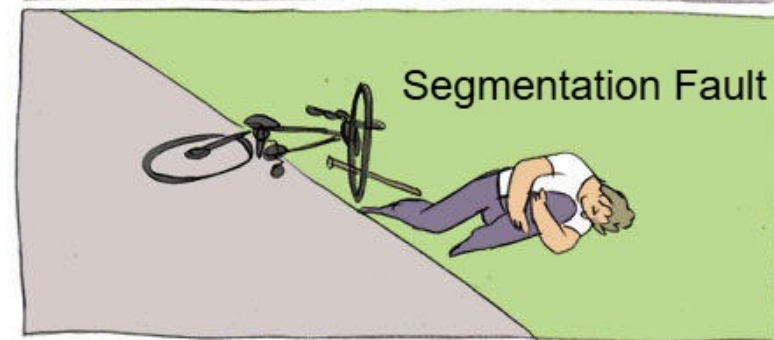
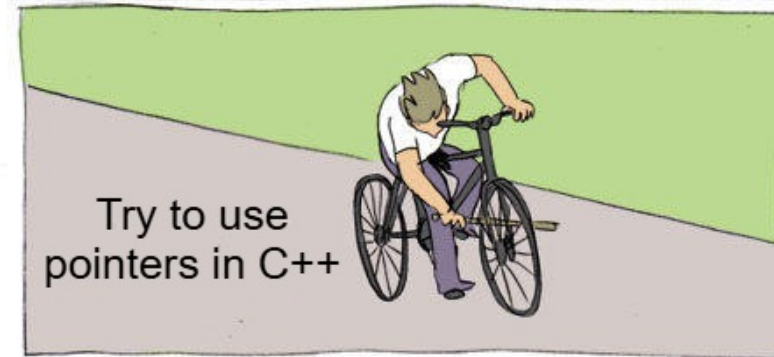
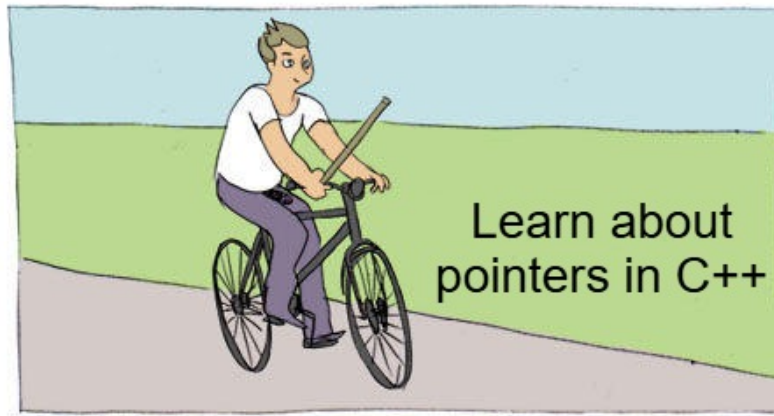




TODO

- Assignment 2 assigned due October 7

Overview for today

- Discuss Lab 2
- In-class debugging lab

A pointer on the heap that points to an int on the heap with value 60

```
int *p1 = malloc(sizeof(int));
```

Stack

(local variables)

p1

(int *)

0x0804A008

Heap

(dynamically allocated)

