# 预学习准备

## 一生一芯通识问卷



## Linux系统安装与基本使用

### 课程概览与shell

课后练习

本课程中的每节课都包含一系列练习题。有些题目是有明确目的的，另外一些则是开放题，例如“尝试使用 X 和 Y”，我们强烈建议您一定要动手实践，用于尝试这些内容。 此外，我们没有为这些练习题提供答案。如果有任何困难，您可以发送邮件给我们并描述你已经做出的尝试，我们会设法帮您解答。

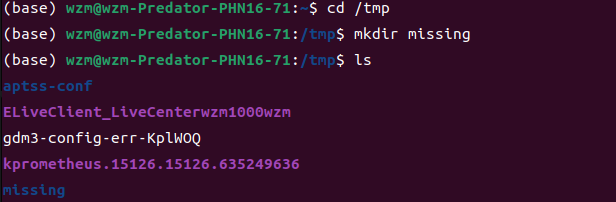
#### homework1

本课程需要使用类Unix shell，例如 Bash 或 ZSH。如果您在 Linux 或者 MacOS 上面完成本课程的练习，则不需要做任何特殊的操作。如果您使用的是 Windows，则您不应该使用 cmd 或是 Powershell；您可以使用Windows Subsystem for Linux或者是 Linux 虚拟机。使用echo $SHELL命令可以查看您的 shell 是否满足要求。如果打印结果为/bin/bash或/usr/bin/zsh则是可以的。



#### homework2

在 /tmp 下新建一个名为 missing 的文件夹。



#### homework3

用 man 查看程序 touch 的使用手册。



#### homework4

用 touch 在 missing 文件夹中新建一个叫 semester 的文件。



#### homework5

将以下内容一行一行地写入 semester 文件：

#!/bin/sh

curl --head --silent https://missing.csail.mit.edu

第一行可能有点棘手， # 在Bash中表示注释，而 ! 即使被双引号（"）包裹也具有特殊的含义。 单引号（'）则不一样，此处利用这一点解决输入问题。更多信息请参考 Bash quoting 手册





#### homework6

尝试执行这个文件。例如，将该脚本的路径（./semester）输入到您的shell中并回车。如果程序无法执行，请使用 ls 命令来获取信息并理解其不能执行的原因。



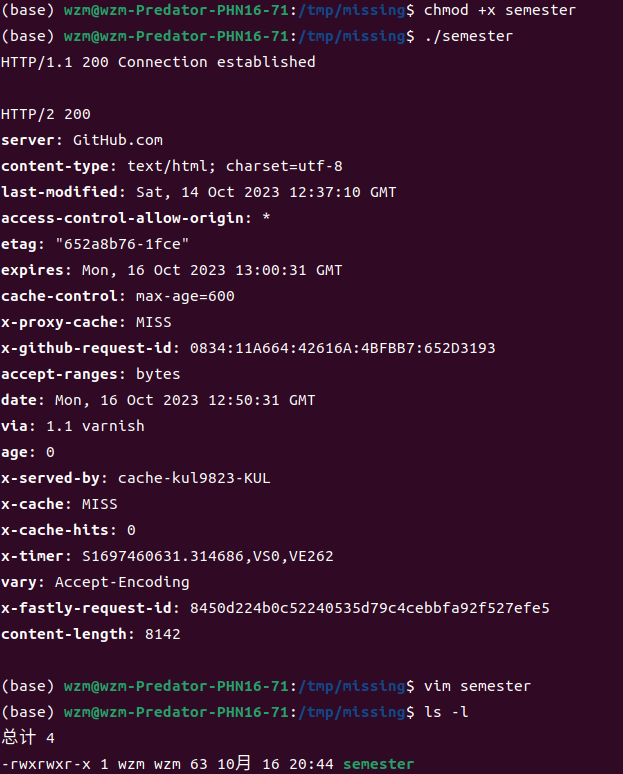
可以看到文件只有读写权限，没有执行权限，所以无法执行

#### homework7

查看 chmod 的手册(例如，使用 man chmod 命令)

#### homework8

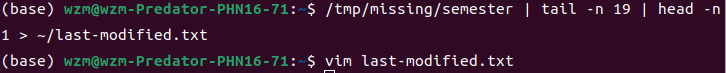
使用 chmod 命令改变权限，使 ./semester 能够成功执行，不要使用 sh semester 来执行该程序。您的 shell 是如何知晓这个文件需要使用 sh 来解析呢？更多信息请参考：shebang



文件添加执行权限后正常运行

#### homework9

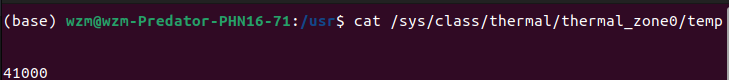
使用 | 和 > ，将 semester 文件输出的最后更改日期信息，写入主目录下的 last-modified.txt 的文件中

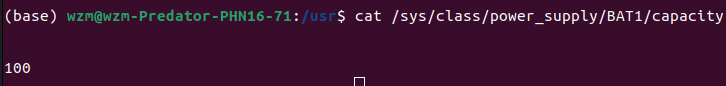




#### homework10

写一段命令来从 /sys 中获取笔记本的电量信息，或者台式机 CPU 的温度。注意：macOS 并没有 sysfs，所以 Mac 用户可以跳过这一题。





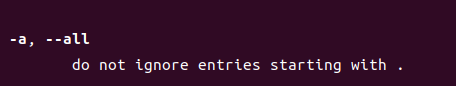
### shell工具和脚本

课后练习

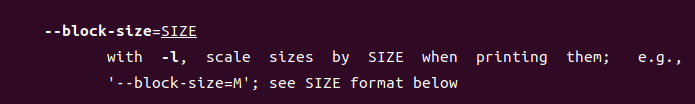
#### homework1

阅读 man ls ，然后使用ls 命令进行如下操作：

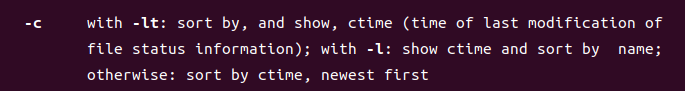
-所有文件（包括隐藏文件）



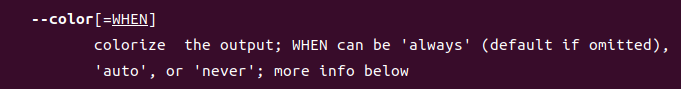
-文件打印以人类可以理解的格式输出 (例如，使用454M 而不是 454279954)



-文件以最近访问顺序排序



-以彩色文本显示输出结果



典型输出如下：

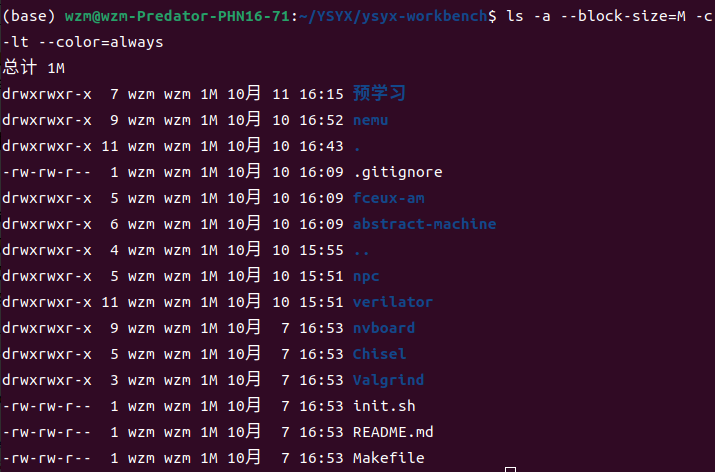
-rw-r--r-- 1 user group 1.1M Jan 14 09:53 baz

drwxr-xr-x 5 user group 160 Jan 14 09:53 .

-rw-r--r-- 1 user group 514 Jan 14 06:42 bar

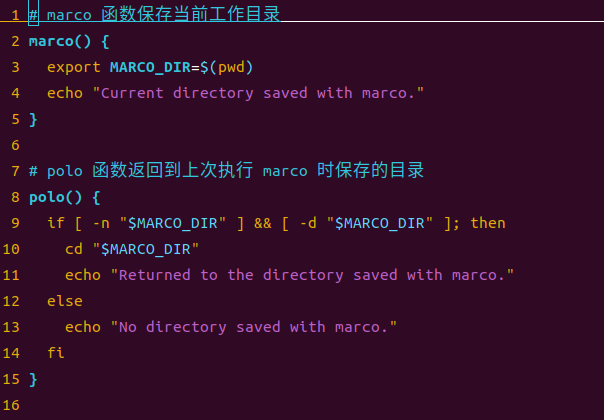
-rw-r--r-- 1 user group 106M Jan 13 12:12 foo

drwx------+ 47 user group 1.5K Jan 12 18:08 ..



#### homework2

编写两个bash函数 marco 和 polo 执行下面的操作.每当你执行 marco 时，当前的工作目录应当以某种形式保存，当执行 polo 时，无论现在处在什么目录下，都应当 cd 回到当时执行 marco 的目录。 为了方便debug，你可以把代码写在单独的文件 marco.sh 中，并通过 source marco.sh命令，（重新）加载函数。



#### homework3

假设您有一个命令，它很少出错。因此为了在出错时能够对其进行调试，需要花费大量的时间重现错误并捕获输出。 编写一段bash脚本，运行如下的脚本直到它出错，将它的标准输出和标准错误流记录到文件，并在最后输出所有内容。 加分项：报告脚本在失败前共运行了多少次。

#!/usr/bin/env bash

n=$(( RANDOM % 100 ))

if [[ n -eq 42 ]]; then

echo "Something went wrong"

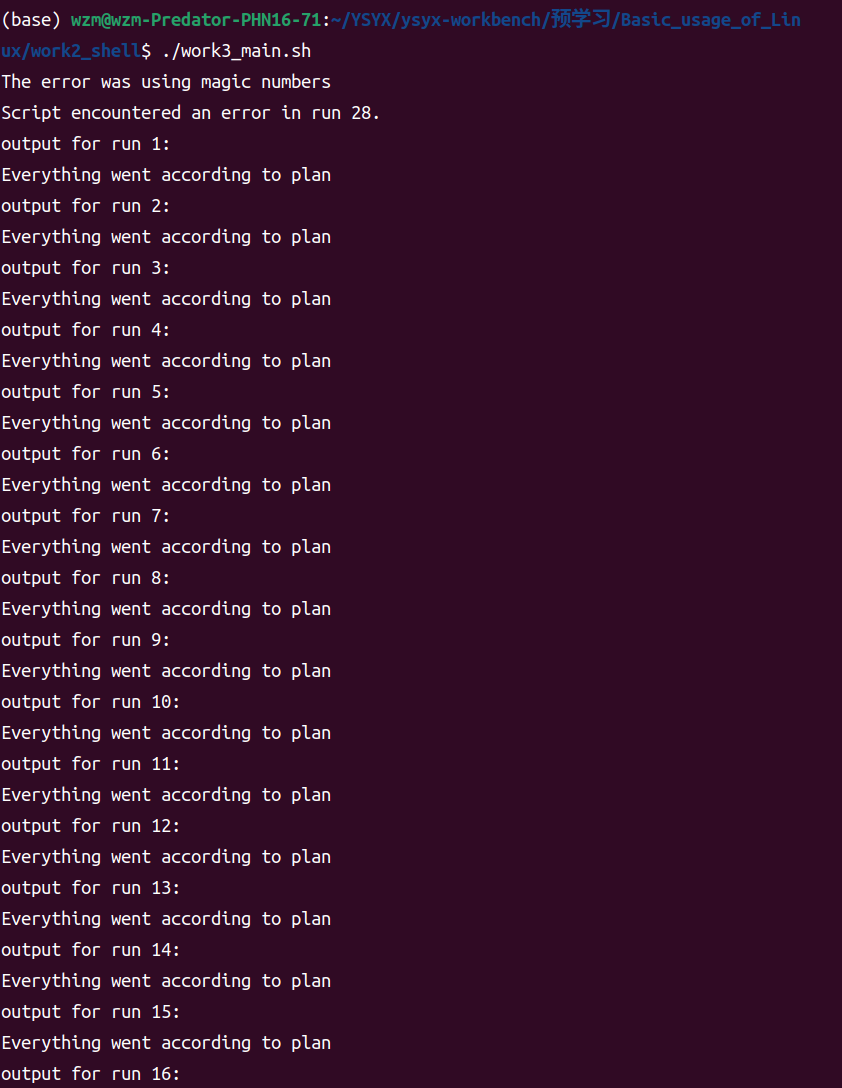
>&2 echo "The error was using magic numbers"

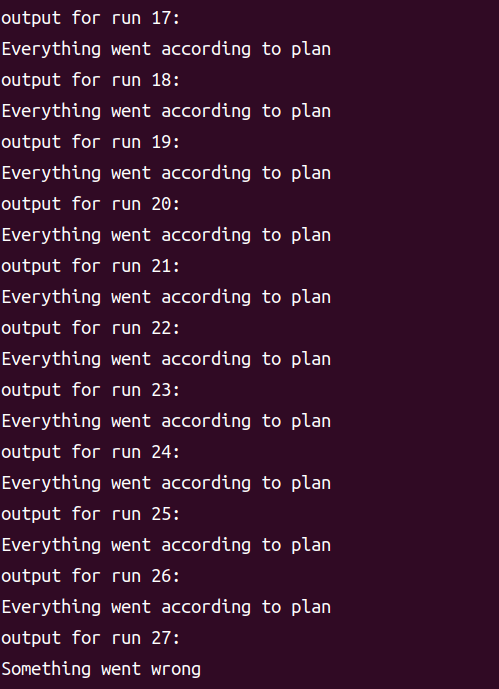
exit 1

fi

echo "Everything went according to plan"





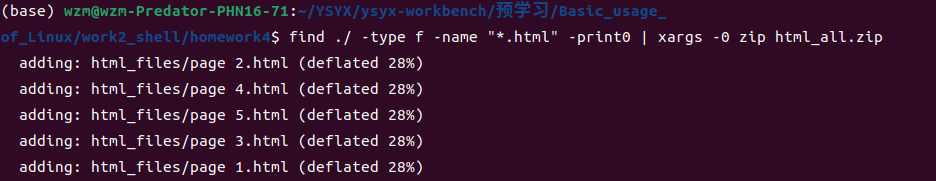


#### homework4

本节课我们讲解的 find 命令中的 -exec 参数非常强大，它可以对我们查找的文件进行操作。但是，如果我们要对所有文件进行操作呢？例如创建一个zip压缩文件？我们已经知道，命令行可以从参数或标准输入接受输入。在用管道连接命令时，我们将标准输出和标准输入连接起来，但是有些命令，例如tar 则需要从参数接受输入。这里我们可以使用xargs 命令，它可以使用标准输入中的内容作为参数。 例如 ls | xargs rm 会删除当前目录中的所有文件。

您的任务是编写一个命令，它可以递归地查找文件夹中所有的HTML文件，并将它们压缩成zip文件。注意，即使文件名中包含空格，您的命令也应该能够正确执行（提示：查看 xargs的参数-d，译注：MacOS 上的 xargs没有-d，查看这个issue）

如果您使用的是 MacOS，请注意默认的 BSD find 与 GNU coreutils 中的是不一样的。你可以为find添加-print0选项，并为xargs添加-0选项。作为 Mac 用户，您需要注意 mac 系统自带的命令行工具和 GNU 中对应的工具是有区别的；如果你想使用 GNU 版本的工具，也可以使用 brew 来安装。



#### homework5

（进阶）编写一个命令或脚本递归的查找文件夹中最近使用的文件。更通用的做法，你可以按照最近的使用时间列出文件吗？

感觉与homework1一样，暂不写。

### 编辑器（Vim）

#### homework1

完成 vimtutor。备注：它在一个 80x24（80 列，24 行） 终端窗口看起来效果最好。

（已完成）

#### homework2

下载我们提供的 vimrc，然后把它保存到 ~/.vimrc。 通读这个注释详细的文件 （用 Vim!）， 然后观察 Vim 在这个新的设置下看起来和使用起来有哪些细微的区别。

（已完成）

#### homework3

安装和配置一个插件： ctrlp.vim.

