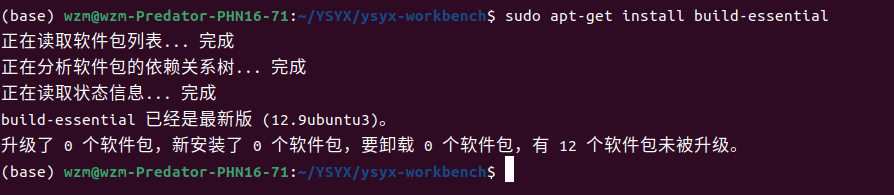
# 预学习准备

## 一生一芯通识问卷

## 

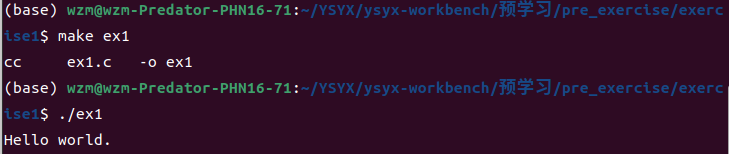
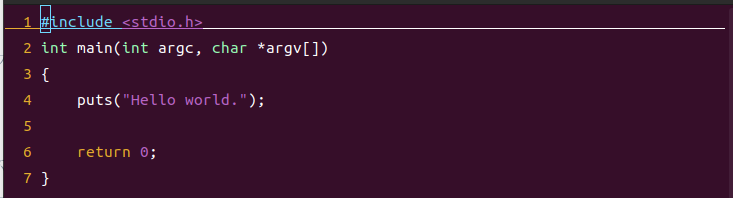
## C语言入门必做题