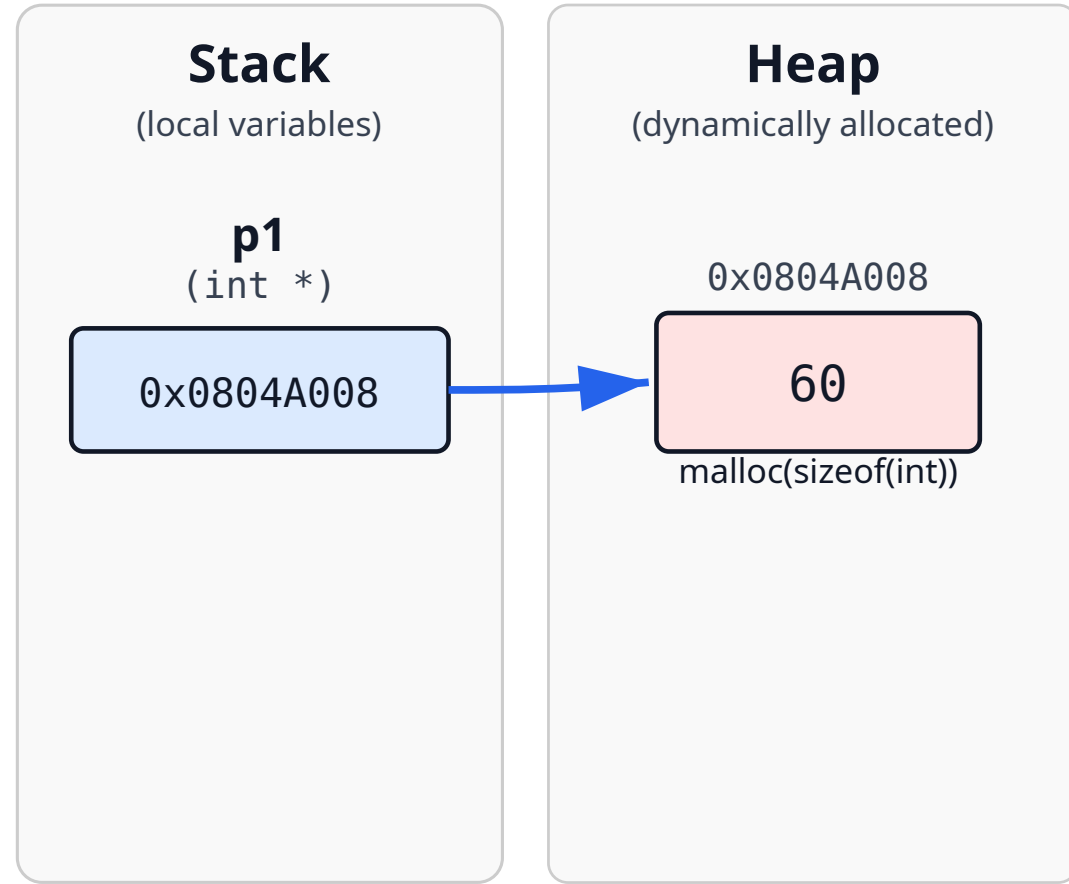
0x0804A008

malloc(sizeof(int))



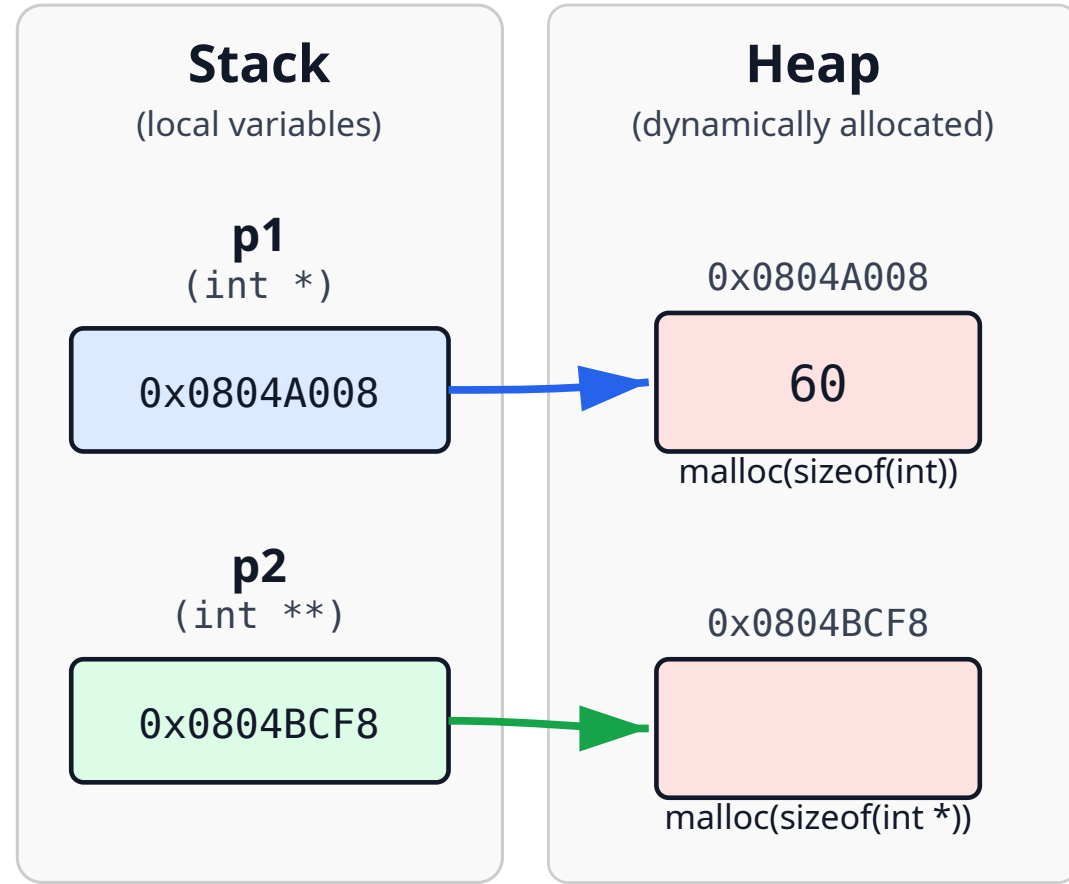
A pointer on the heap that points to an int on the heap with value 60

```
int *p1 = malloc(sizeof(int));  
*p1 = 60;
```