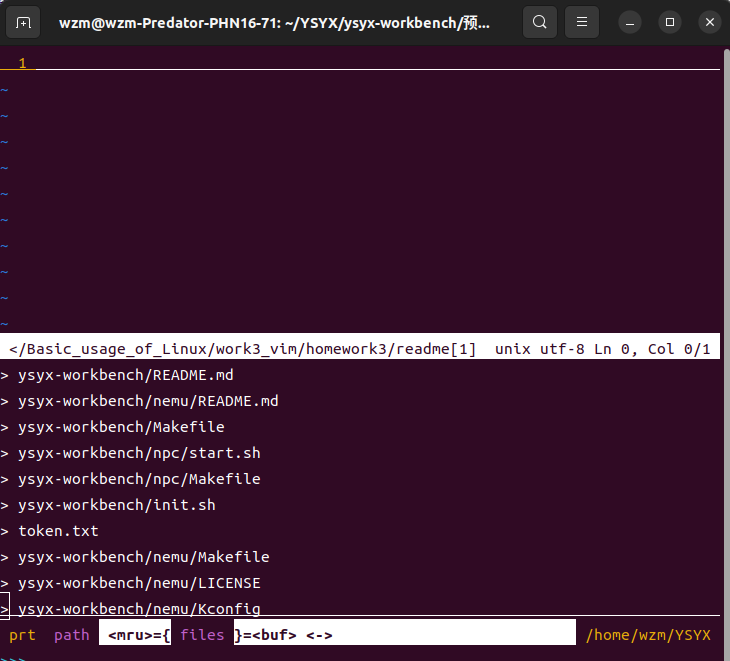
1.用 mkdir -p ~/.vim/pack/vendor/start 创建插件文件夹

2.下载这个插件： cd ~/.vim/pack/vendor/start;

git clone https://github.com/ctrlpvim/ctrlp.vim

3.阅读这个插件的 文档。 尝试用 CtrlP 来在一个工程文件夹里定位一个文件，打开 Vim, 然后用 Vim 命令控制行开始 :CtrlP.

4.自定义 CtrlP：添加 configuration 到你的 ~/.vimrc 来用按 Ctrl-P 打开 CtrlP



#### homework4

练习使用 Vim, 在你自己的机器上重做演示。

#### homework5

下个月用 Vim 完成所有的文件编辑。每当不够高效的时候，或者你感觉 “一定有一个更好的方式”时， 尝试求助搜索引擎，很有可能有一个更好的方式。如果你遇到难题，可以来我们的答疑时间或者给我们发邮件。

#### homework6

在其他工具中设置 Vim 快捷键 （见上面的操作指南）

#### homework7

进一步自定义你的 ~/.vimrc 和安装更多插件。

#### homework8

（高阶）用 Vim 宏将 XML 转换到 JSON (例子文件)。 尝试着先完全自己做，但是在你卡住的时候可以查看上面宏 章节。

### 数据整理

#### homework1

学习一下这篇简短的 交互式正则表达式教程.

（已完成）

#### homework2

统计words文件 (/usr/share/dict/words) 中包含至少三个a 且不以's 结尾的单词个数。这些单词中，出现频率前三的末尾两个字母是什么？ sed的 y命令，或者 tr 程序也许可以帮你解决大小写的问题。共存在多少种词尾两字母组合？还有一个很 有挑战性的问题：哪个组合从未出现过？

正则表达式：

grep -E '.\*a.\*a.\*a.\*[^s]$' /usr/share/dict/words | tr '[:upper:]' '[:lower:]' | awk '{print substr($0, length($0)-1, 2)}' | sort | uniq -c | sort -rn | head -3

--grep -E '.\*a.\*a.\*a.\*[^s]$' /usr/share/dict/words：这个命令使用正则表达式查找包含至少三个字母 "a" 且不以 "s" 结尾的单词。

--tr '[:upper:]' '[:lower:]'：这个命令将所有单词转换为小写字母，以确保大小写不会影响结果。

--awk '{print substr($0, length($0)-1, 2)}'：这个命令提取每个单词的末尾两个字母。

--sort：将提取的末尾两个字母排序。

--uniq -c：对排序后的末尾两个字母进行计数，以查看每个组合出现的次数。

--sort -rn：将结果按出现次数逆序排序，这样最频繁出现的组合将排在前面。

--head -3：最后，使用 head 命令获取前三个组合。



（挑战性任务：未完成）

#### homework3

进行原地替换听上去很有诱惑力，例如： sed s/REGEX/SUBSTITUTION/ input.txt > input.txt。但是这并不是一个明智的做法，为什么呢？还是说只有 sed是这样的? 查看 man sed 来完成这个问题

输出重定向会在打开文件之前截断或清空文件。这可能会导致数据丢失或文件内容被清空。

可以用sed -i重新写回原始文件

sed -i 's/REGEX/SUBSTITUTION/' input.txt

#### homework4

找出您最近十次开机的开机时间平均数、中位数和最长时间。在Linux上需要用到 journalctl ，而在 macOS 上使用 log show。找到每次起到开始和结束时的时间戳。在Linux上类似这样操作：

Logs begin at ...

和

systemd[577]: Startup finished in ...

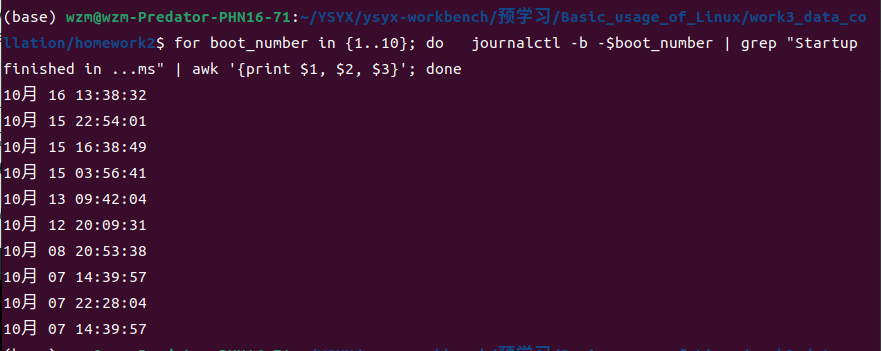
在 macOS 上, 查找:

=== system boot:

和

Previous shutdown cause: 5

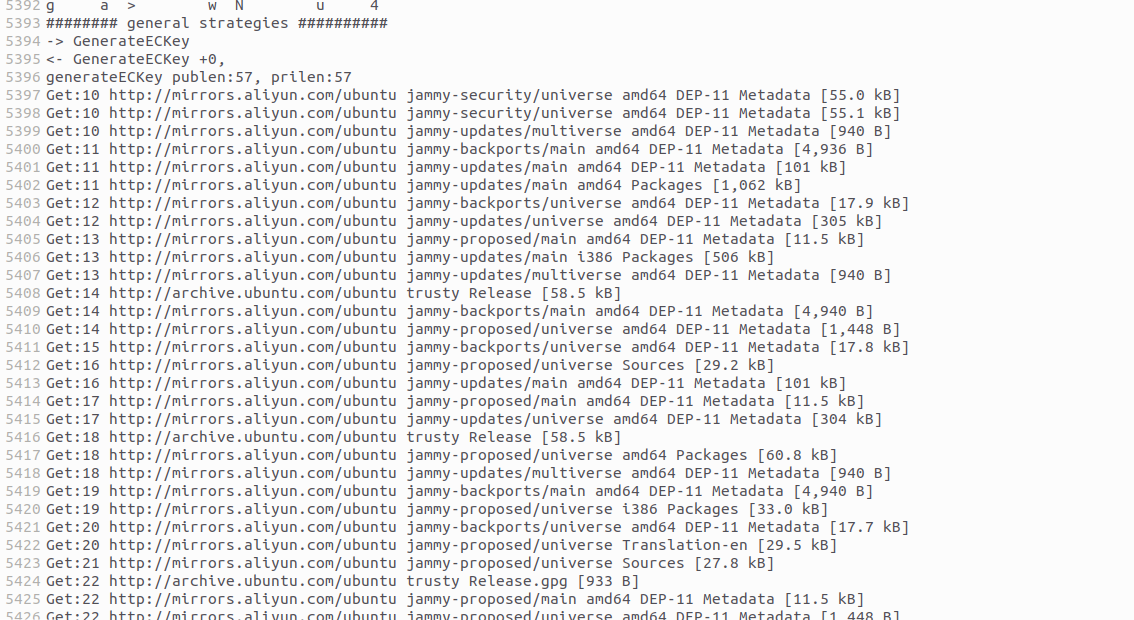
最近十次开机时间的时间戳





#### homework5

查看之前三次重启启动信息中不同的部分(参见 journalctl的-b 选项)。将这一任务分为几个步骤，首先获取之前三次启动的启动日志，也许获取启动日志的命令就有合适的选项可以帮助您提取前三次启动的日志，亦或者您可以使用sed '0,/STRING/d' 来删除STRING匹配到的字符串前面的全部内容。然后，过滤掉每次都不相同的部分，例如时间戳。下一步，重复记录输入行并对其计数(可以使用uniq )。最后，删除所有出现过3次的内容（因为这些内容是三次启动日志中的重复部分）。



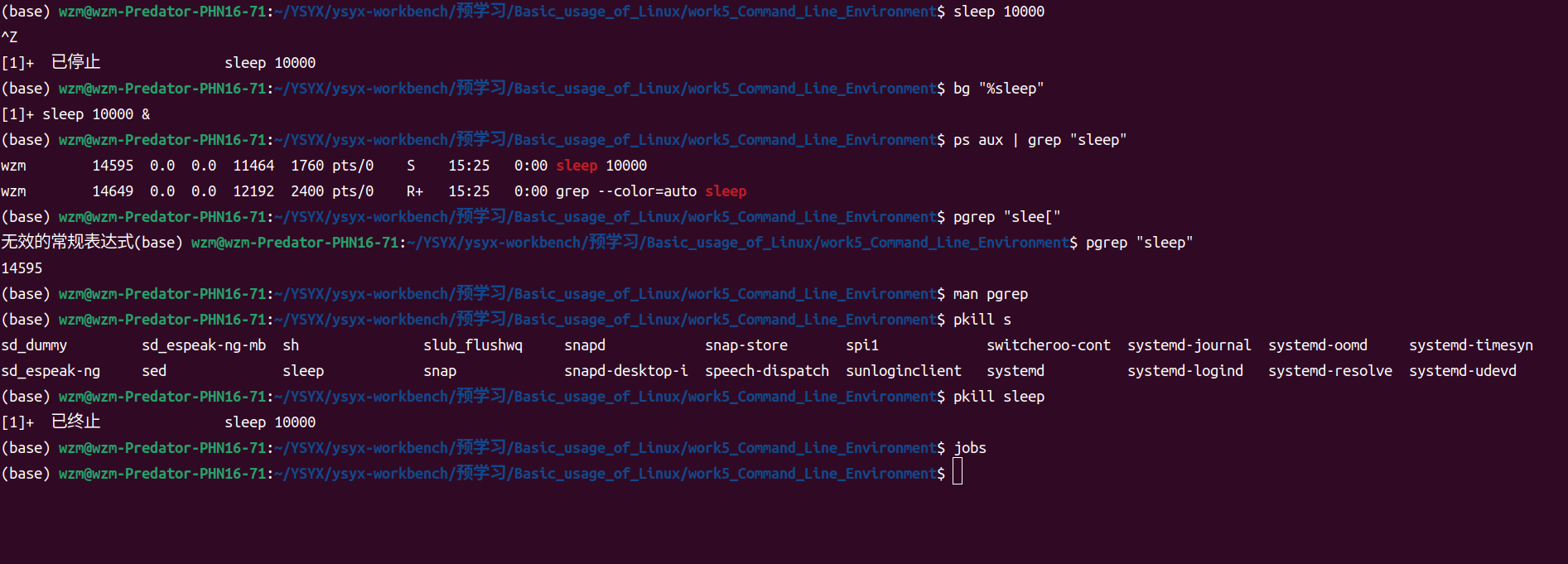
#### homework6

### 命令行环境

#### 任务控制

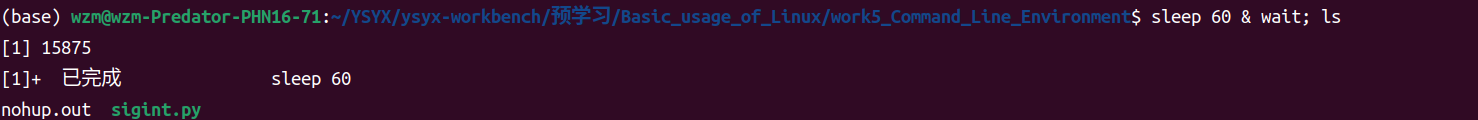
##### homework1

我们可以使用类似 ps aux | grep 这样的命令来获取任务的 pid ，然后您可以基于pid 来结束这些进程。但我们其实有更好的方法来做这件事。在终端中执行 sleep 10000 这个任务。然后用 Ctrl-Z 将其切换到后台并使用 bg来继续允许它。现在，使用 pgrep 来查找 pid 并使用 pkill 结束进程而不需要手动输入pid。(提示：: 使用 -af 标记)。

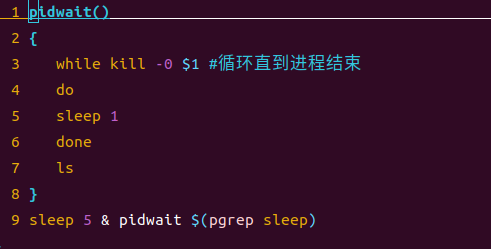


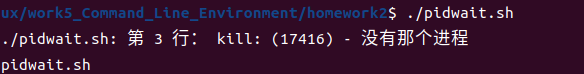
##### homework2

如果您希望某个进程结束后再开始另外一个进程， 应该如何实现呢？在这个练习中，我们使用 sleep 60 & 作为先执行的程序。一种方法是使用 wait 命令。尝试启动这个休眠命令，然后待其结束后再执行 ls 命令。



但是，如果我们在不同的 bash 会话中进行操作，则上述方法就不起作用了。因为 wait 只能对子进程起作用。之前我们没有提过的一个特性是，kill 命令成功退出时其状态码为 0 ，其他状态则是非0。kill -0 则不会发送信号，但是会在进程不存在时返回一个不为0的状态码。请编写一个 bash 函数 pidwait ，它接受一个 pid 作为输入参数，然后一直等待直到该进程结束。您需要使用 sleep 来避免浪费 CPU 性能。