### 练习0

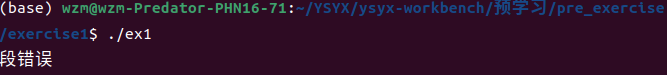


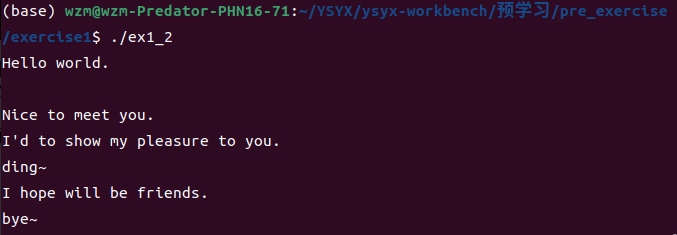
### 练习1

原始编译执行



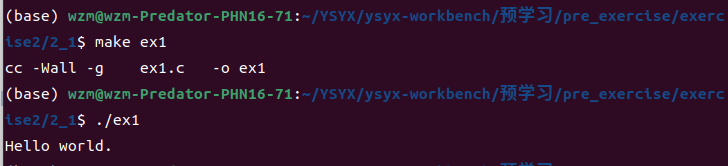
修改exl（删去两行）后执行

编写ex1\_2.c

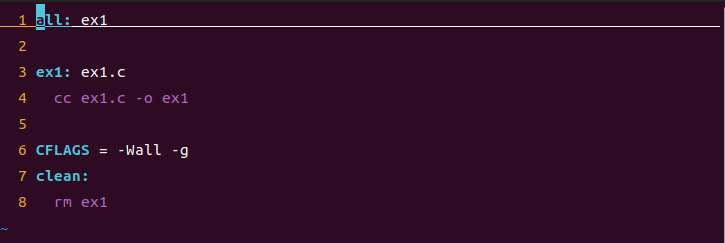


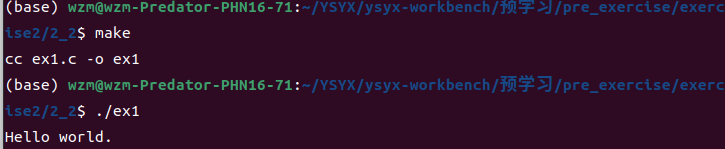
### 练习2

基础练习（文件夹2\_1）



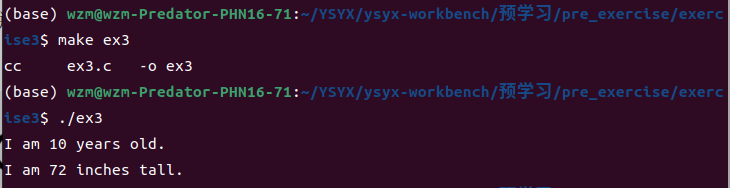
附加题1（文件夹2\_2）





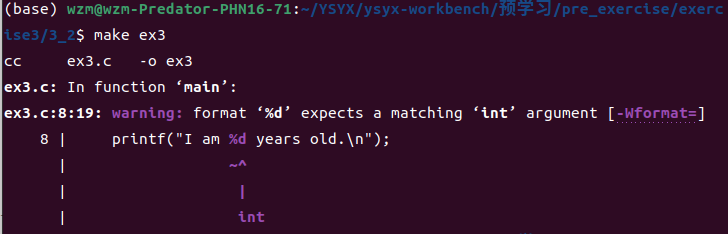
### 练习3

基础练习（文件夹3\_1）

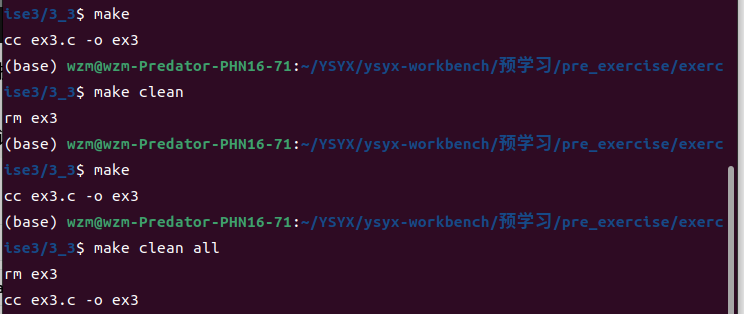


崩溃练习（文件夹3\_2）

去除age后

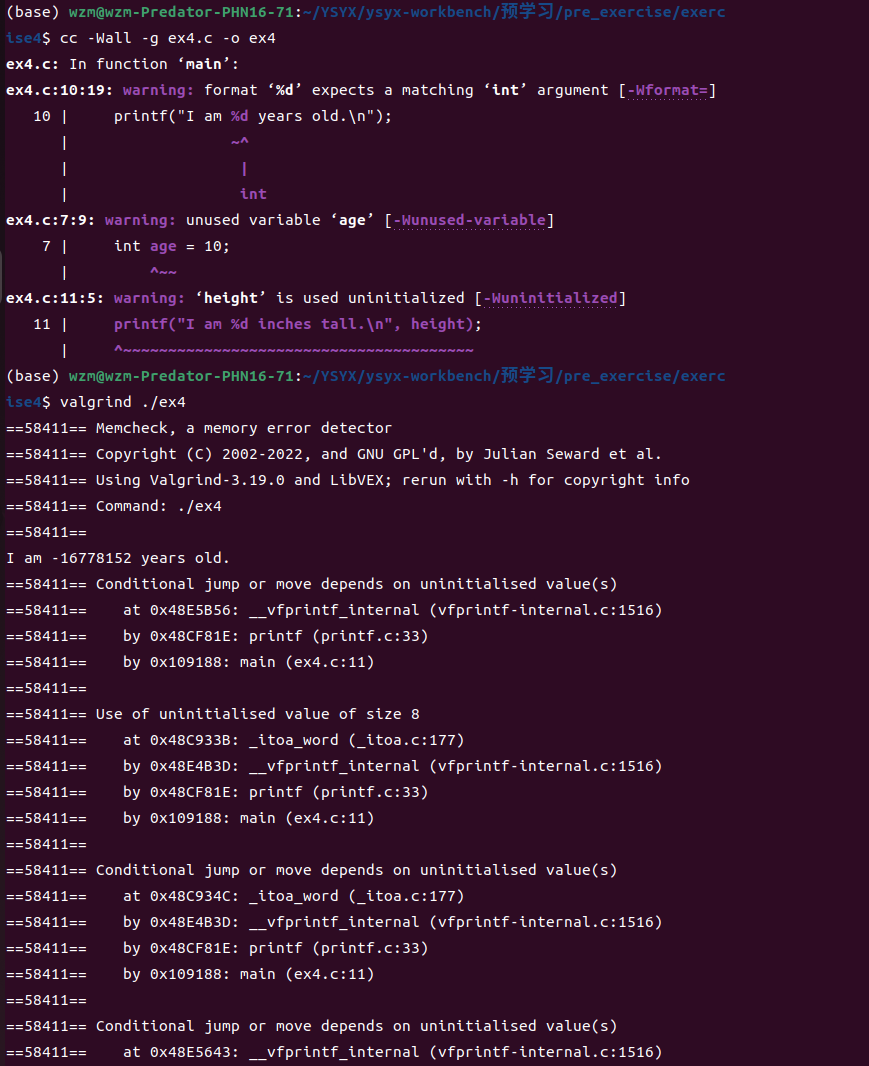


make clean / make clean all（文件夹3\_3）

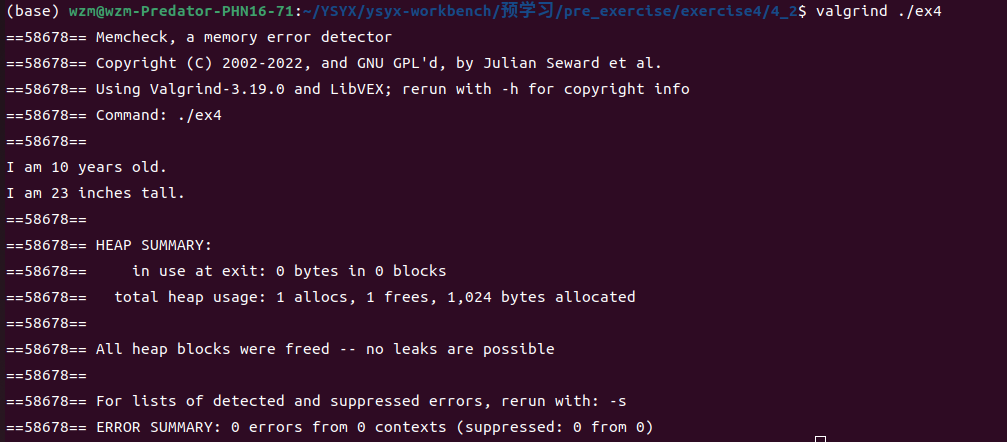


### 练习4

基础练习（文件夹4\_1）

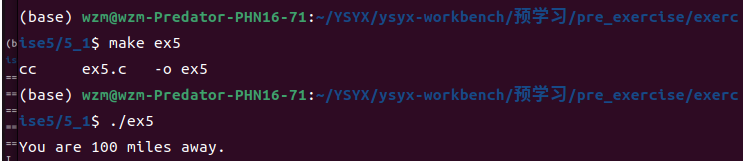


修复后（文件夹4\_2）