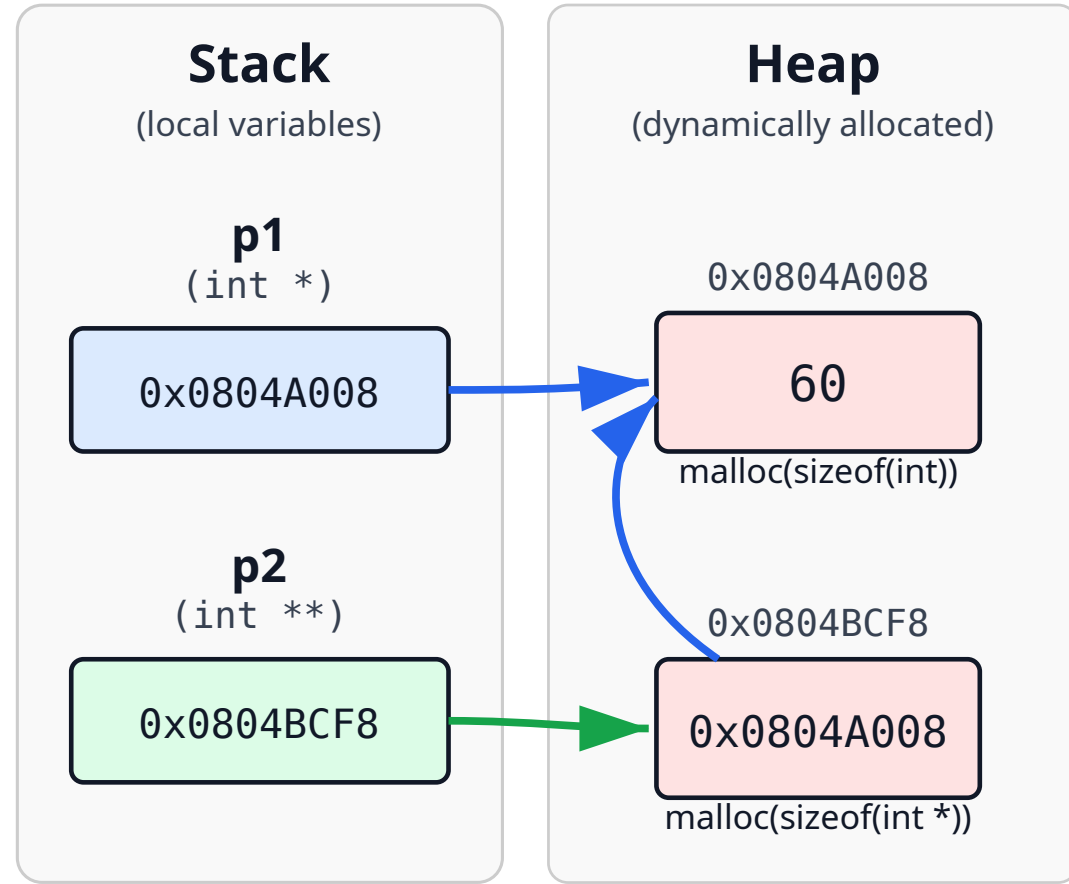
A pointer on the heap that points to an int on the heap with value 60

```
int *p1 = malloc(sizeof(int));  
*p1 = 60;  
int **p2 = malloc(sizeof(int *));
```



A pointer on the heap that points to an int on the heap with value 60

```
int *p1 = malloc(sizeof(int));  
*p1 = 60;  
int **p2 = malloc(sizeof(int *));  
*p2 = p1;
```



Solution

```
int *p = malloc(sizeof(int));  
*p = 60;  
int **p2; = malloc(sizeof(int *));  
*p2 = p;
```

Why not?

```
int *p = malloc(sizeof(int));
```

```
*p = 60;
```

```
int **p2; = malloc(sizeof(int *));
```

```
*p2 = &p;
```

Why not?

```
int *p = malloc(sizeof(int));  
*p = 60;  
int **p2; = malloc(sizeof(int *));  
*p2 = &p;  
// *p2 is of type int *  
// &p is of type int ** (type error)
```