#### 终端多路复用

##### homework1

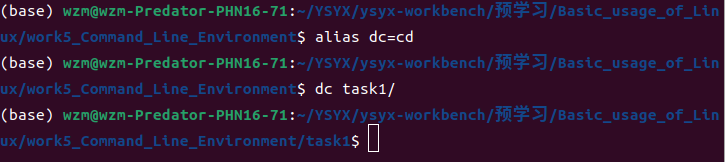
请完成这个 tmux 教程 参考这些步骤来学习如何自定义 tmux。

（已完成）

#### 别名

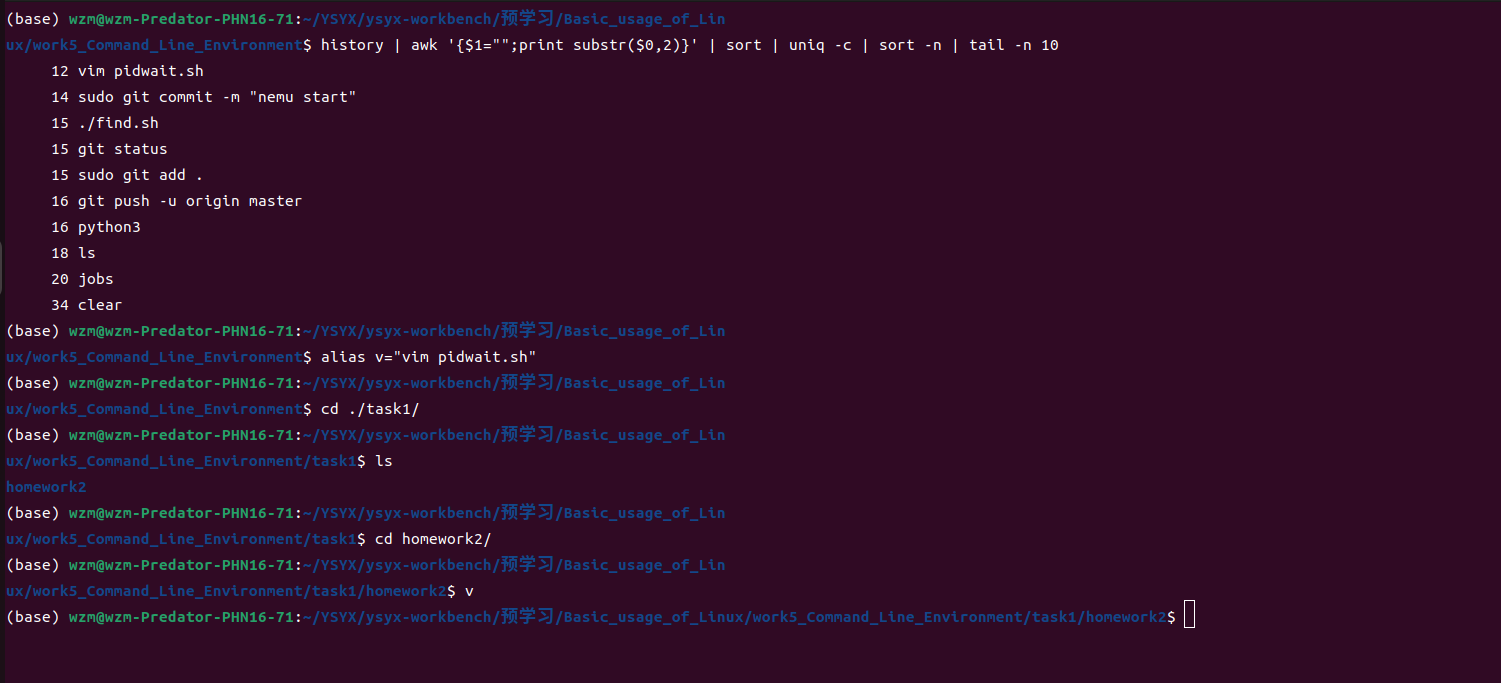
##### homework1

创建一个 dc 别名，它的功能是当我们错误的将 cd 输入为 dc 时也能正确执行。



##### homework2

执行 history | awk '{$1="";print substr($0,2)}' | sort | uniq -c | sort -n | tail -n 10 来获取您最常用的十条命令，尝试为它们创建别名。注意：这个命令只在 Bash 中生效，如果您使用 ZSH，使用history 1 替换 history。



#### 配置文件

（已经完成好配置）

#### 远端设备

（未完成，由于是linux操作系统）

### 版本控制（git）

#### homework1

如果您之前从来没有用过 Git，推荐您阅读 Pro Git 的前几章，或者完成像 Learn Git Branching这样的教程。重点关注 Git 命令和数据模型相关内容；

（完成）

#### homework2

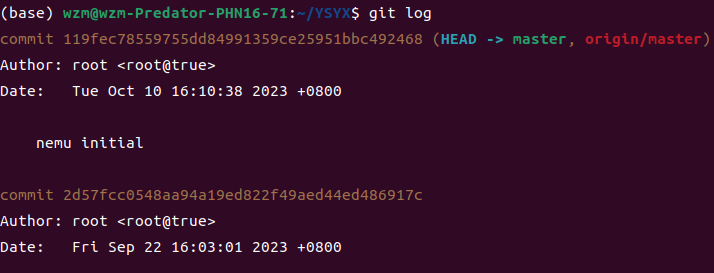
Fork 本课程网站的仓库

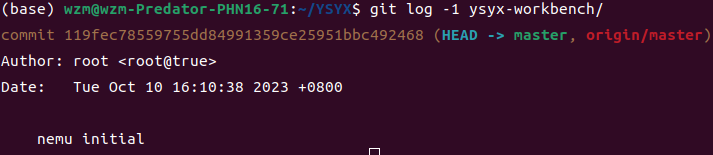
将版本历史可视化并进行探索

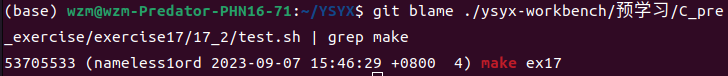
是谁最后修改了 README.md文件？（提示：使用 git log 命令并添加合适的参数）

最后一次修改\_config.yml 文件中 collections: 行时的提交信息是什么？（提示：使用 git blame 和 git show）

（注：为方便起见，我对一生一芯项目进行操作）







#### homework3

使用 Git 时的一个常见错误是提交本不应该由 Git 管理的大文件，或是将含有敏感信息的文件提交给 Git 。尝试向仓库中添加一个文件并添加提交信息，然后将其从历史中删除 ( 这篇文章也许会有帮助)；





#### homework4

从 GitHub 上克隆某个仓库，修改一些文件。当您使用 git stash 会发生什么？当您执行 git log --all --oneline 时会显示什么？通过 git stash pop 命令来撤销 git stash 操作，什么时候会用到这一技巧？

使用 git stash 命令时：

所有未提交的更改（无论是在工作目录中还是在暂存区中）都会被保存起来，得到一个干净的工作状态。

这提供了一个机会去切换分支或执行其他任务，而不必提交尚未完成的工作。

执行 git log --all --oneline 命令时：

显示所有分支的提交日志，每个提交都在一行中。

使用 git stash pop 命令来撤销 git stash 操作：

恢复之前保存的更改，并从stash列表中删除它。如果在 git stash 之后你进行了其他更改，执行 git stash pop 时可能会遇到合并冲突。

什么时候会用到这一技巧:

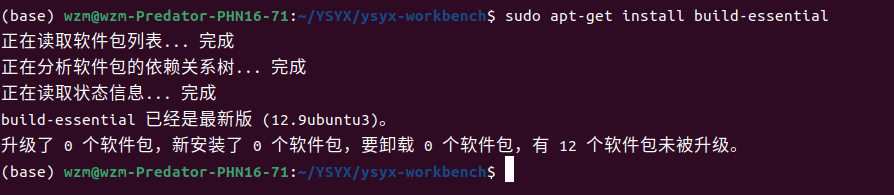
正在进行某些更改，但突然需要切换到其他分支处理某些任务时。使用 git stash 可以临时保存当前更改，然后在返回时使用 git stash pop 恢复这些更改。

想快速保存当前的更改，但不想创建一个新的提交时。

需要从远程获取最新的更改，但不想提交本地的工作时。使用 git stash 保存本地更改，执行 git pull 获取远程更改，然后使用 git stash pop 恢复本地更改。

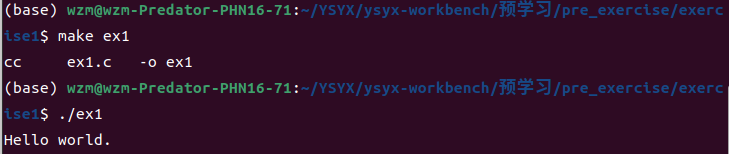
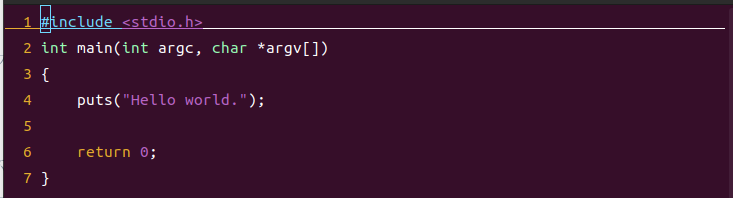
## C语言入门必做题