### 练习5

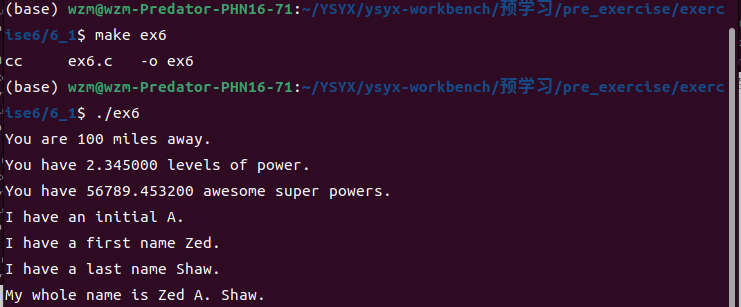
基础练习（文件夹5\_1）



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：

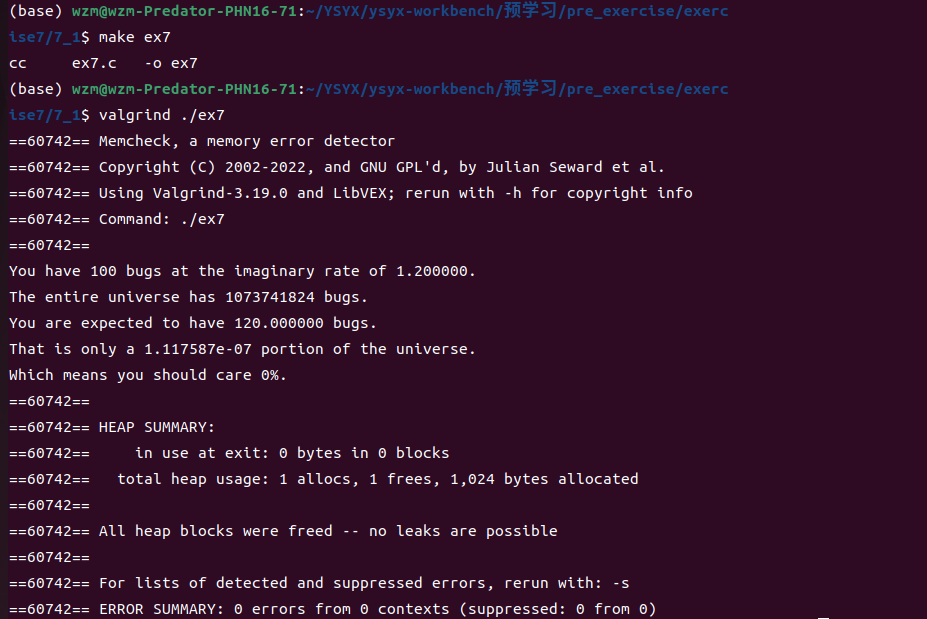
- 宽度字段：指定输出字段的最小宽度，可以使用数字来表示。例如，`%5d` 表示输出一个最小宽度为 5 的十进制整数。

- 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。

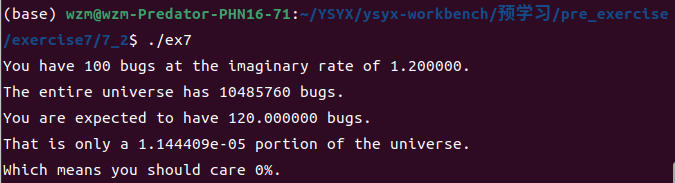
### 练习7

基础练习（文件夹7\_1）

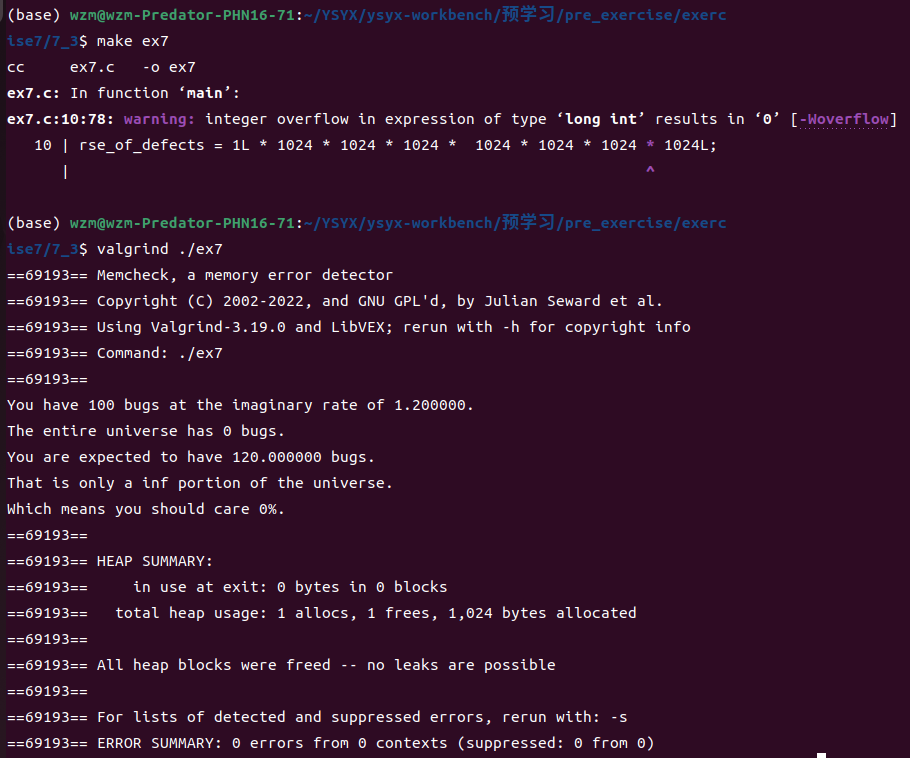


修改后（文件夹7\_2）





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。

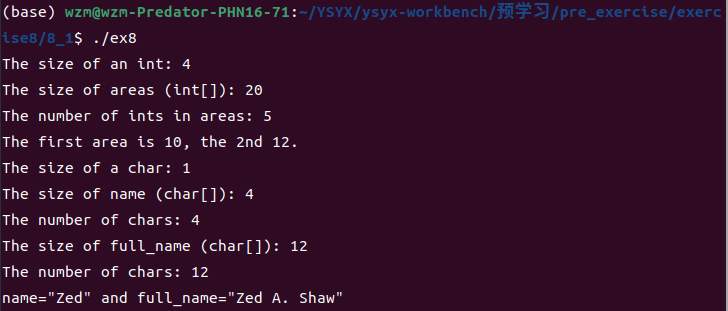
修改算术运算规则：在无符号整数类型之间进行算术运算时，通常采用模运算的方式。例如，无符号整数的加法、减法和乘法都会执行模运算，超出范围的值将被截断。

**char与int相乘**

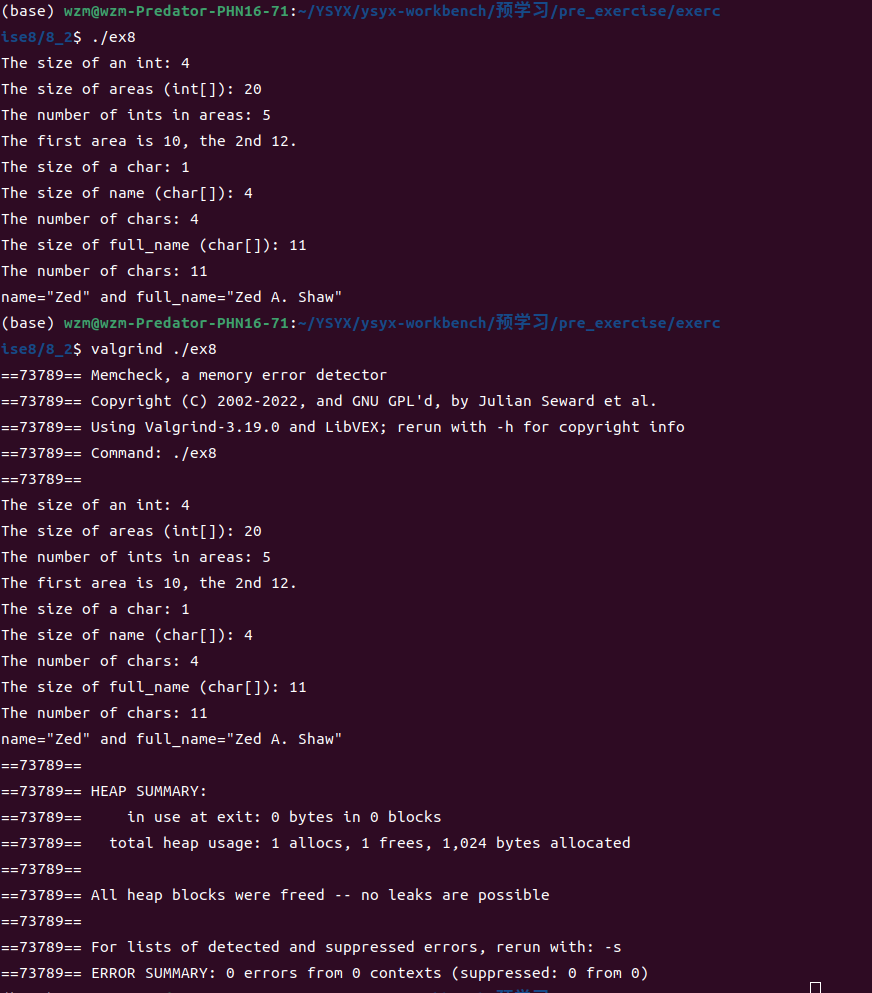
char和int相乘是char根据编码转化为对应的数，提升为int型，从而与int相乘，得到一个int型的数。

