment is posted. Additional tests that test your shell error handling will be used for final testing ofyour program. A pull

request will be submitted to your github repository adding test features when these testcases are available.

Assignment submission

You will use GitHub classroom to submit a working program on or before the due date. Be sure to commit and push all of your changes to the main branch on your repository prior to the due date!

Supporting and Reference Materials

Before starting, you should become familiar with the Unix system calls defined in Section 2. of the UNIX manual There are also several library routines in Section 3 that provide convenient interfaces to some of the more cryptic system calls. However, you may *not* use the library routine system nor any of the routines prohibitied on the ish man page. Several chapters of the Richard Stevens book *Advanced Programming in the UNIX Environment* contain helpful information; Chapters 7, 8, and 9 are especially relevant, and this book is available online for free through the UNM library.

Additional Advice

To implement ish, you will be creating a process that forks off other processes, which in turn forks off more processes, etc. If you inadvertently fork too many processes, you will cause Unix to run out, making yourself and everyone else on the machine very unhappy. _Be careful about this.

If you are in doubt about the functionality of ish or how it should behave in a particular situation, model the behavior on that of csh. If you have specific questions about the project, ask in Discord.

A few final bits of advice. First, and most importantly, get started early; you almost certainly have a lot to learn before you can start implementing anything. Second, once you have a good understanding of what you are being asked to do, I strongly suggest that you develop a detailed design, implementation, and testing plan. My personal style is to get functionality working one step at a time, for example, processing of simple commands, then environment handling and PATH searching, then I/O redirection.

截止日期

截止日期：2025 年 3 月 3 日星期一，晚上 11:59

概述

你的任务是**独立**实现一个基本的 Unix 命令行 Shell。你的 Shell **必须**使用提供的 C（src-c/ 目录）或 C++ 解析器（src/ 目录），并实现 ISH 手册页中描述的一部分功能。需要注意的是，你**不需要**实现作业控制或管道（pipelines）。

你需要实现的基本功能包括：

基本命令执行

- Shell 提示符应正确显示，符合手册页描述
- 能够运行完整命令名称的命令
- 能够运行带有参数的完整命令名称的命令
- **cd** 内建命令可以正常工作（可用 /bin/pwd 进行检查）
- **quit** 内建命令和 EOF（文件结束符）能正确退出 Shell 并清理资源

环境变量和别名管理

- **环境变量**能够正确传递给子进程
- **PATH 变量**搜索功能正常
- **错误的 PATH 语法**能够被正确处理
- **printenv**（如果 setenv 无参数时）能够正确显示环境变量
- **unsetenv** 能够正确移除环境变量
- 能够正确添加别名
- 能够正确移除别名

其他功能

- **.ishrc** 文件能够正确执行

文件重定向

- 输出重定向到**一个普通文件**能够正常工作
- **追加模式**（appending to a file）能够正常工作
- **标准输出（stdout）和标准错误（stderr）** 的重定向能够正常工作
- **从文件中读取输入**能够正常工作
- **检测到模棱两可的重定向（ambiguous redirections）时应报错**
- **输入和输出同时重定向**能够正常工作

错误处理

- Shell 应该能够**优雅地处理错误**，并在权限或其他问题导致命令执行失败时，像 csh 那样正确报错

你不需要实现的功能（ISH 手册页中提到但不要求实现）

- **作业控制**（& 分隔符、^Z 信号处理，以及 bg / fg / jobs 内建命令）
- **管道（pipelines）**

起始代码和编程限制

你的 Shell **必须使用 C 或 C++ 编写**，并使用 cmake 进行编译，生成可执行文件 ish。

提供的 C++ 解析器（src/ 目录）能够解析 ish 需要实现的功能，但**不解析管道（因为本作业不要求实现管道）**。此外，还提供了 **C 版本的起始代码**（src-c/ 目录），你可以选择使用。如果要切换到 C 版本，你需要修改 CMakeLists.txt，让它指向 src-c/ 目录下的源代码。

你的代码必须满足以下要求：

- **干净整洁** —— 代码应当**无警告（warnings）**，并且在一个标准 UNIX 系统上能够正确编译（需要安装 Boost C++ 库）。
- **只能使用提供的解析器** —— 你**必须**使用提供的 C 或 C++ 解析器来实现 Shell。你可以**修改解析器**（例如添加对内建命令的检测、扩展命令结构等），但整体解析逻辑必须保持一致。你**不能**使用自己编写的解析器，也不能从其他地方获取解析器。
- **必须使用 fork() 和 execve()** 来启动新进程
- **必须使用 dup() 或 dup2()** 来处理文件重定向
- **禁止使用 system()** 或其他 exec 系列的变体，也不能使用它们的 C++ 版本，**必须直接使用 execve()**

测试

- 约 **85% 的最终成绩** 来自于官方提供的**自动评分测试用例**（将在作业发布约一周后提供）

- 其余 **15% 的成绩** 来自于**错误处理测试**（例如检测错误的命令、权限问题等）
- 当测试用例发布时，系统会提交一个 **pull request** 到你的 GitHub 仓库，以添加这些测试

作业提交

- 你需要使用 **GitHub Classroom** 来提交你的代码
- 在**截止日期前**，请确保**所有的修改都已提交并推送（commit & push）**到你的 GitHub 仓库的 main 分支

支持和参考资料

在开始之前，你应该先熟悉 UNIX 手册第 **2 章**（系统调用部分）以及 **第 3 章**（提供了一些便捷的库函数）。然而，你**不能**使用 system() 以及 ISH 手册页中禁止使用的其他函数。

此外，Richard Stevens 的书籍 **《Advanced Programming in the UNIX Environment》**（《UNIX 环境高级编程》）对本作业非常有帮助，**第 7、8、9 章尤其相关**。这本书可以通过 **UNM 图书馆**免费获取。

额外建议

- **小心进程管理**：你的 Shell 需要创建子进程，而子进程可能会继续创建更多的进程。如果你不小心创建了**太多进程**，可能会导致 Unix 资源耗尽，影响整个系统的运行。请务必**谨慎管理进程**，避免导致系统崩溃。
- **尽早开始**：这个作业涉及很多内容，理解清楚要求并提前规划是非常重要的。
- **设计实现计划**：在实现 Shell 之前，建议**先写详细的设计、实现和测试计划**，再开始编码。
- **逐步实现功能**：建议先实现最基础的功能，例如**解析和执行简单命令**，然后再依次添加**环境变量处理**、**PATH 解析**，最后实现**文件 I/O 重定向**等功能。
- **如果不确定行为**：可以参考 csh 的行为来设计你的 Shell。
- **如有问题**：可以在 **Discord** 上提问。

总结

本次作业的目标是让你用 C/C++ **实现一个基本的 Unix Shell**，重点在于**进程管理、环境变量、文件重定向和错误处理**。你需要使用 fork() 和 execve() 来启动进程，使用 dup() 或 dup2() 来管理 I/O 重定向，并使用提供的解析器完成命令解析。作业**不涉及作业控制和管道**。

你需要在 **GitHub 上提交代码**，并在**作业截止前 push 到主分支**。此外，官方测试用例将在作业发布一周后提供。