### 练习0

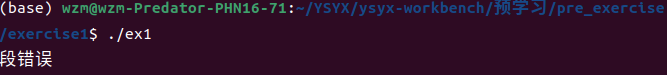


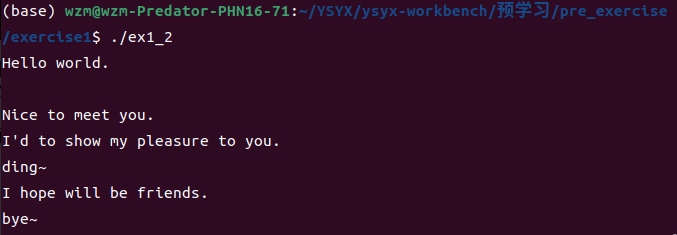
### 练习1

原始编译执行



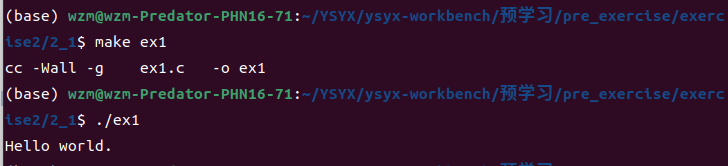
修改exl（删去两行）后执行

编写ex1\_2.c

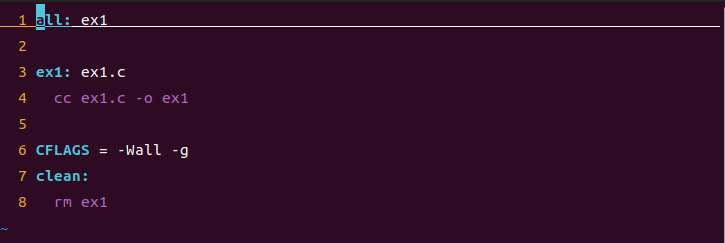


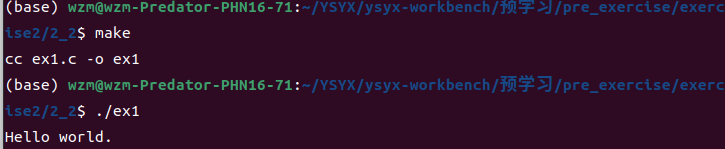
### 练习2

基础练习（文件夹2\_1）



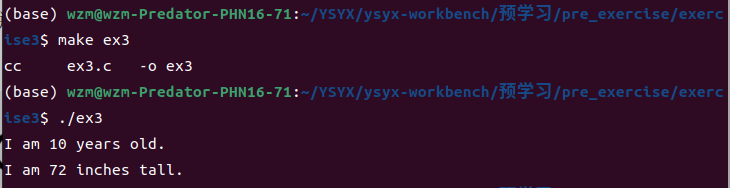
附加题1（文件夹2\_2）





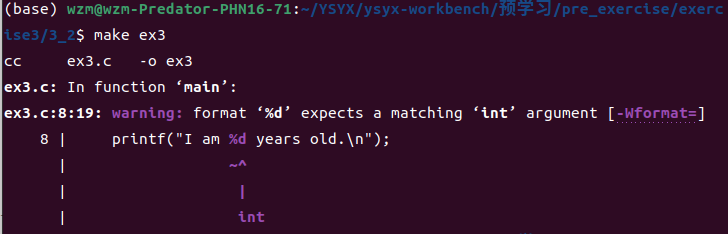
### 练习3

基础练习（文件夹3\_1）

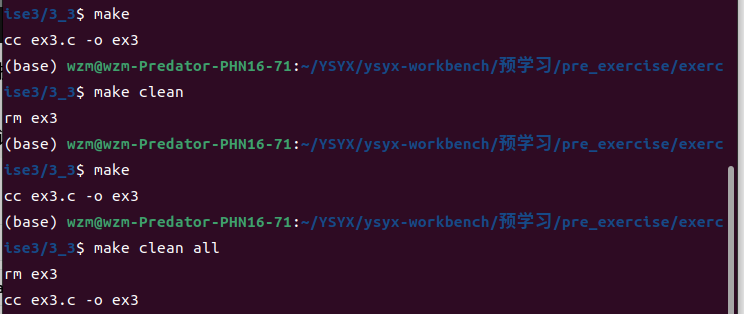


崩溃练习（文件夹3\_2）

去除age后

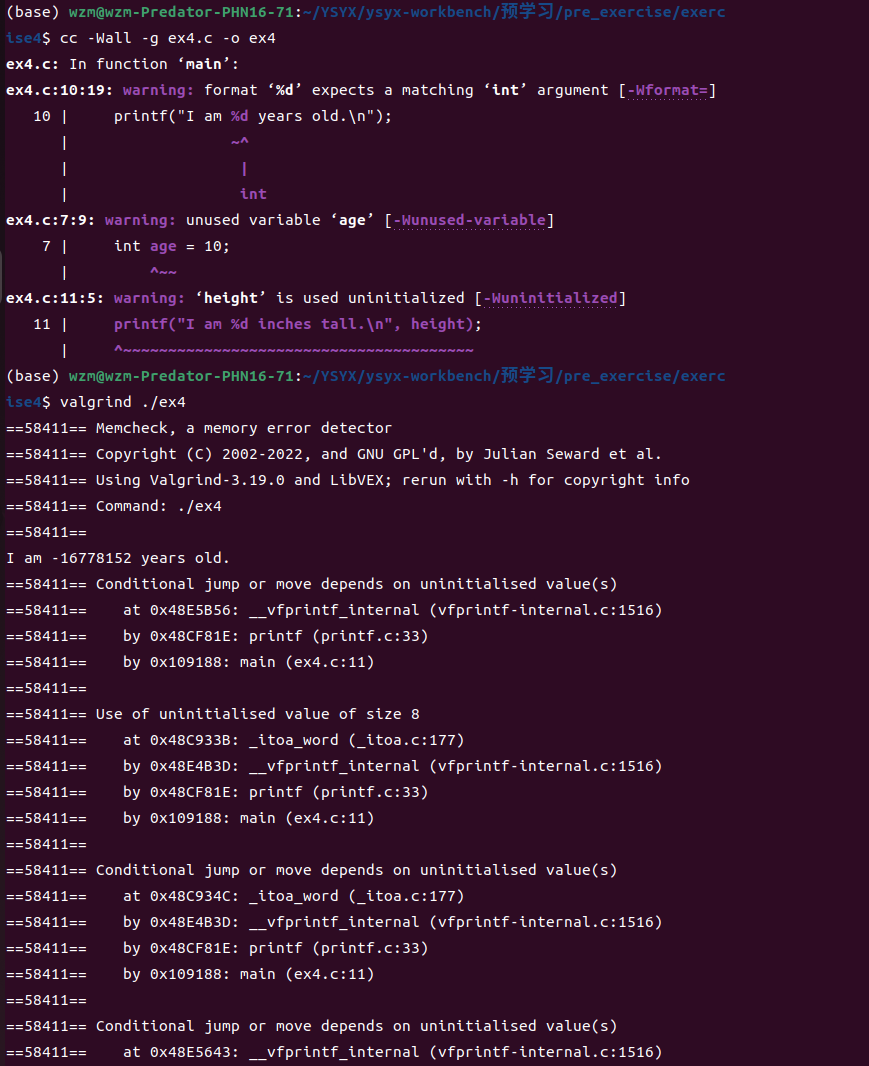


make clean / make clean all（文件夹3\_3）

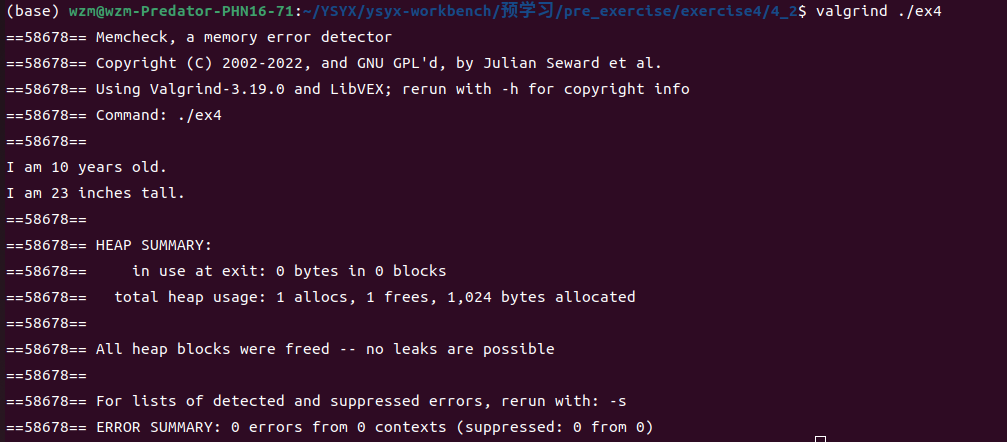


### 练习4

基础练习（文件夹4\_1）

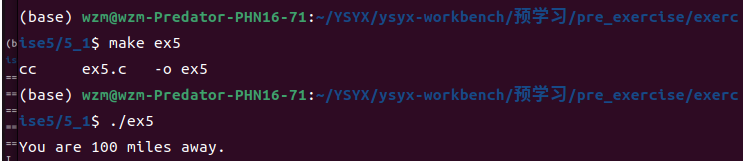


修复后（文件夹4\_2）



### 练习5

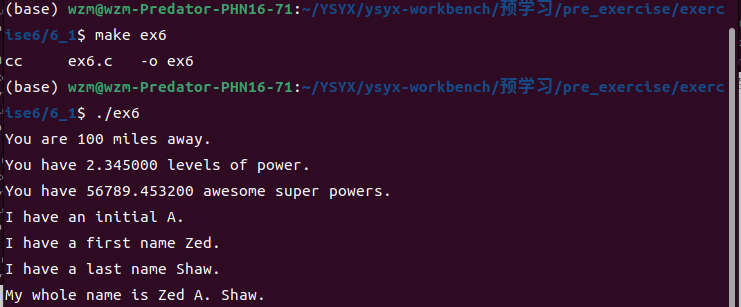
基础练习（文件夹5\_1）



这一节没有特别的任务，略。

### 练习6

基础练习（文件夹6\_1）



调试崩溃内容，略

**printf格式化**

下面是一些常用的格式化选项和转换说明符：

- `%d`：输出十进制整数。

- `%f`：输出浮点数。

- `%s`：输出字符串。

- `%c`：输出单个字符。

- `%x` 或 `%X`：输出十六进制整数（小写或大写）。

- `%o`：输出八进制整数。

- `%e` 或 `%E`：输出科学计数法表示的浮点数（小写或大写）。

- `%u`：输出无符号整数。

- `%p`：输出指针的地址。

除了上述的基本转换说明符外，还有一些修饰符可以用于进一步控制输出的格式：

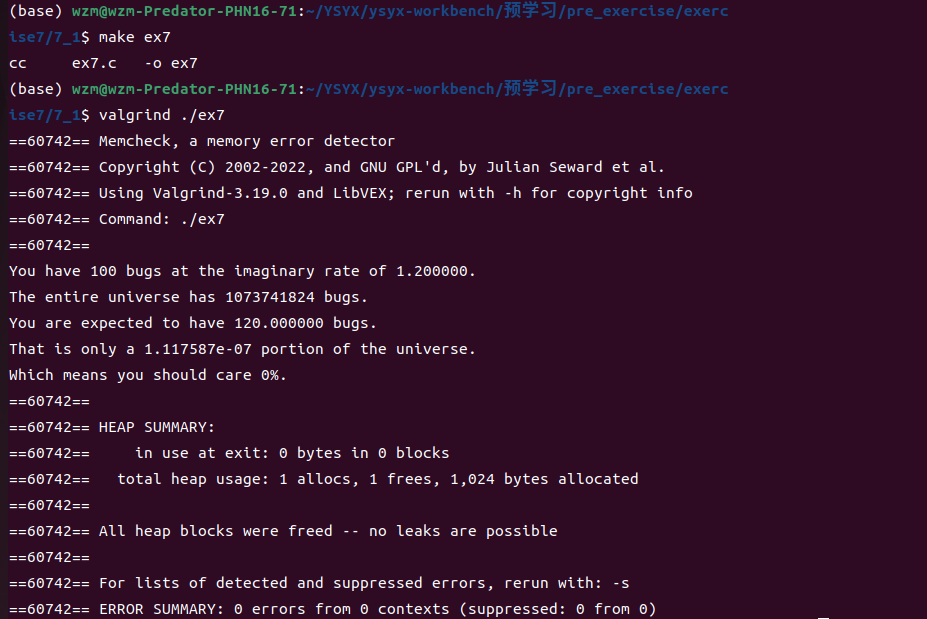
- 宽度字段：指定输出字段的最小宽度，可以使用数字来表示。例如，`%5d` 表示输出一个最小宽度为 5 的十进制整数。

- 精度字段：对于浮点数，可以使用精度字段来控制小数点后的位数。例如，`%0.2f` 表示输出一个保留两位小数的浮点数。

- 标志字段：可以使用一些标志来修改输出的行为，例如使用 `+` 标志显示正负号，使用 `0` 标志在宽度不足时填充零等。

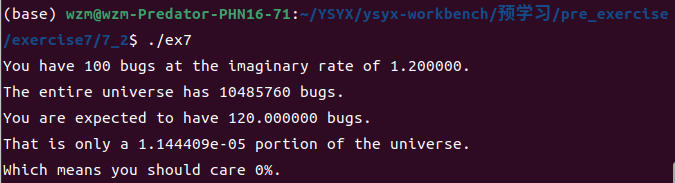
### 练习7

基础练习（文件夹7\_1）

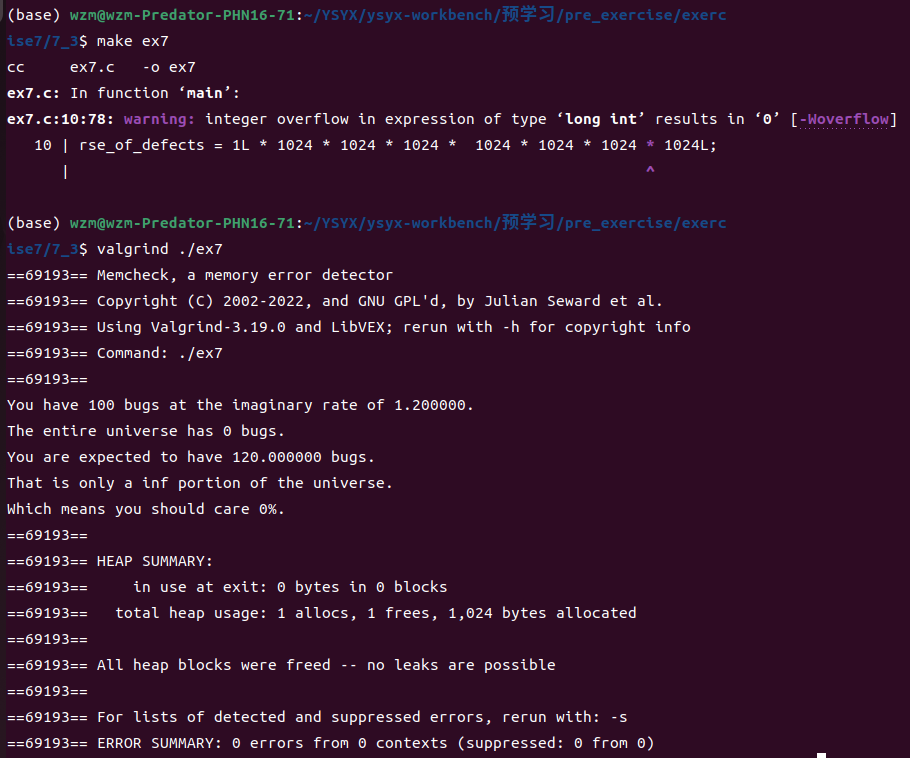


修改后（文件夹7\_2）





修改后——long溢出，得到零值（文件夹7\_3）



**改为unsigned long后发生什么**

改为unsigned long后，不溢出，略。

**unsigned做了什么**

当使用 unsigned 修饰符声明一个变量时，它会改变该变量的取值范围和解释方式。具体来说，unsigned 所做的事情包括：

扩展取值范围：使用 unsigned 修饰符可以扩展整数类型的取值范围。默认情况下，带符号整数类型的范围是对称的，一半用于负数，一半用于非负数。而使用 unsigned 修饰符后，整个范围都可以用于非负数。

消除符号位：带符号整数类型使用最高位作为符号位，用于表示正负。而无符号整数类型不具有符号位，所以所有位都用于表示数值。

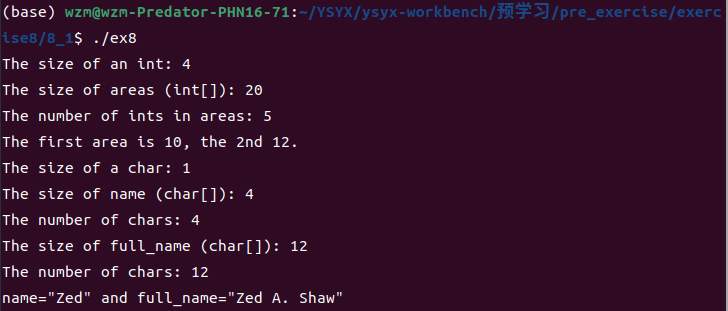
修改算术运算规则：在无符号整数类型之间进行算术运算时，通常采用模运算的方式。例如，无符号整数的加法、减法和乘法都会执行模运算，超出范围的值将被截断。

**char与int相乘**

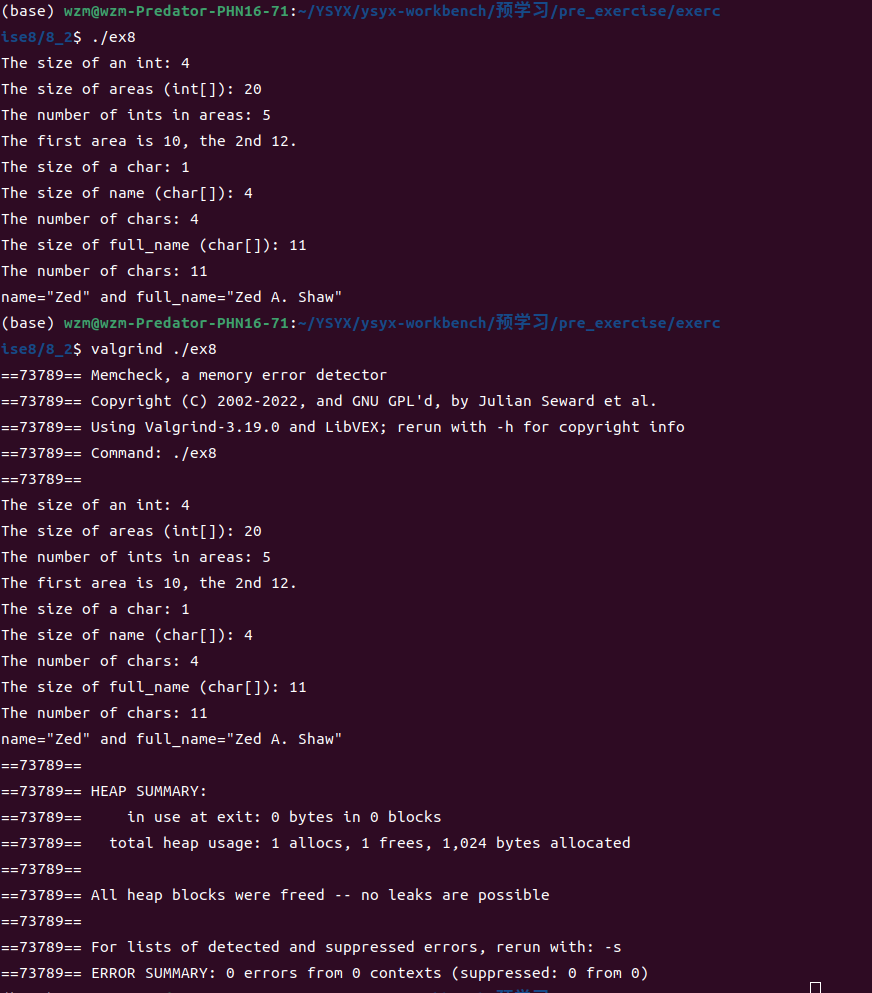
char和int相乘是char根据编码转化为对应的数，提升为int型，从而与int相乘，得到一个int型的数。