### 练习8

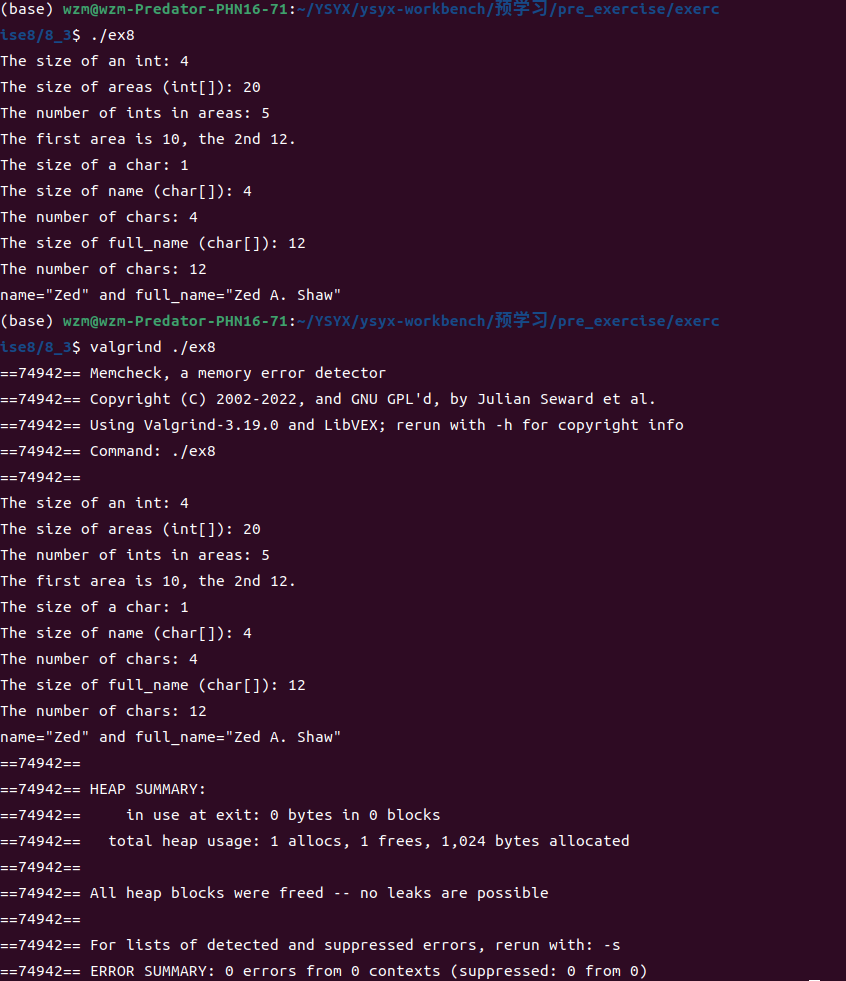
基础练习（文件夹8\_1）



删去’\0’未报错（文件夹8\_2）

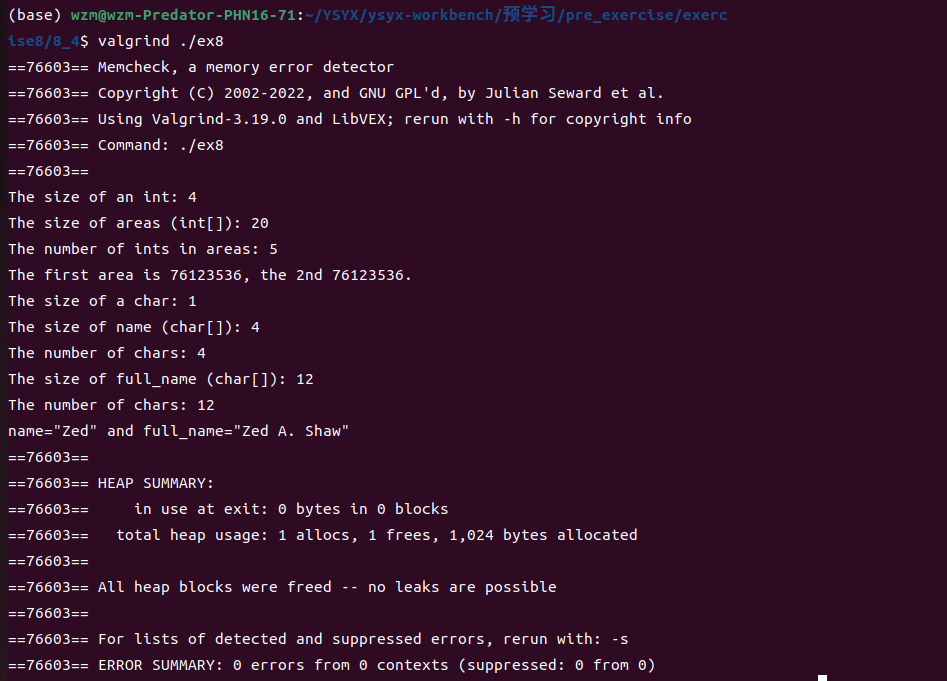


full\_name 转移到上面未报错（文件夹8\_3）



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

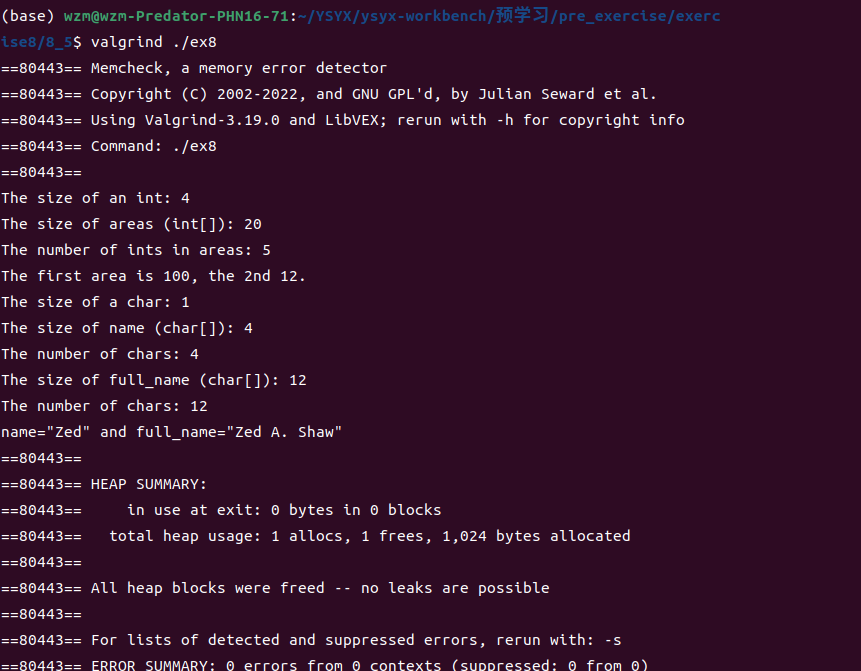


name与full\_name变式略

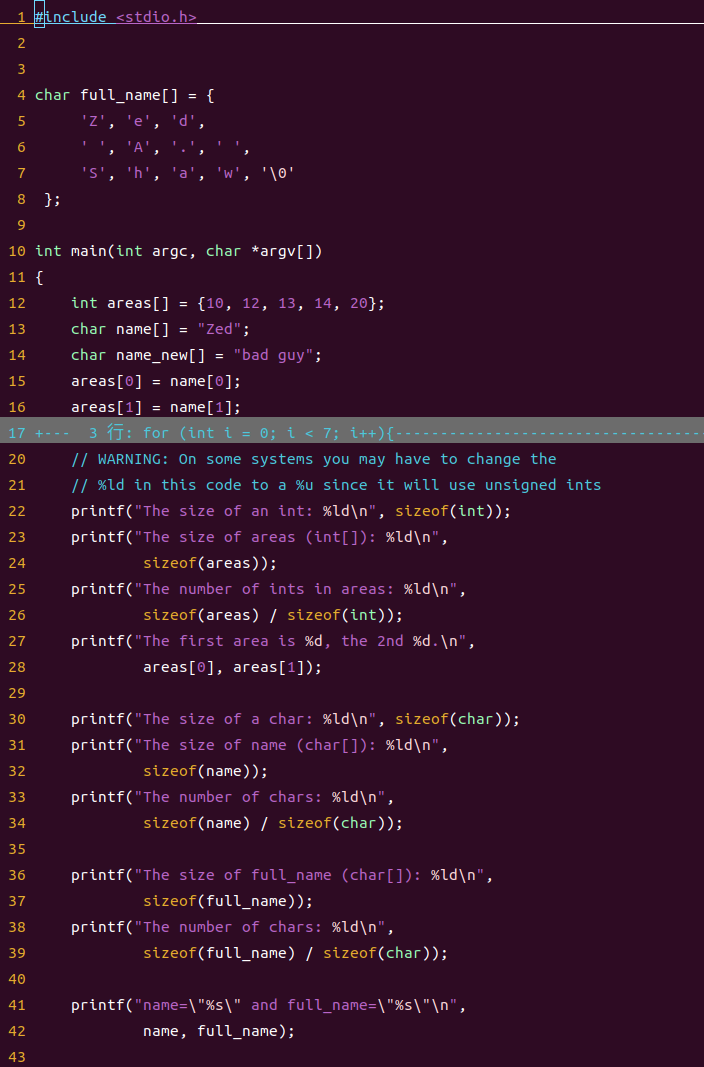
**附加题**

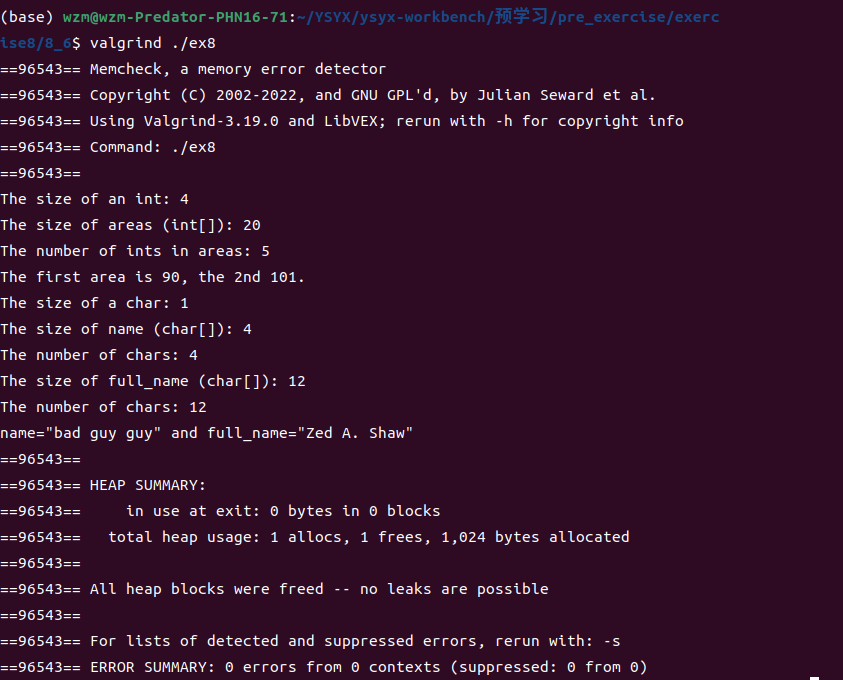
尝试使用areas[0] = 100;以及相似的操作对areas的元素赋值（文件夹8\_5）