### 练习8

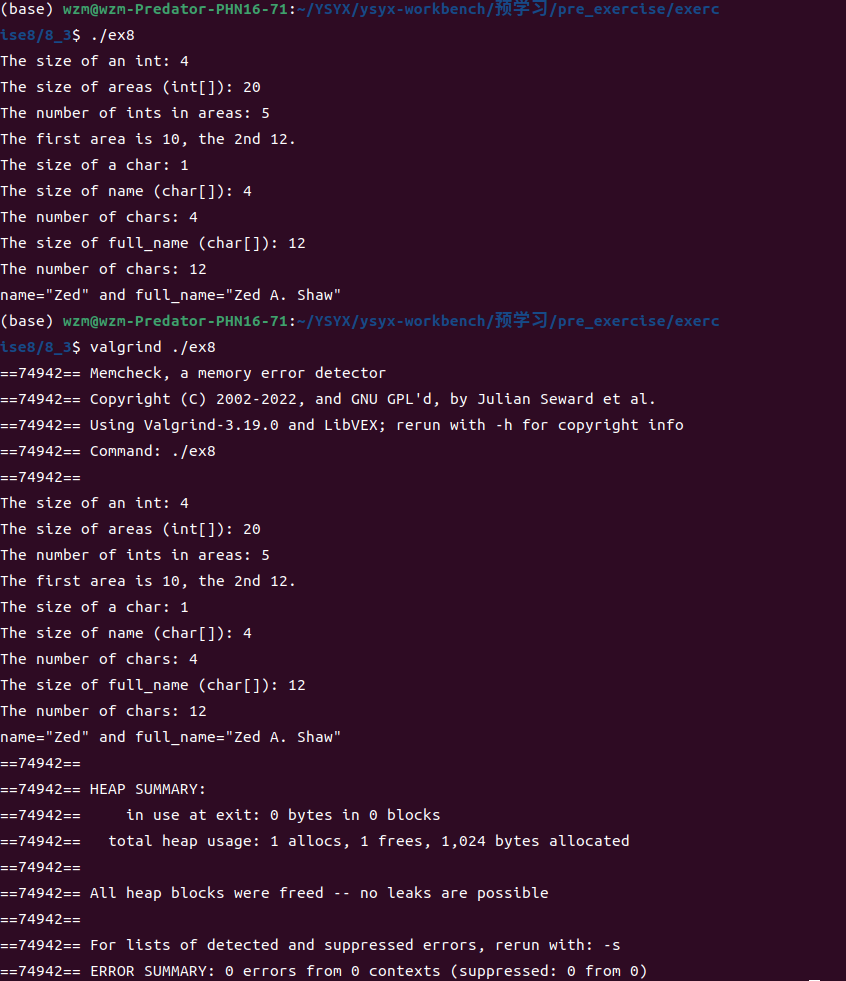
基础练习（文件夹8\_1）



删去’\0’未报错（文件夹8\_2）

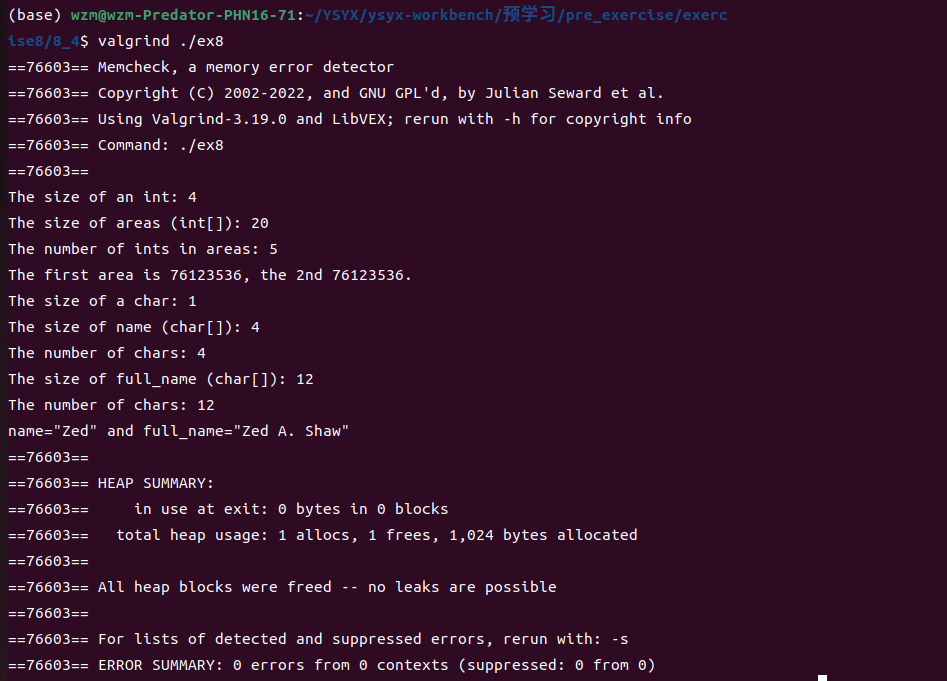


full\_name 转移到上面未报错（文件夹8\_3）



修改areas[0]为areas[10]（文件夹8\_4）

可以看到，在溢出的寻址上，得到了意料之外的值

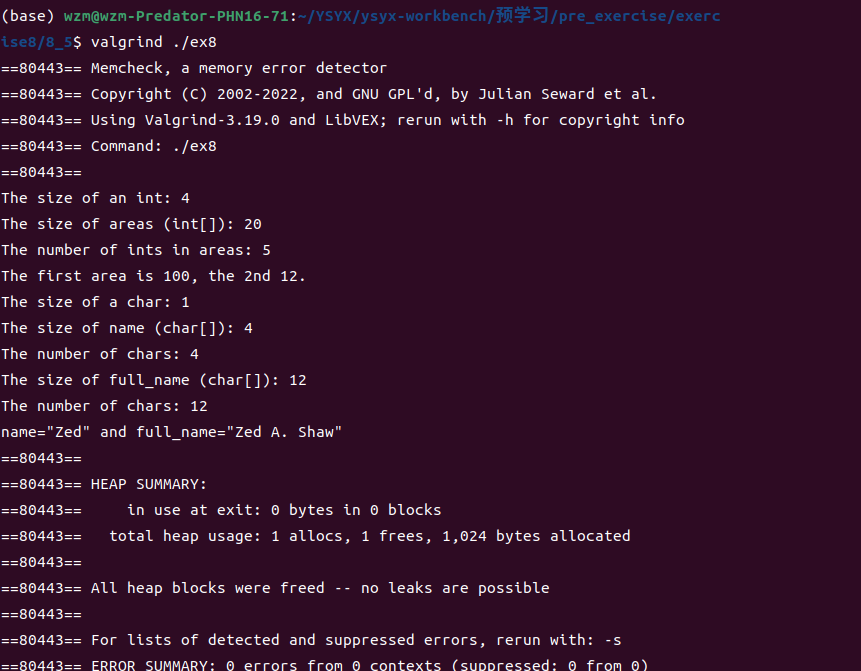


name与full\_name变式略

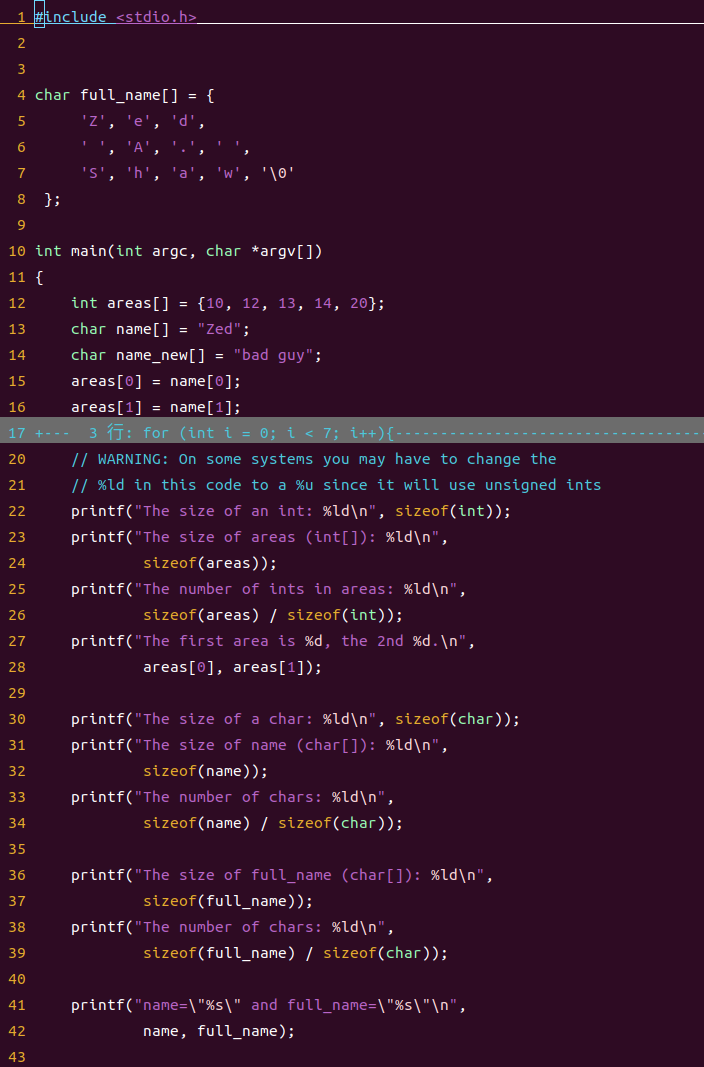
**附加题**

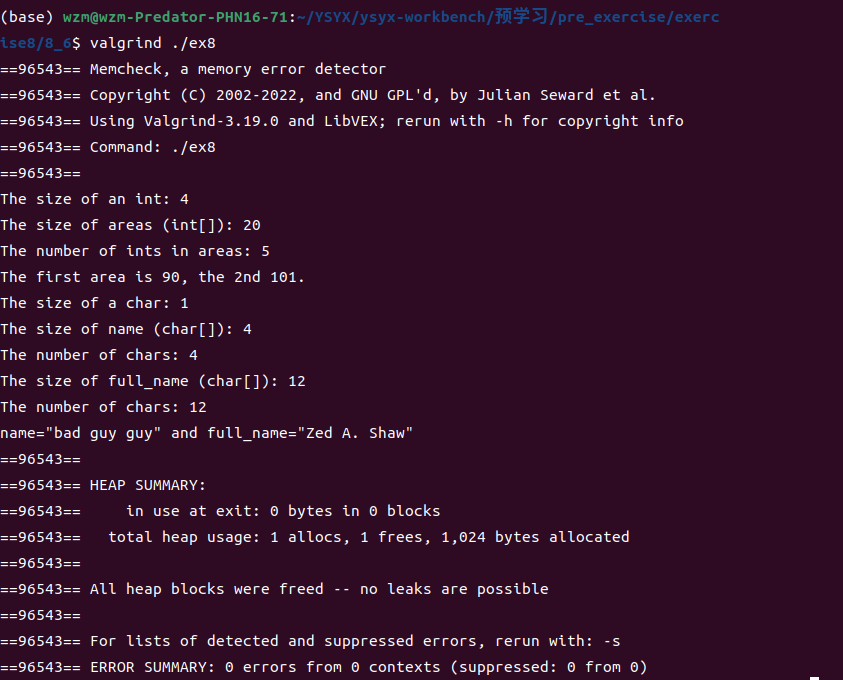
尝试使用areas[0] = 100;以及相似的操作对areas的元素赋值（文件夹8\_5）

修改后如下图所示，修改完成。



对name以不同方式赋值结果（文件夹8\_6）





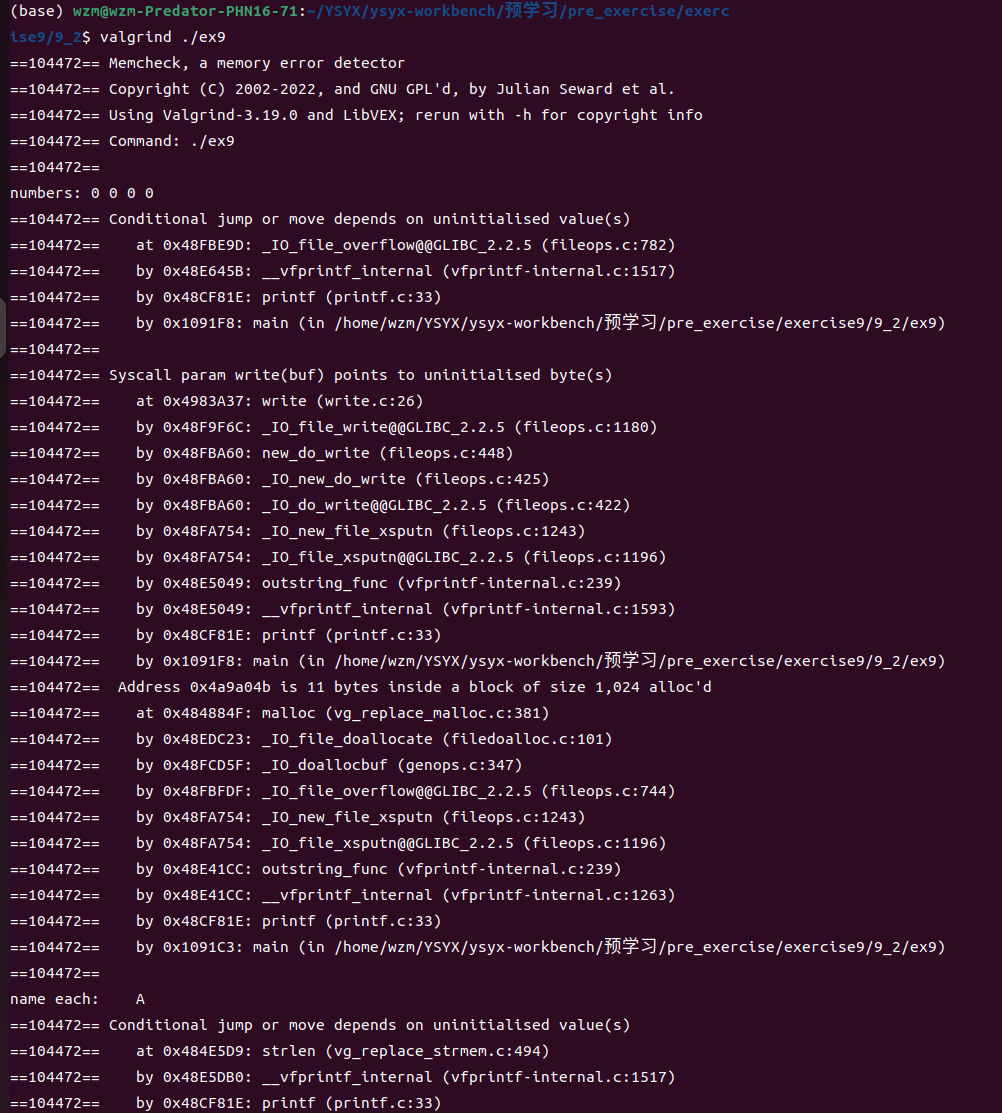
### 练习9

基础练习（文件夹9\_1）



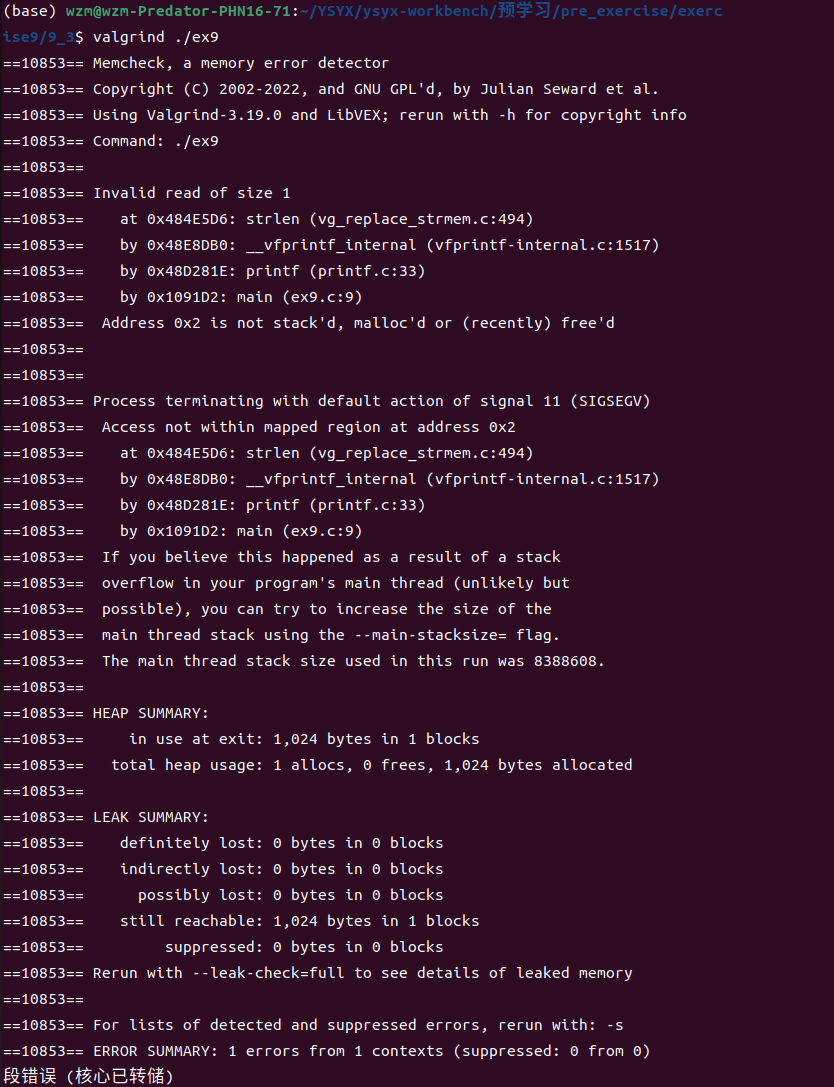
崩溃练习（文件夹9\_2）

修改name未初始化，name[3]赋值为’A’



**附加题**

将一些字符赋给numbers的元素，之后用printf一次打印一个字符，对names执行上述的相反操作，把names当成int数组，并一次打印一个int（文件夹9\_3）