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）





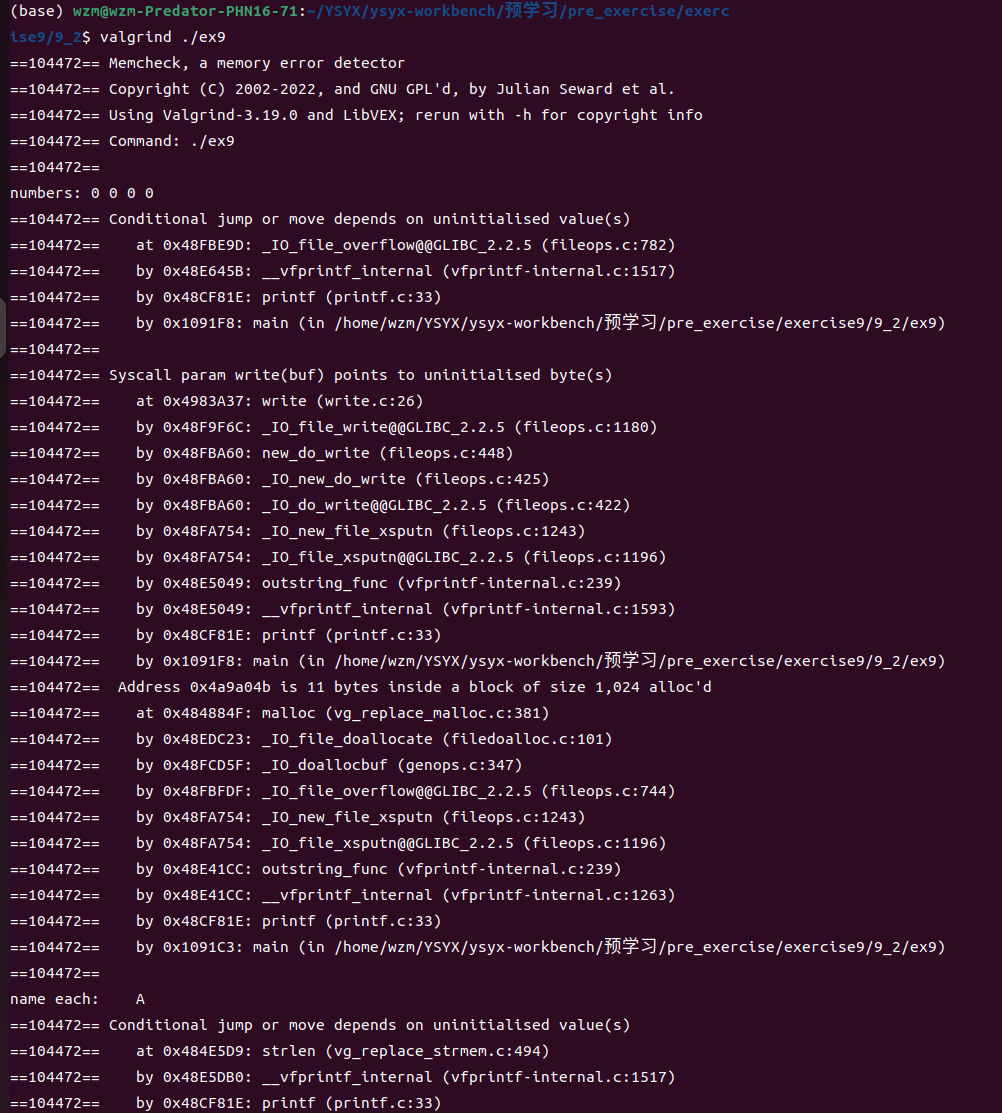
### 练习9

基础练习（文件夹9\_1）



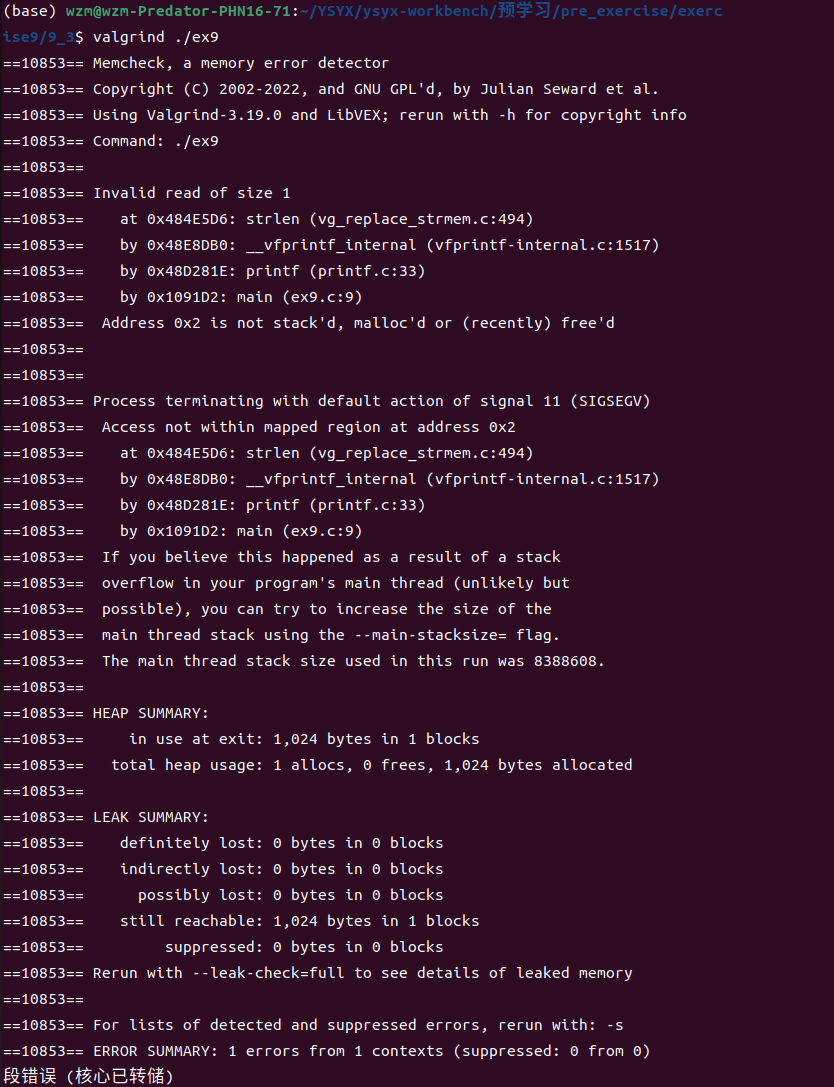
崩溃练习（文件夹9\_2）

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）

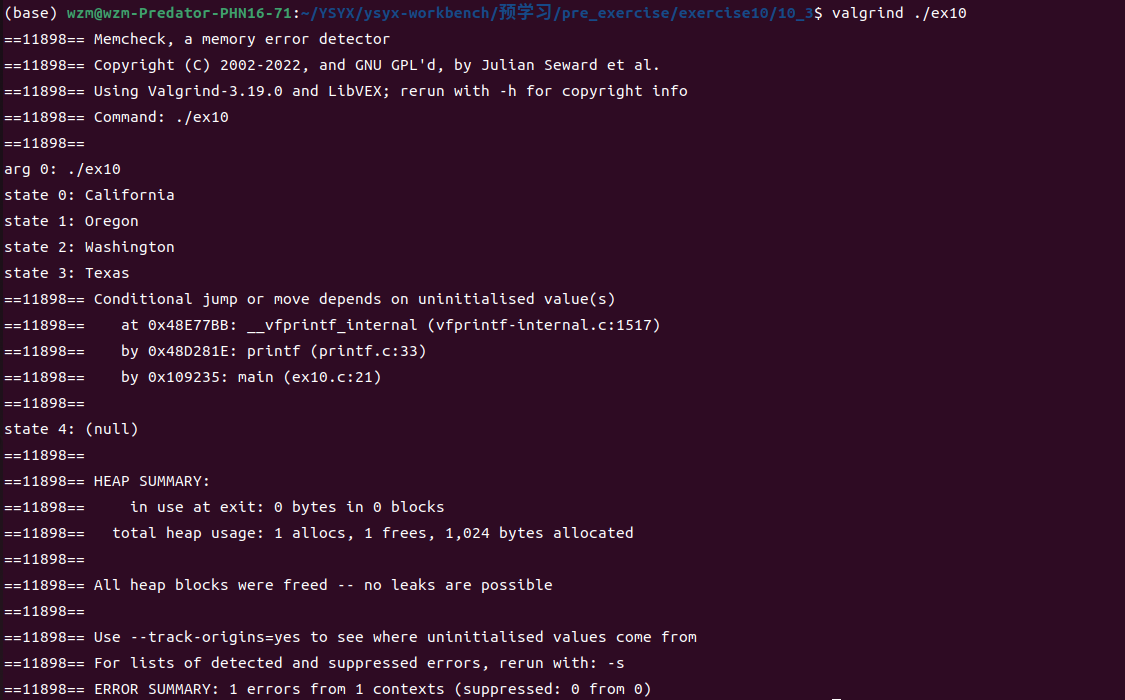


崩溃练习1：将i初始化为0（文件夹10\_2）



可以看到，程序正常运行，输入大概是被存储到arg[0]的位置

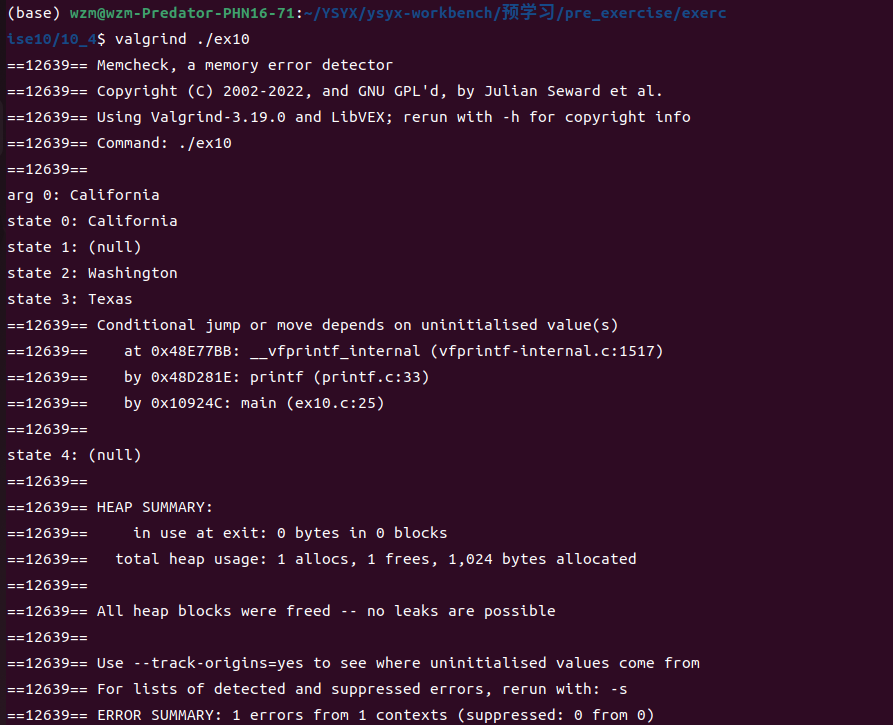
崩溃练习2：将num\_states改为错误的值使它变大（文件夹10\_3）



可以看到，标记未初始化。

**附加题：**

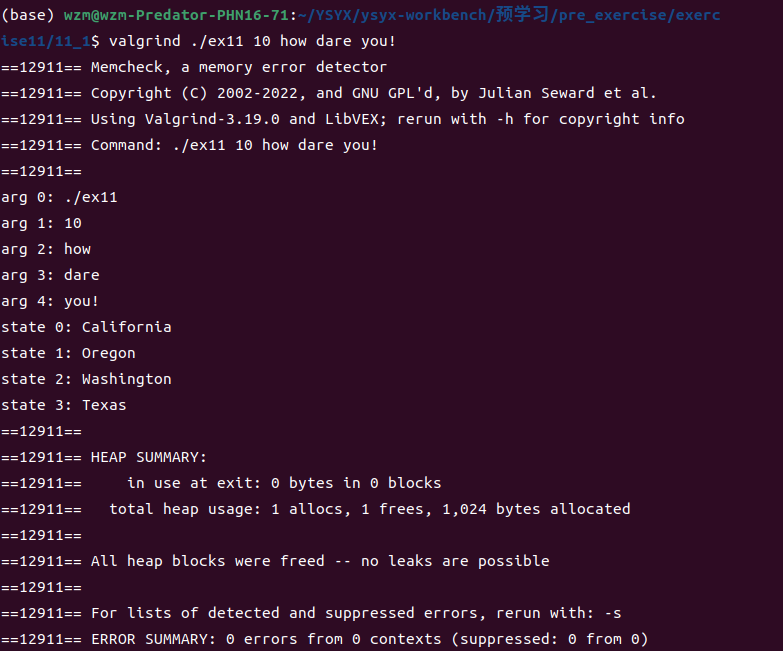
看看你是否能在打印之前将states的一个元素赋值给argv中的元素，再试试相反的操作（文件夹10\_4）



可以看到，报出未初始化的错误

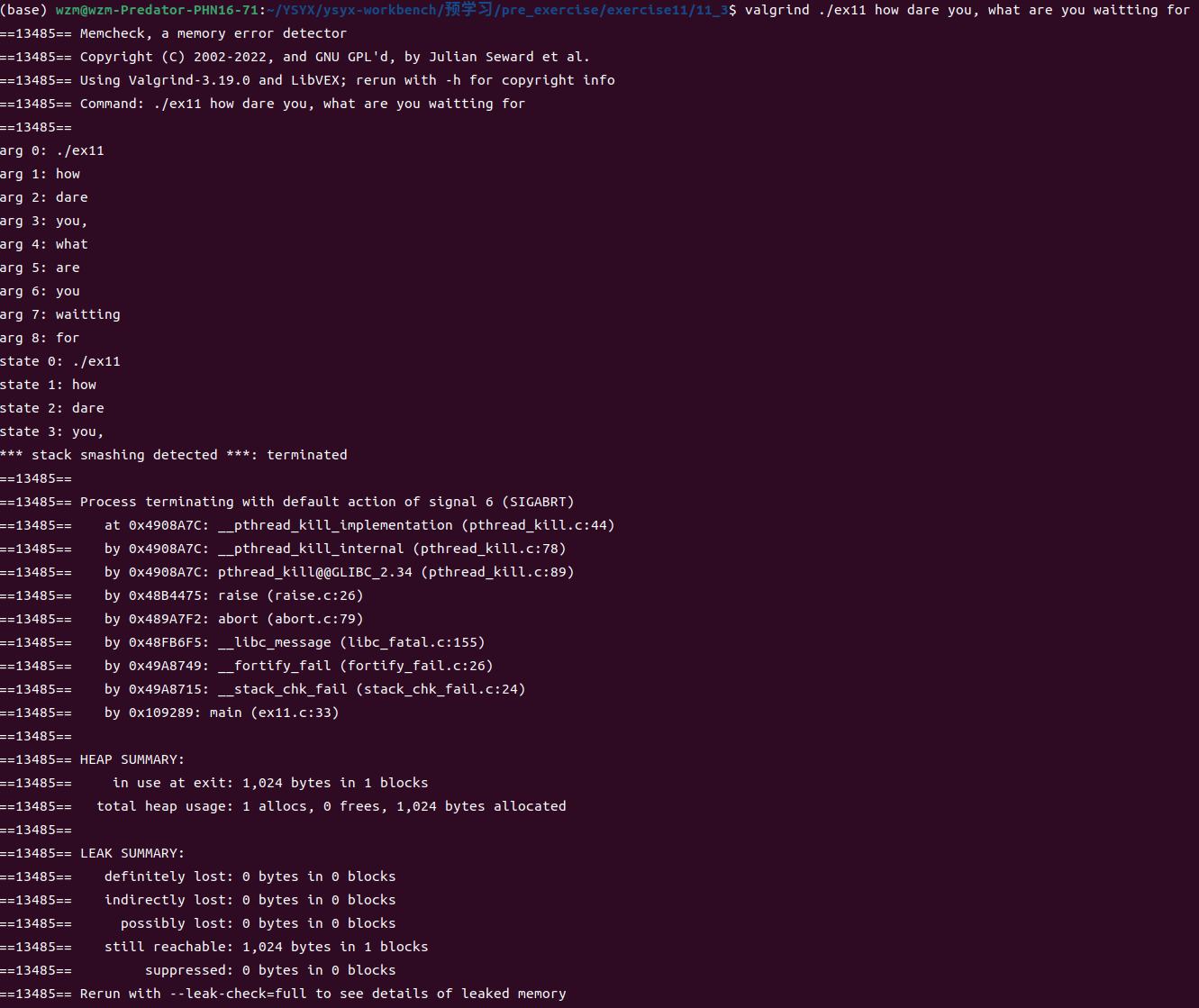
### 练习11

基础练习（文件夹11\_1）



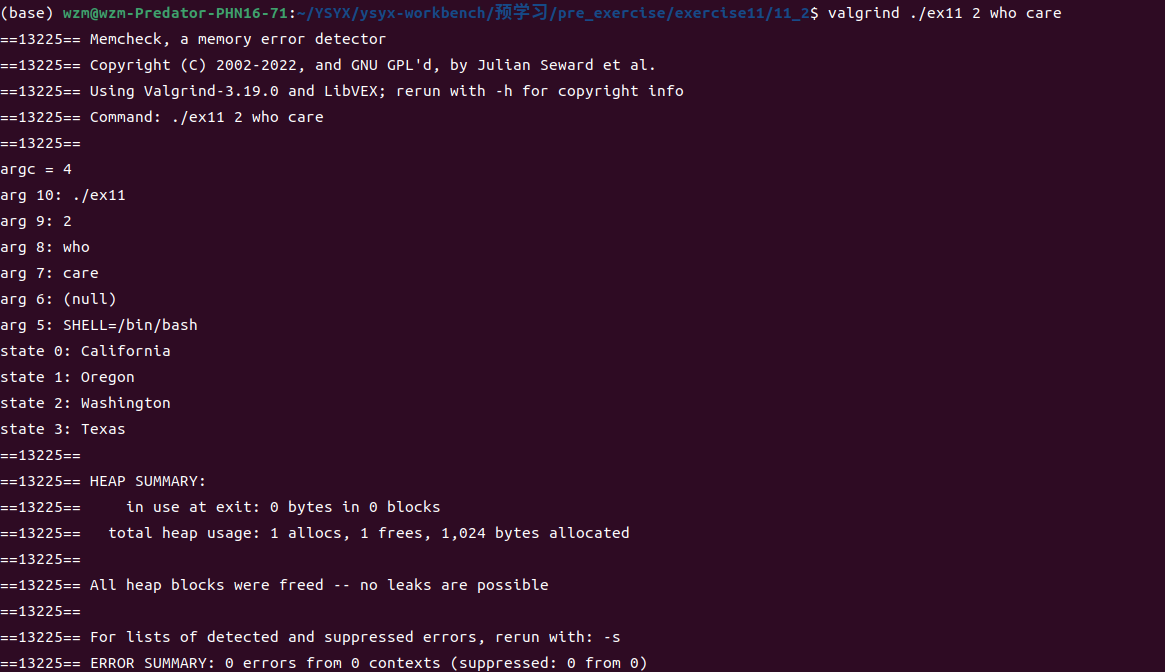
**附加题**

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）



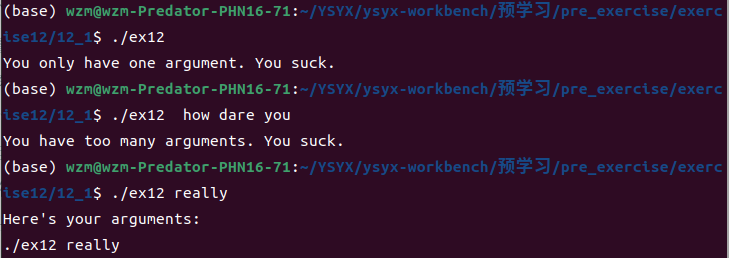
可以看到，正常工作。

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

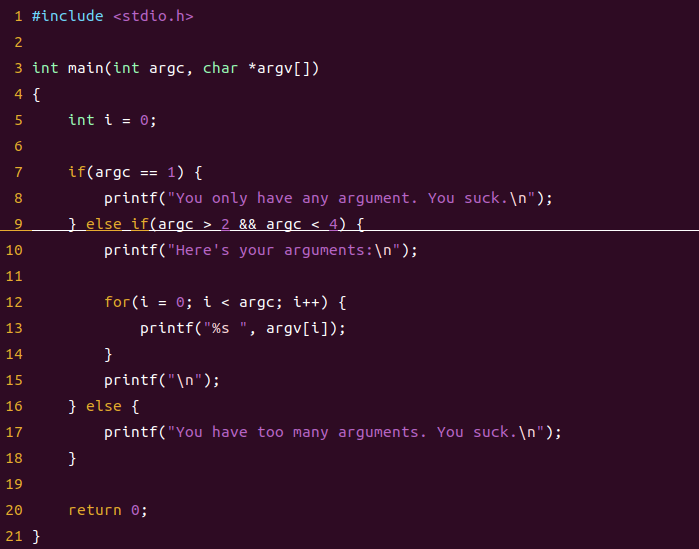
基础练习（文件夹12\_1）



**附加题**

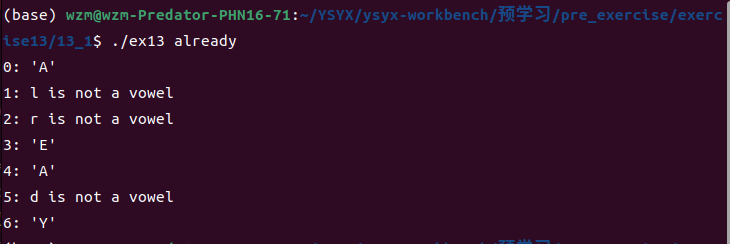
第一个判断所输出的话真的正确吗？由于你的“第一个参数”不是用户输入的第一个参数，把它改正

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：



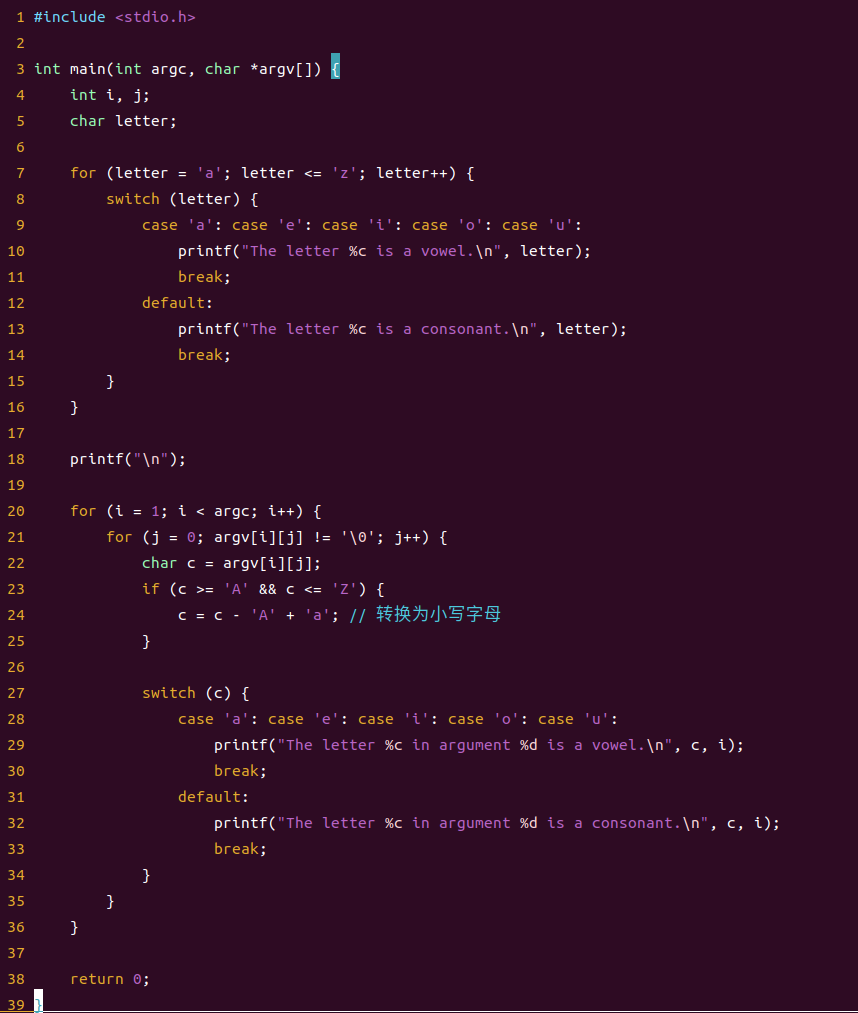
### 练习13

基础练习（文件夹13\_1）



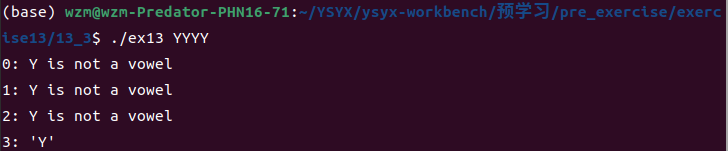
**附加题**

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）



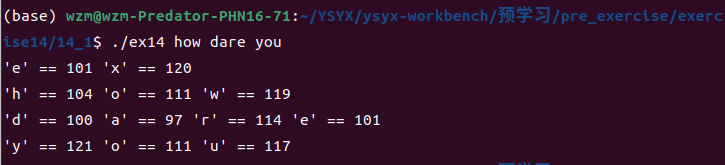
在“Y”的例子中，我在if代码块外面写了个break。这样会产生什么效果？如果把它移进if代码块，会发生什么？（文件夹13\_3）

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。



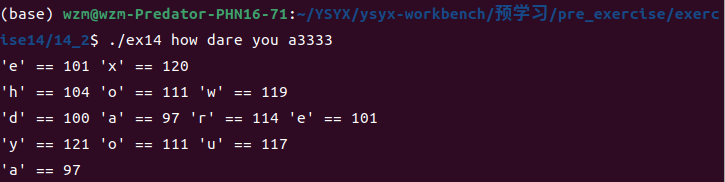
### 练习14

基础练习（文件夹14\_1）

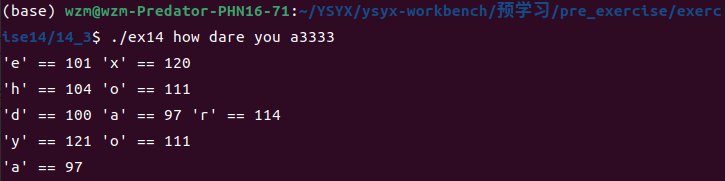


**附加题**

重新编写这些函数，使它们的数量减少。（文件夹14\_2）

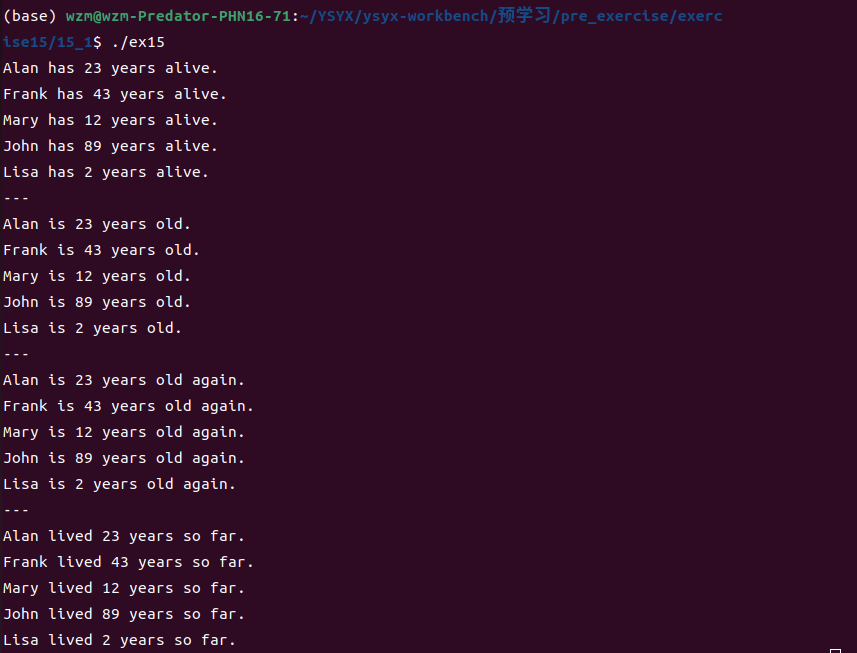


使用strlen函数，让print\_arguments知道每个字符串参数都有多长，之后将长度传入print\_letters。然后重写print\_letters，让它只处理固定的长度，不按照'\0'终止符。你需要#include <string.h>来实现它（文件夹14\_3）