如果一个字符数组占四个字节，一个整数也占4个字节，你可以像整数一样使用整个name吗？你如何用黑魔法实现它？

将字符串数组存储后，建立一个int的指针进行读取输出

将name转换成another的形式，看看代码是否能正常工作

略

### 练习10

基础练习（文件夹10\_1）

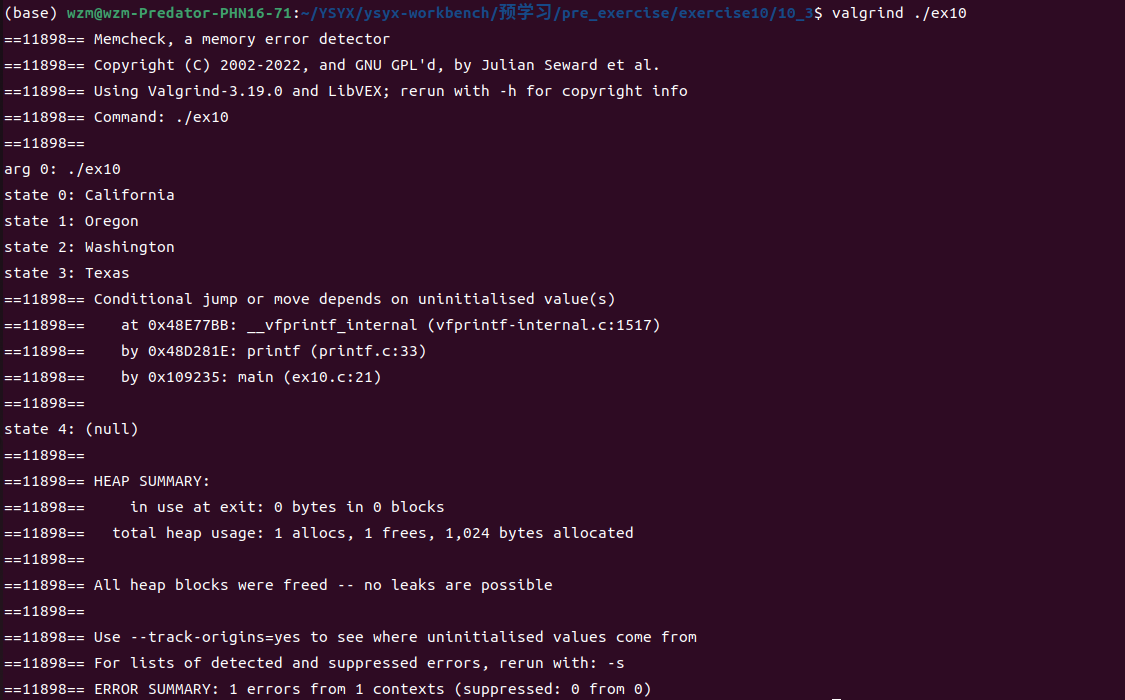


崩溃练习1：将i初始化为0（文件夹10\_2）



可以看到，程序正常运行，输入大概是被存储到arg[0]的位置

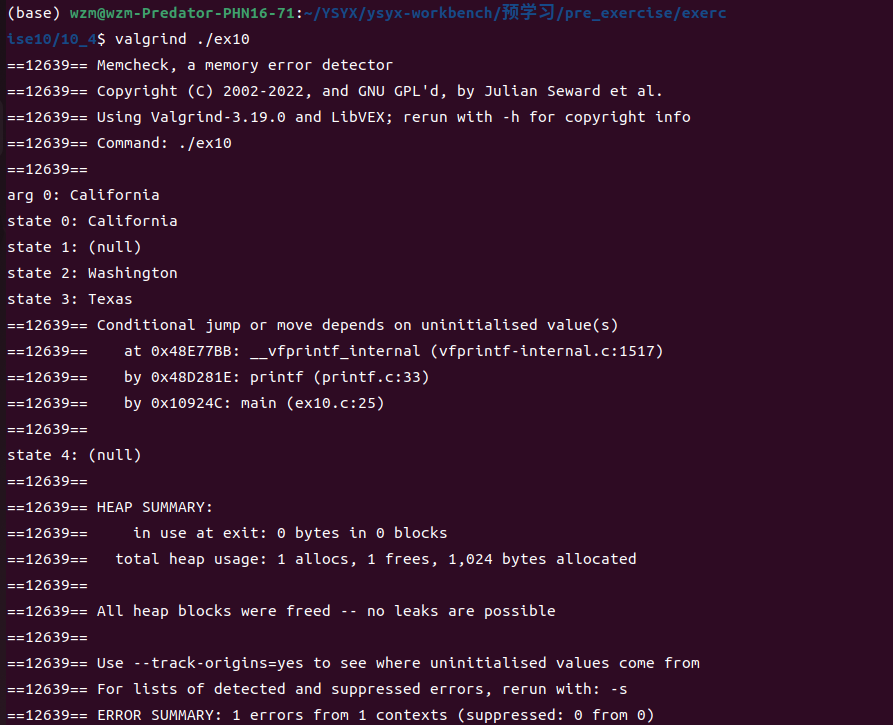
崩溃练习2：将num\_states改为错误的值使它变大（文件夹10\_3）



可以看到，标记未初始化。

**附加题：**

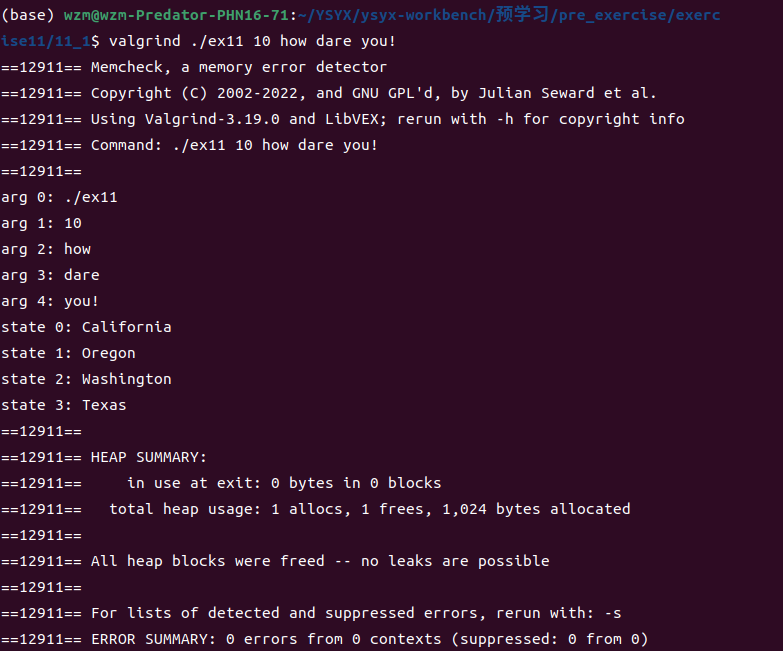
看看你是否能在打印之前将states的一个元素赋值给argv中的元素，再试试相反的操作（文件夹10\_4）



可以看到，报出未初始化的错误

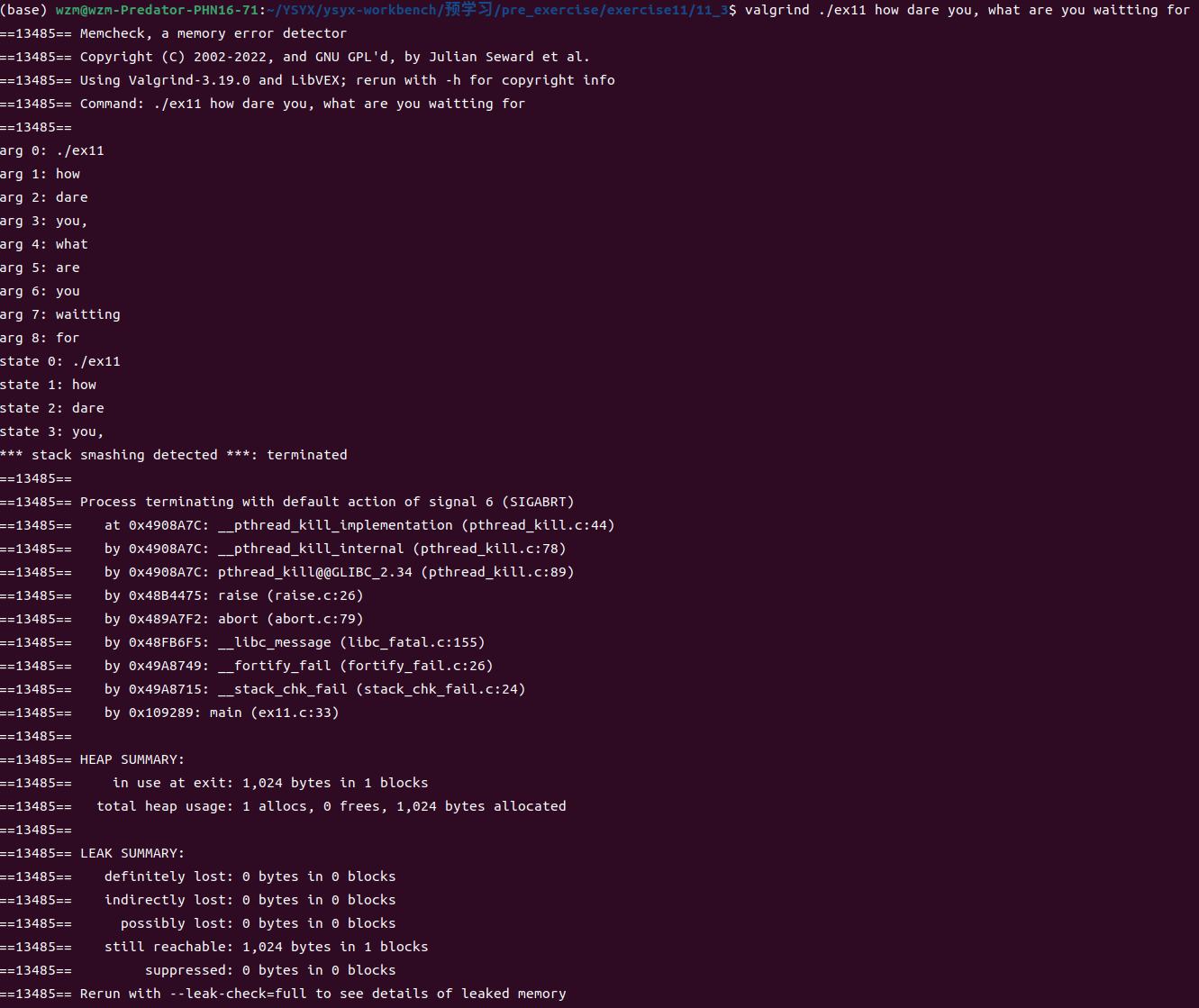
### 练习11

基础练习（文件夹11\_1）



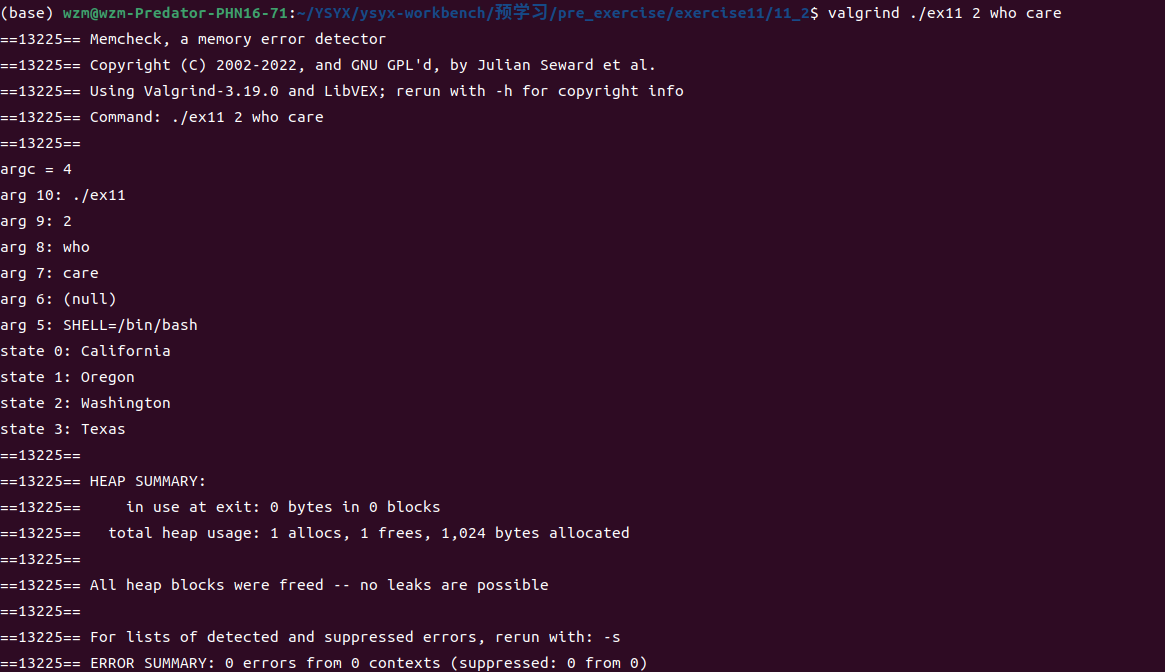
**附加题**

让这些循环倒序执行，通过使用i--从argc开始递减直到0。你可能需要做一些算数操作让数组的下标正常工作（文件夹11\_2）



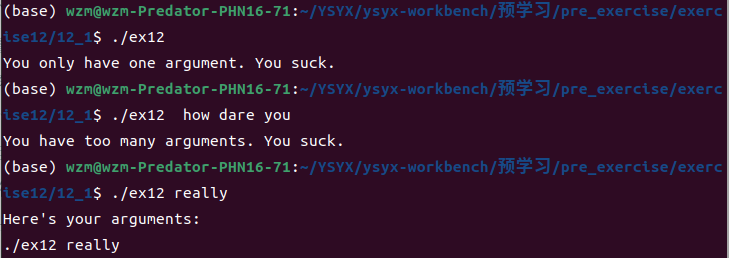
可以看到，正常工作。

使用while循环将argv中的值复制到states，溢出不报错（文件夹11\_3）



### 练习12

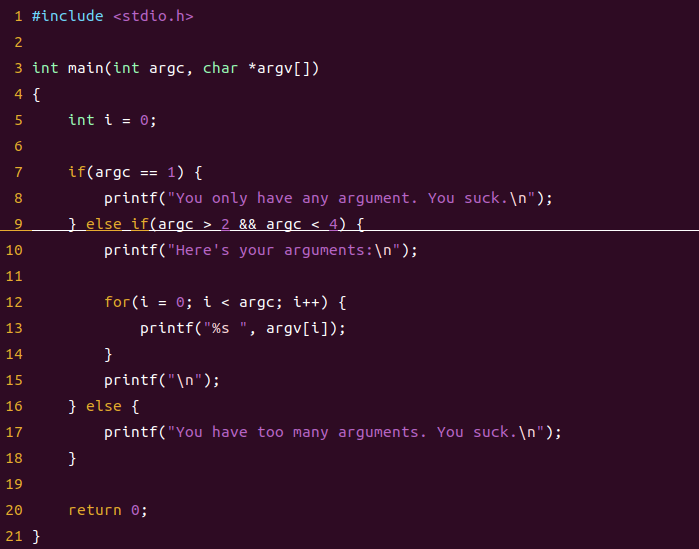
基础练习（文件夹12\_1）



**附加题**

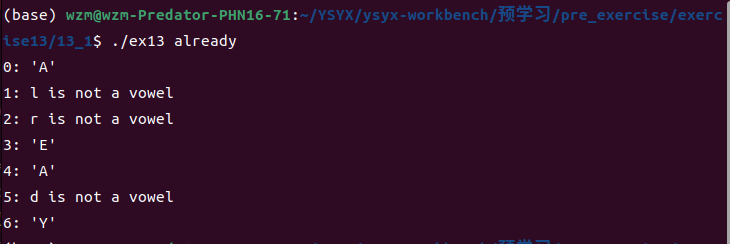
第一个判断所输出的话真的正确吗？由于你的“第一个参数”不是用户输入的第一个参数，把它改正