### 练习15

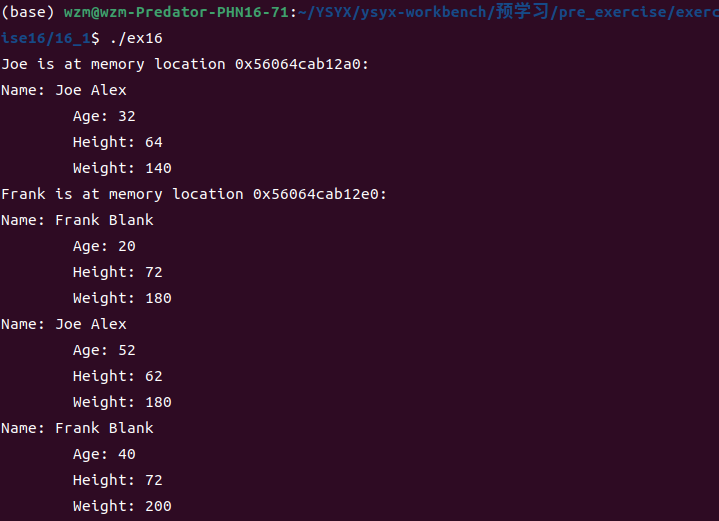
基础练习（文件夹15\_1）



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）



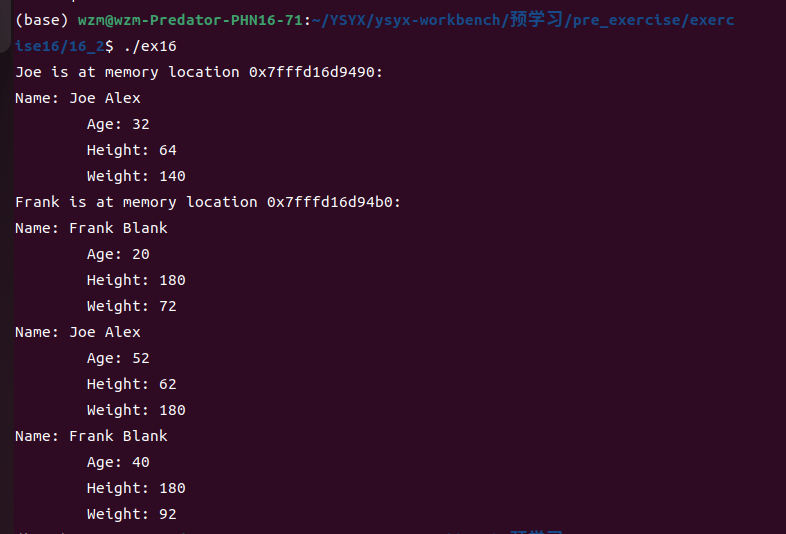
**附加题**

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。

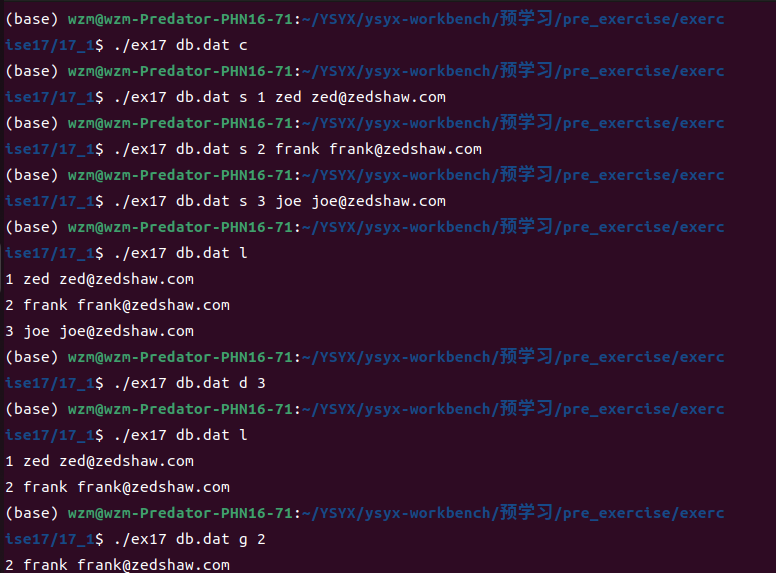
如何不使用指针来将结构体传给其它函数。（文件夹16\_2）

代码见文件夹中ex16.c文件，输出结果如下：



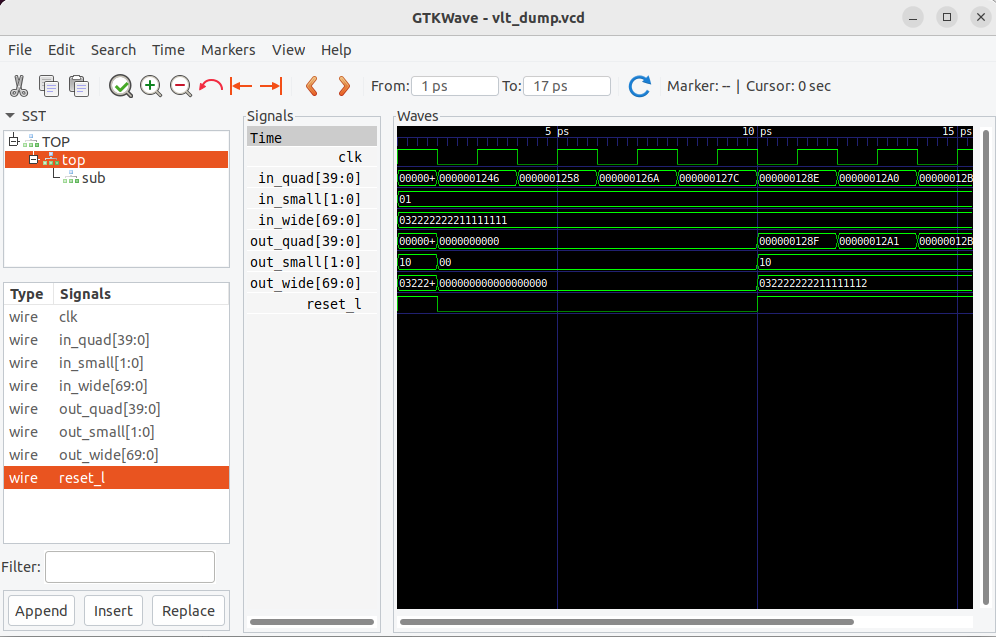
### 练习17

基础练习（文件夹17\_1）

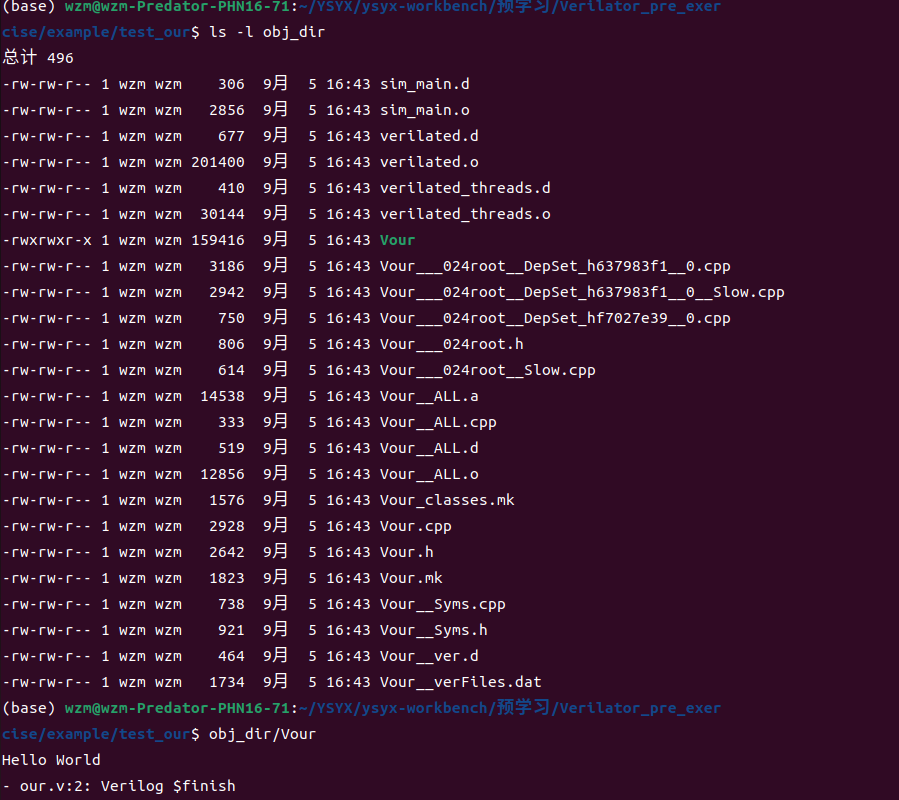


## Verilator环境搭建

源代码中例子



例程运行后例子



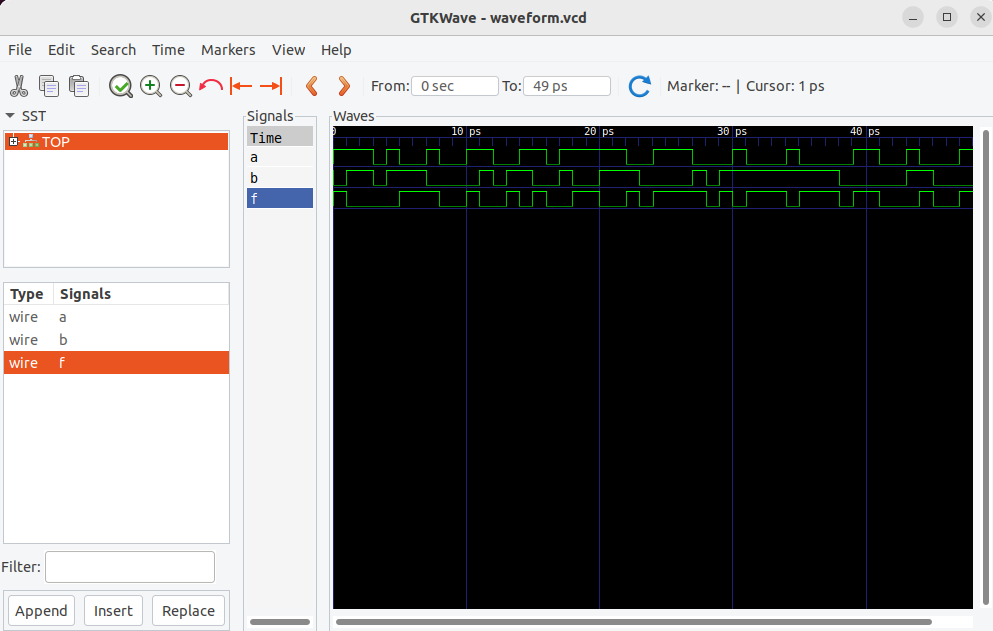