修改后如下。argv[0]存储的是函数名而非用户输入的第一个参数，现改为如下：



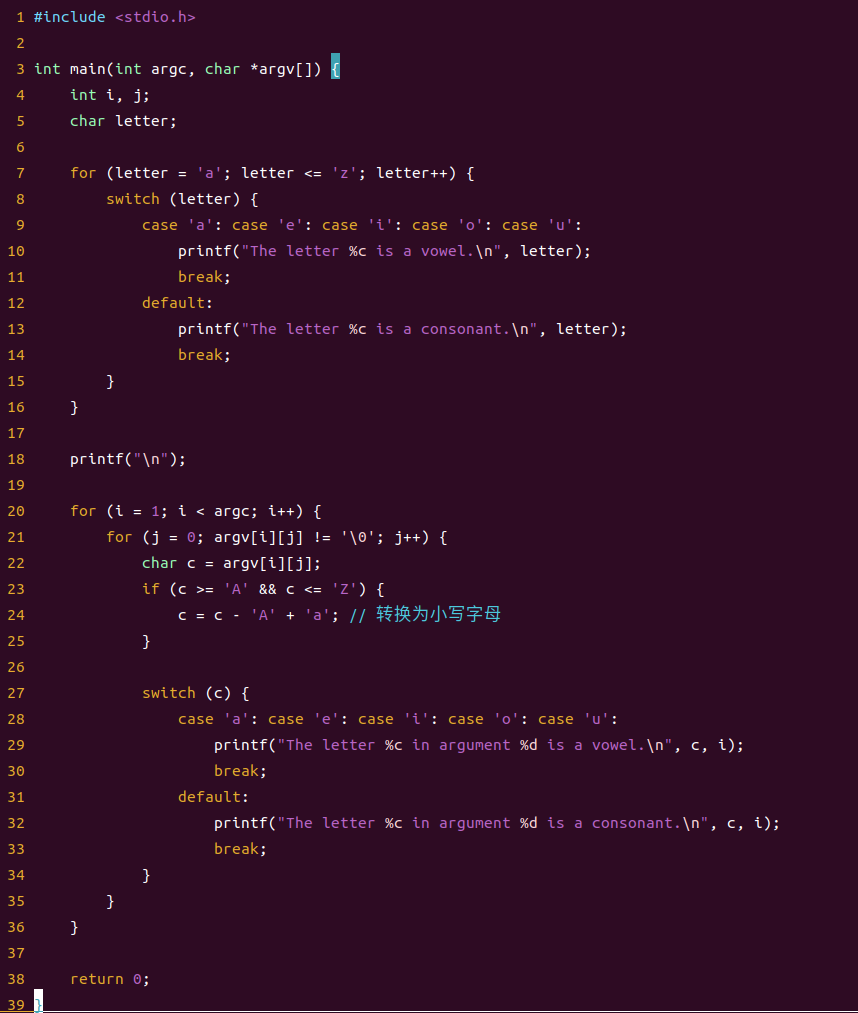
### 练习13

基础练习（文件夹13\_1）



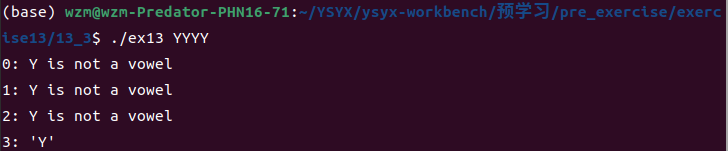
**附加题**

编写另一个程序，在字母上做算术运算将它们转换为小写，并且在switch中移除所有额外的大写字母（文件夹13\_2）



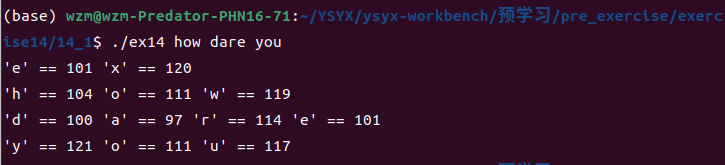
在“Y”的例子中，我在if代码块外面写了个break。这样会产生什么效果？如果把它移进if代码块，会发生什么？（文件夹13\_3）

若移进代码快内部，则y不满足大于2时，break不会发生，会自动执行下一条分支语句而不是跳出循环。



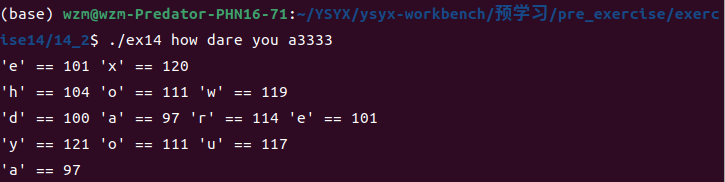
### 练习14

基础练习（文件夹14\_1）

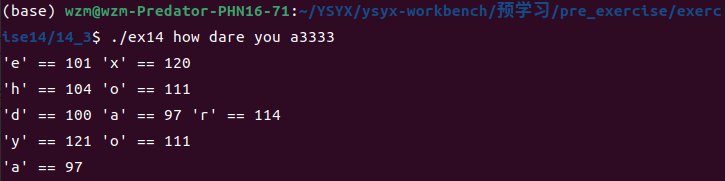


**附加题**

重新编写这些函数，使它们的数量减少。（文件夹14\_2）

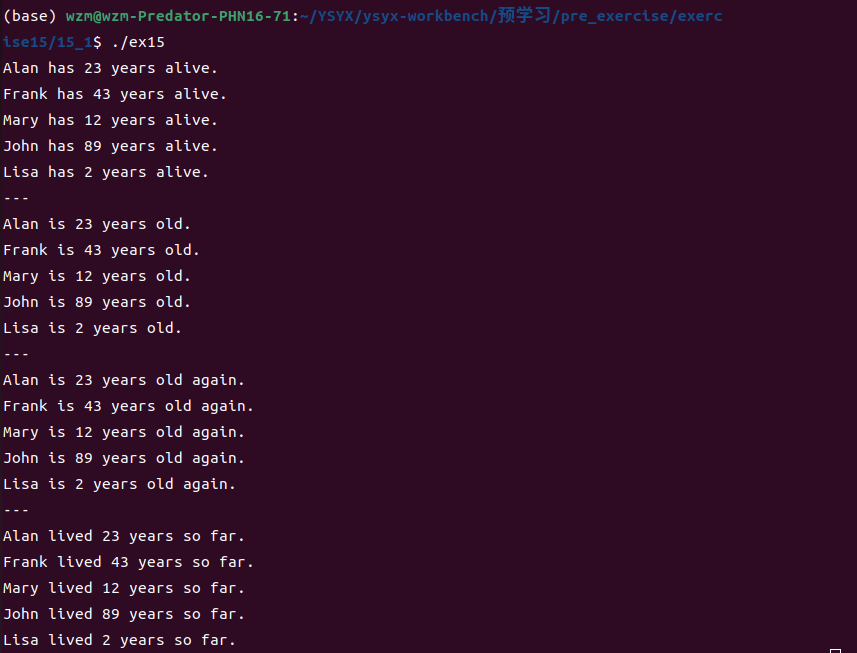


使用strlen函数，让print\_arguments知道每个字符串参数都有多长，之后将长度传入print\_letters。然后重写print\_letters，让它只处理固定的长度，不按照'\0'终止符。你需要#include <string.h>来实现它（文件夹14\_3）



### 练习15

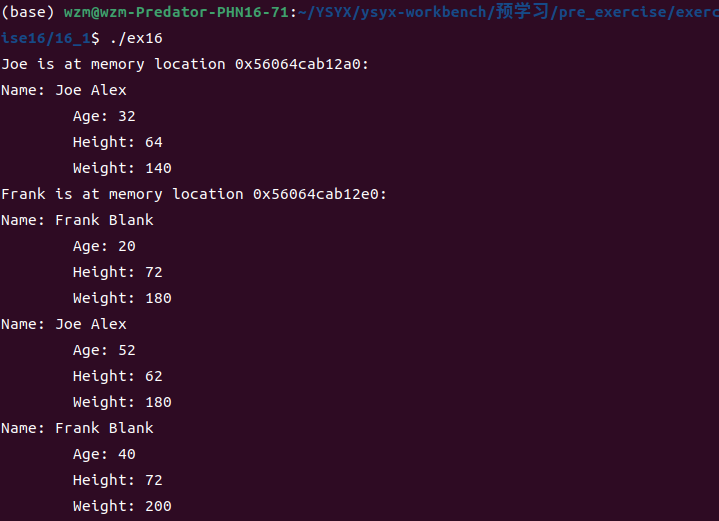
基础练习（文件夹15\_1）



附加题有点没读懂题目i意思，不知道对哪个程序改着写，暂且空着。

### 练习16

基础练习（文件夹16\_1）



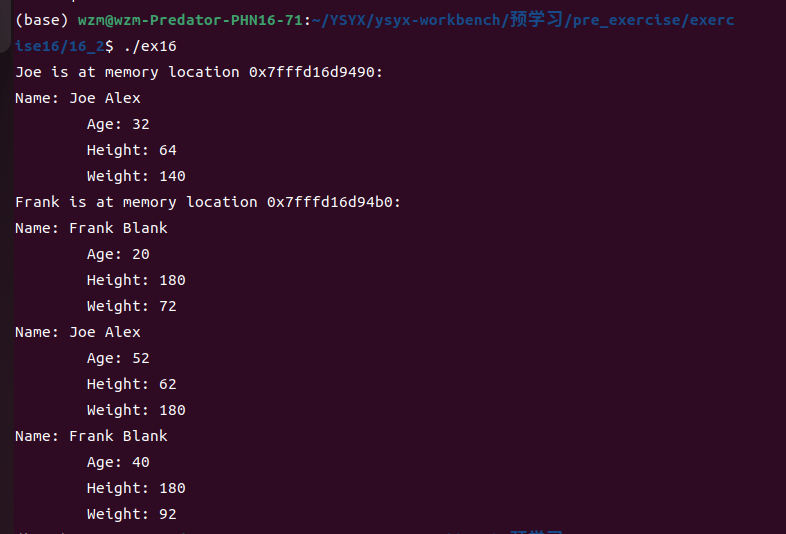
**附加题**

如何在栈上创建结构体，就像你创建任何其它变量那样。

如何使用x.y而不是x->y来初始化结构体。

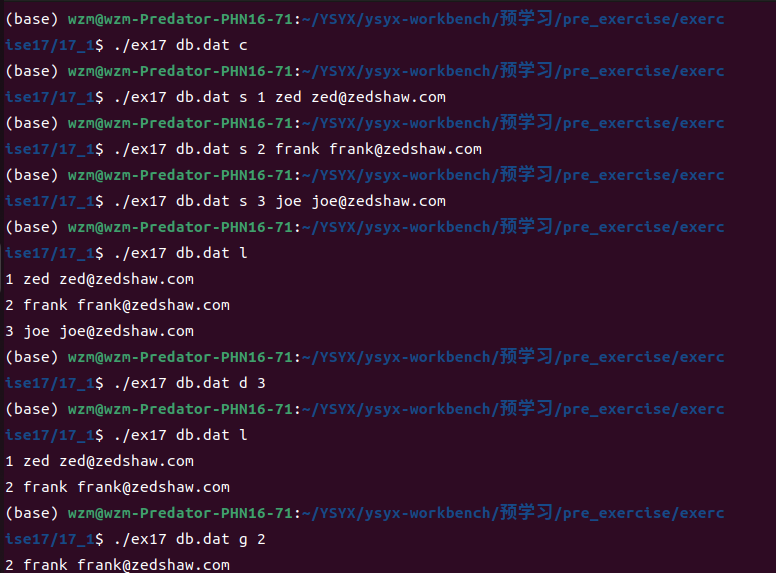
如何不使用指针来将结构体传给其它函数。（文件夹16\_2）

代码见文件夹中ex16.c文件，输出结果如下：



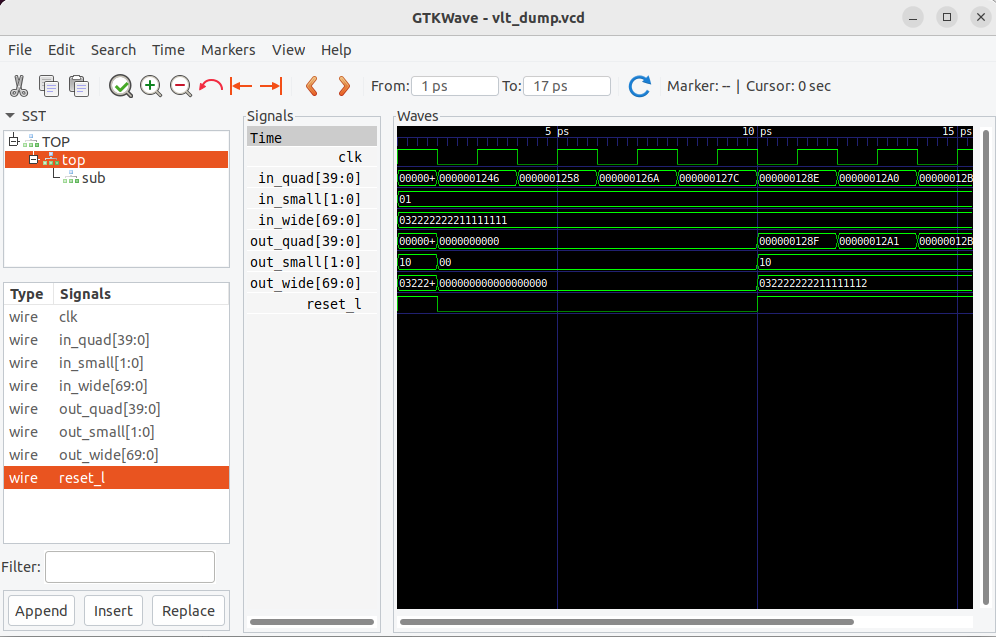
### 练习17

基础练习（文件夹17\_1）

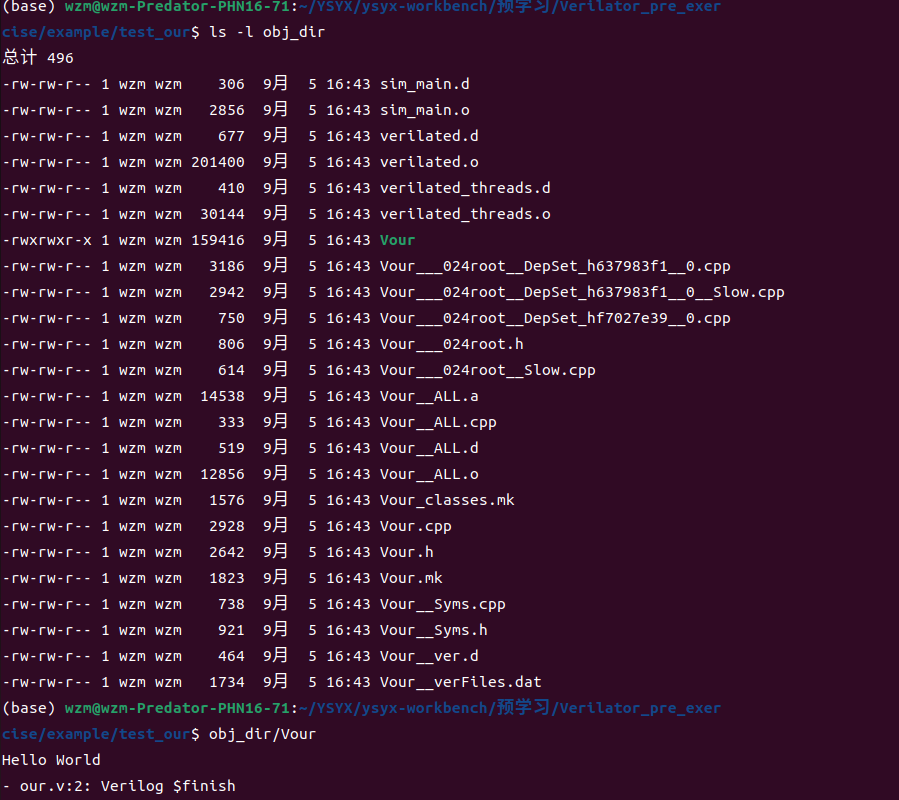


## Verilator环境搭建

源代码中例子



例程运行后例子



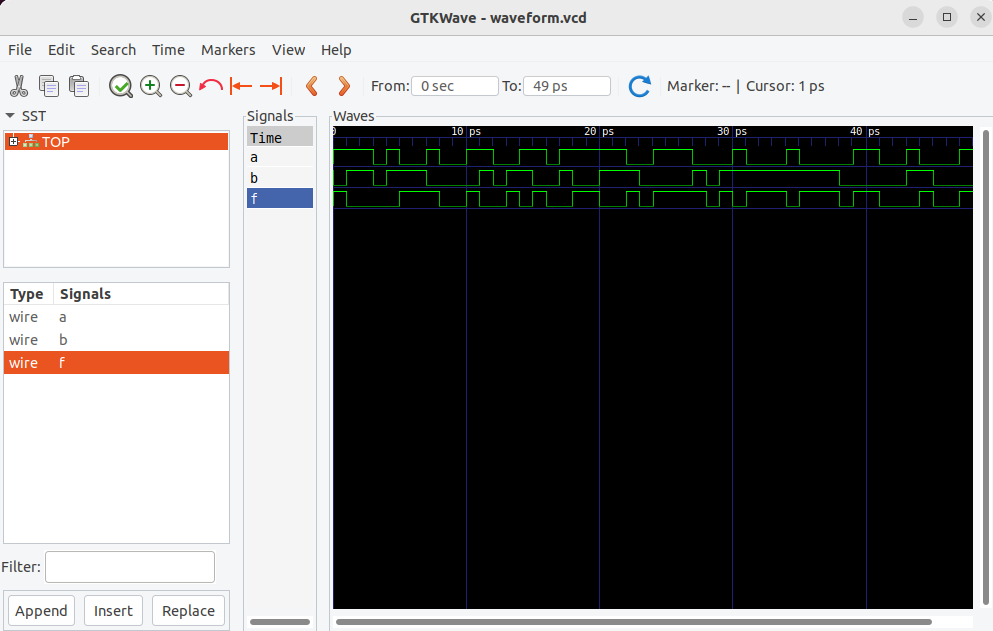
### 双控开关

进入目录

YSYX/ysyx-workbench/预学习/Verilator\_pre\_exercise/on-off switch/npc

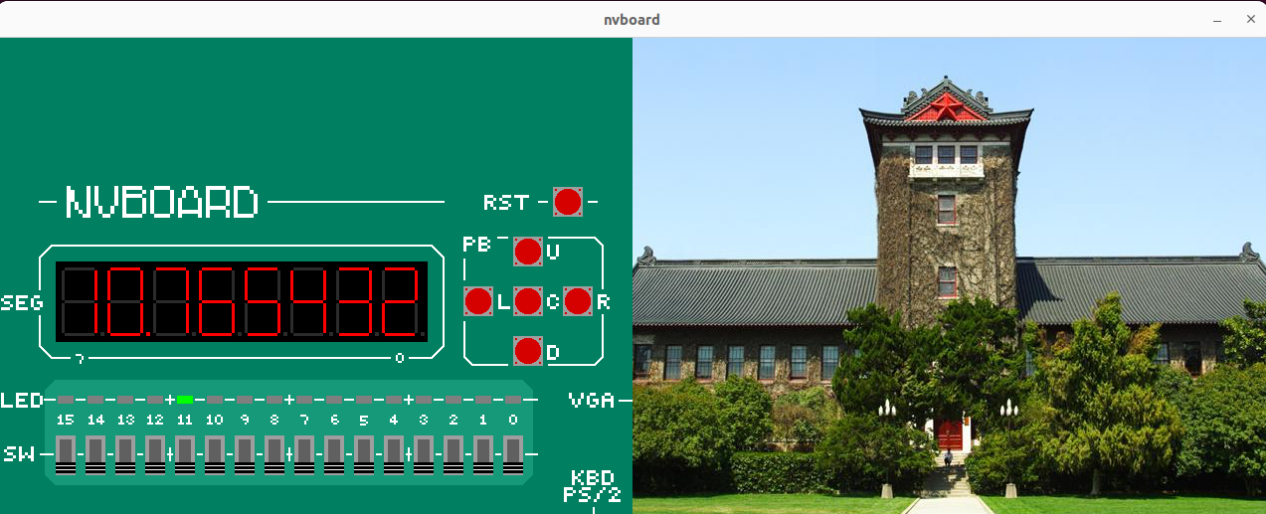
运行start.sh，一键编译、执行、波形图打开

波形图如下所示：

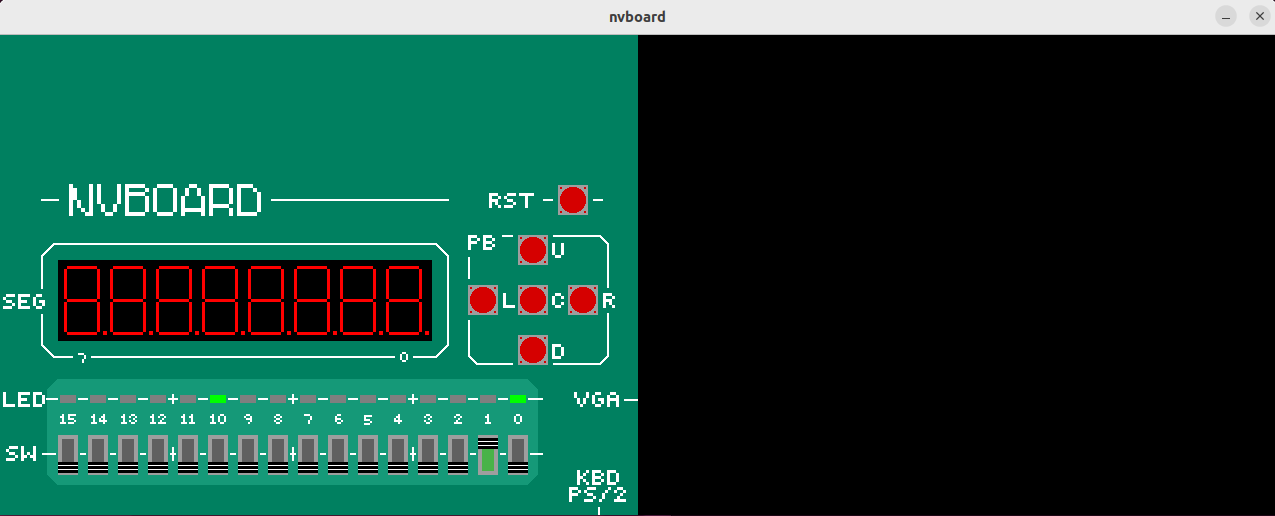


另外，由于本次没有使用makefile（不能打开锁文件），故在YSYX文件夹下完成文件提交。

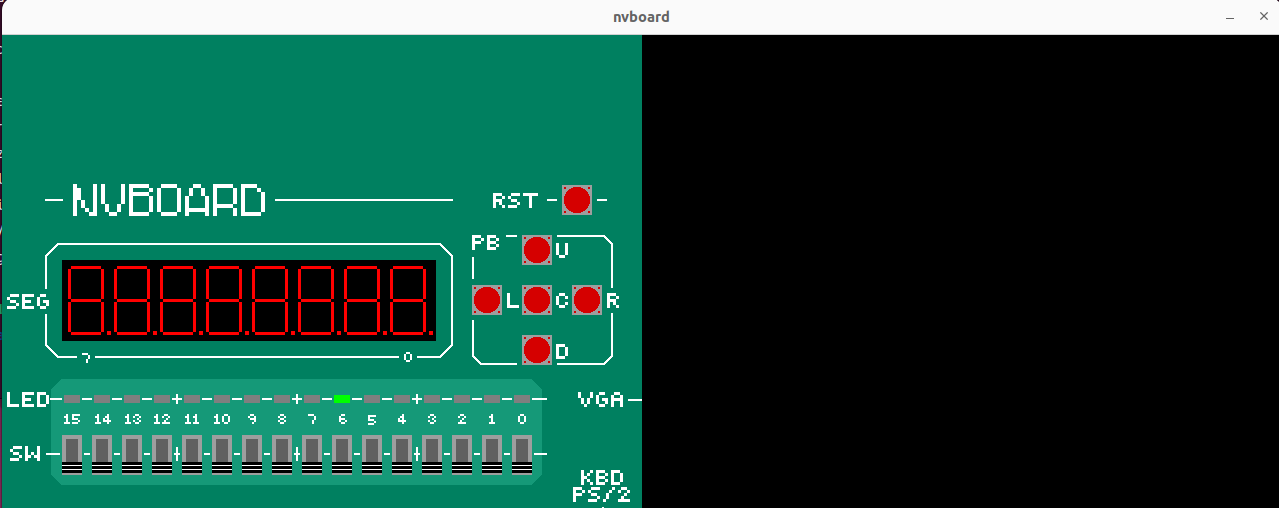
### NVBoard Example

make run之后，得到如下结果：  


### NVBoard 双控开关

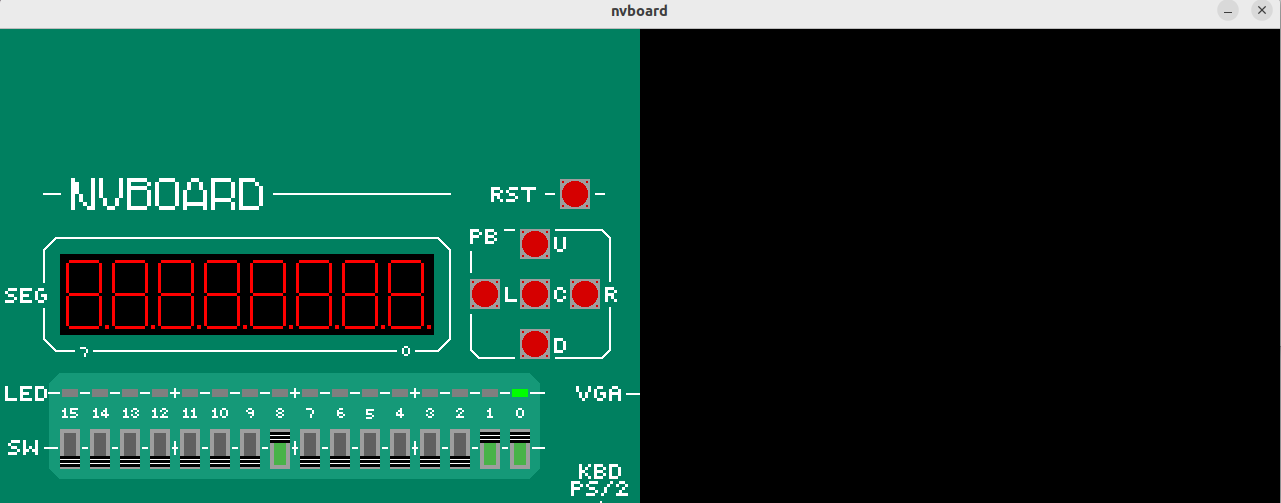


### NVBoard 流水灯

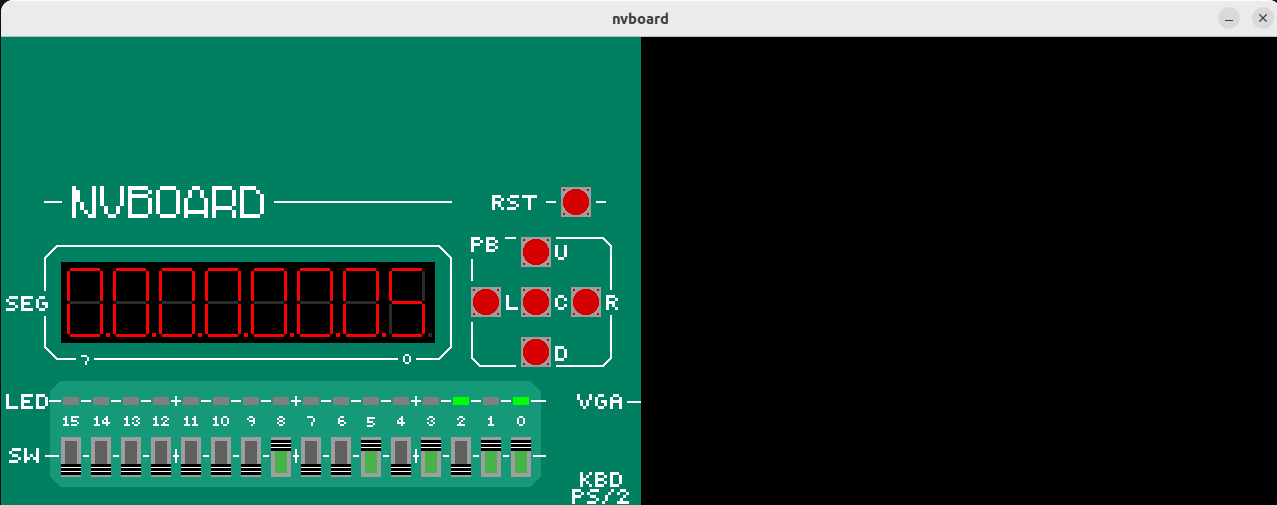


## 数字电路实验

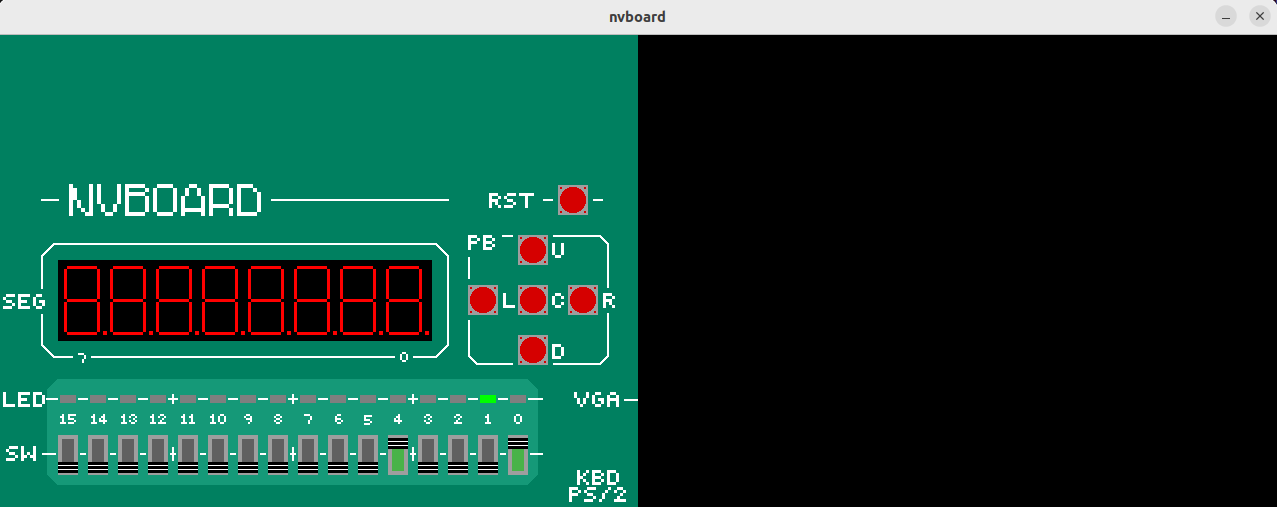
### 实验一



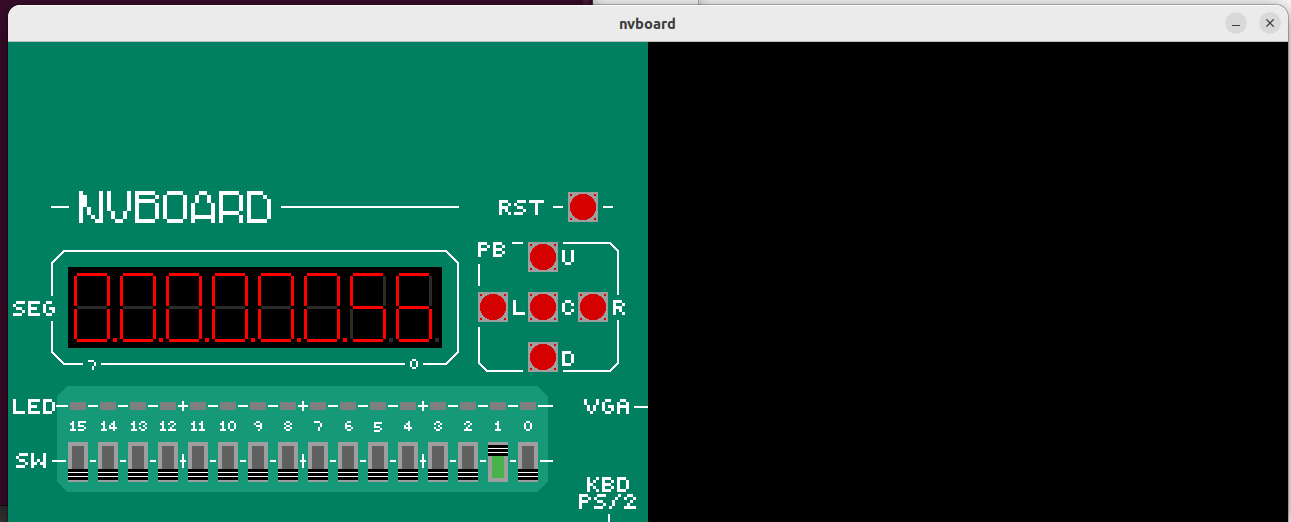
### 实验二



### 实验三



### 实验六



## PA1

第一个客户程序，这里选择riscv32架构