### 双控开关

进入目录

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

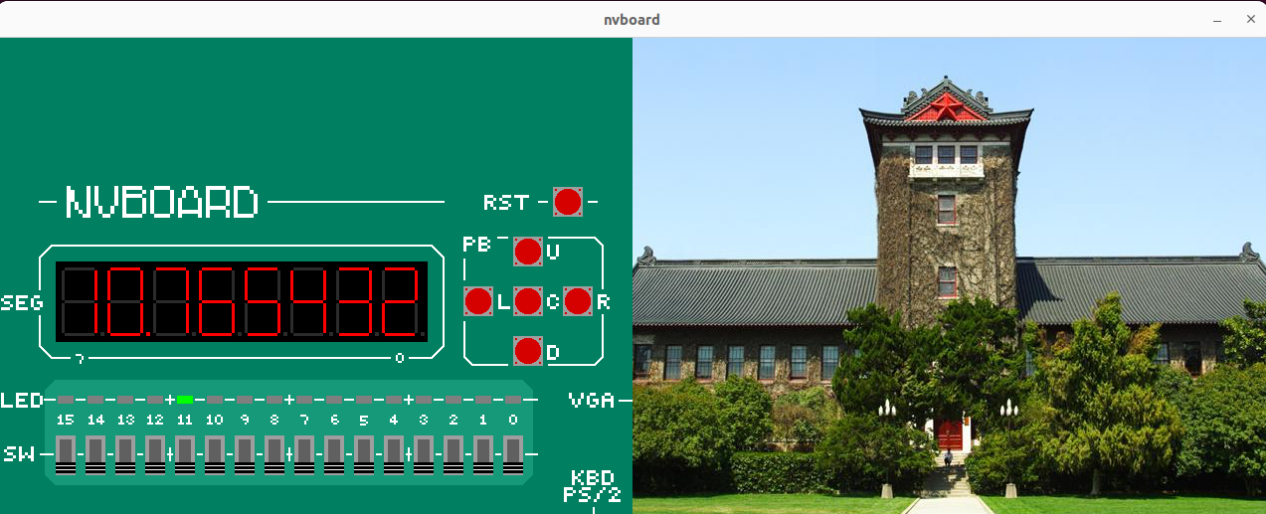
运行start.sh，一键编译、执行、波形图打开

波形图如下所示：

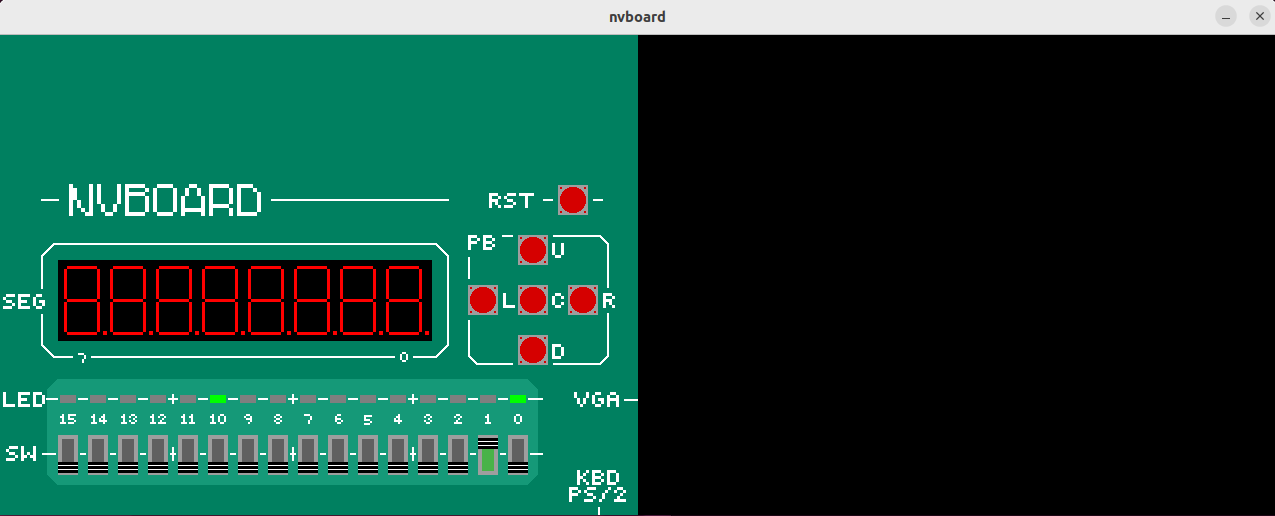


另外，由于本次没有使用makefile（不能打开锁文件），故在YSYX文件夹下完成文件提交。

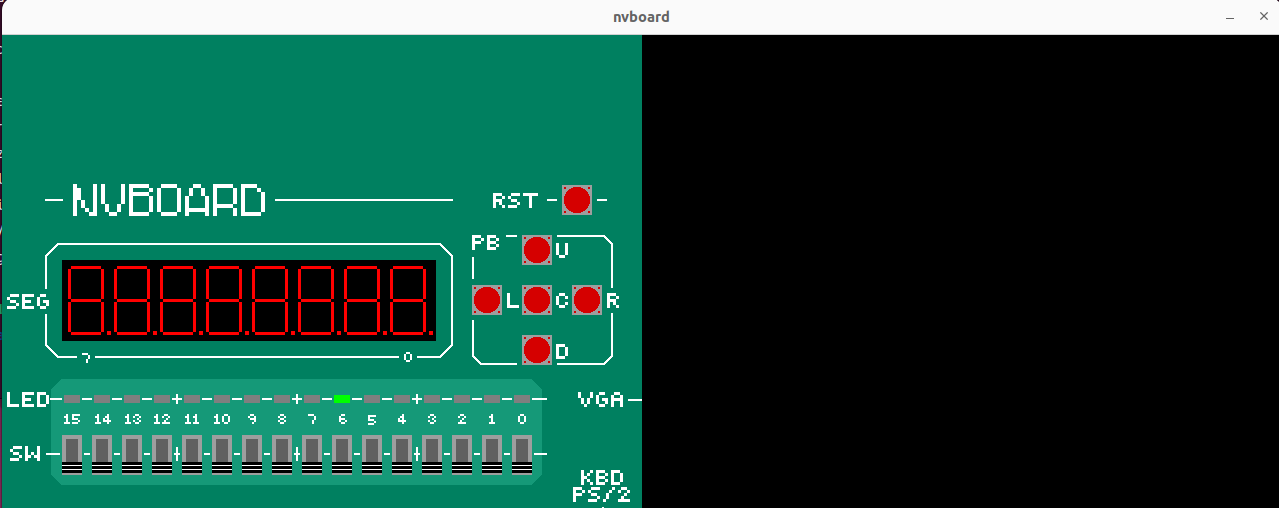
### NVBoard Example

make run之后，得到如下结果：  


### NVBoard 双控开关

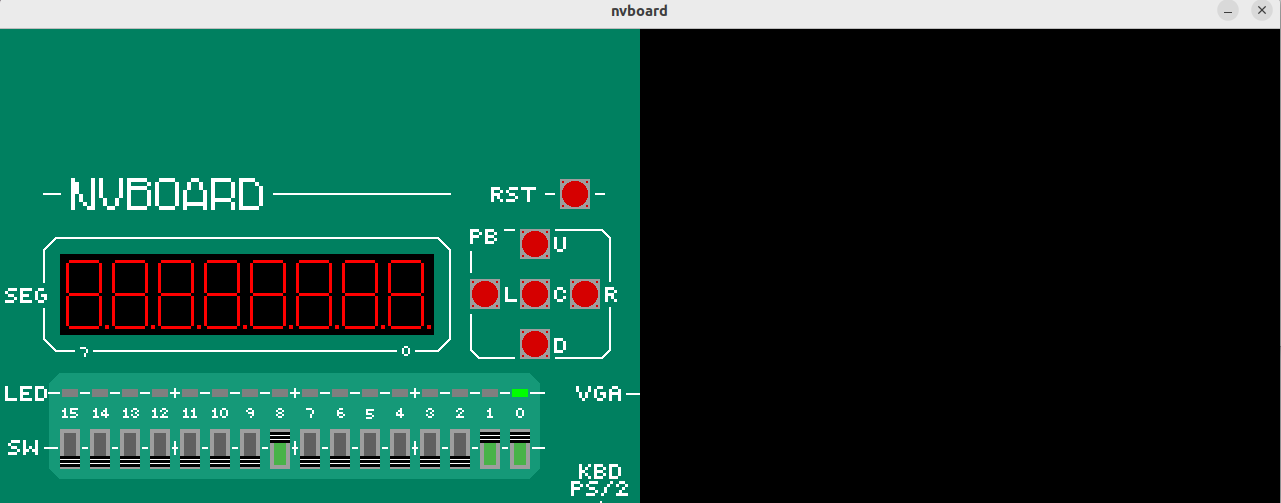


### NVBoard 流水灯

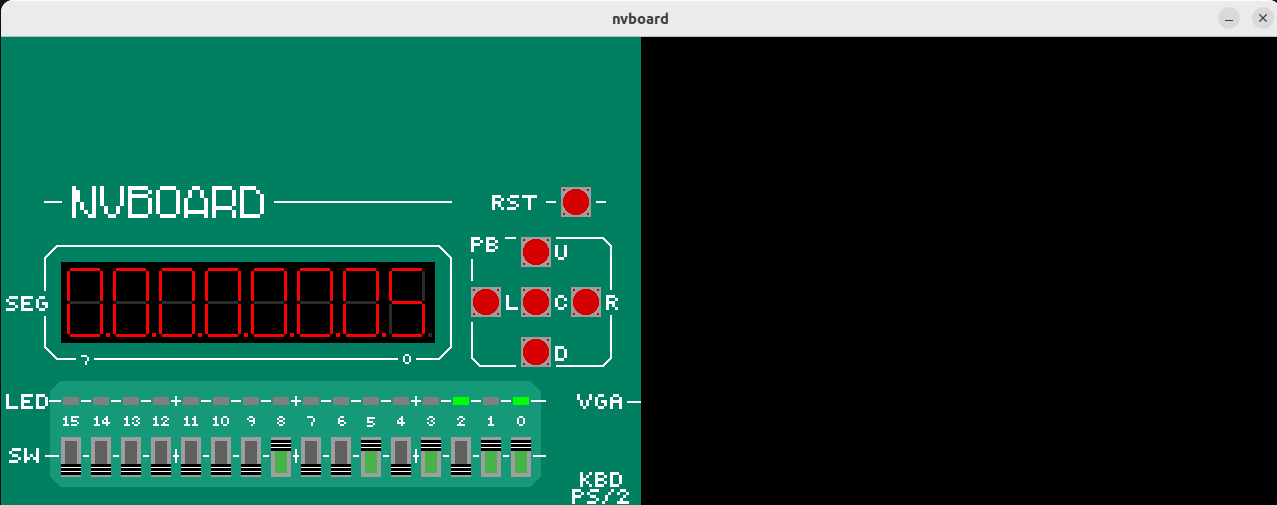


## 数字电路实验

### 实验一



### 实验二



### 实验三