Why not?

```
int *p = malloc(sizeof(int));  
*p = 60;  
int **p2; = malloc(sizeof(int *));  
*p2 = &p;  
// p is on the stack  
// so p2 points to the stack
```

Lab 2

- Will help solidify our understanding of:
 - C strings
 - Pointers
 - `malloc()/free()`

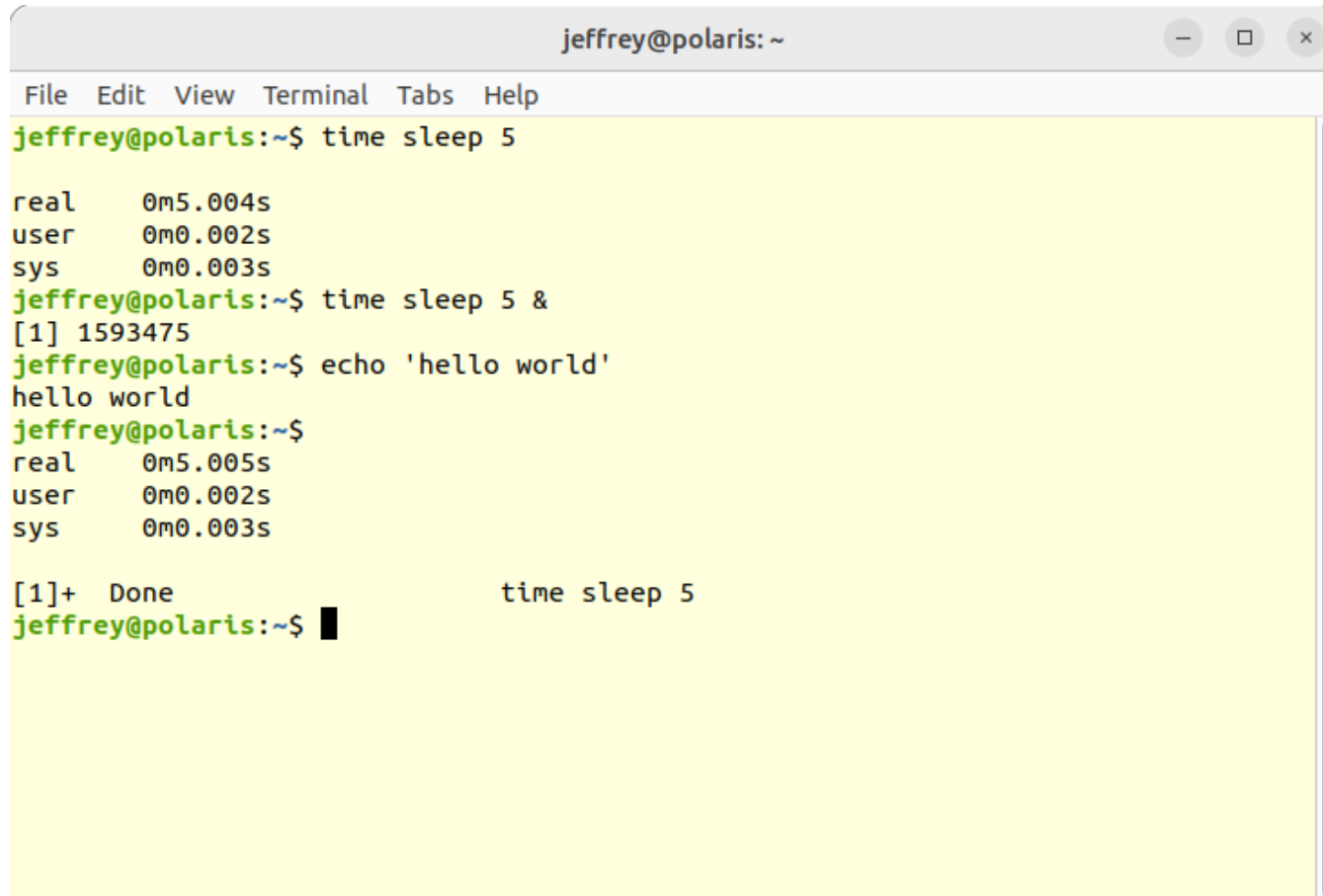
Lab 2

- More of a “conventional” lab
- Mostly programming
- Conceptual challenges:
 - Working with C
 - Working with pointers and pointer arithmetic
 - Working with `malloc()` and `free()`

Lab 2

```
j.knockel@krebs: ~/assignments/lab2
File Edit View Terminal Tabs Help
j.knockel@krebs:~/assignments/lab2$ ls -l -a
total 52
drwxr-xr-x. 2 j.knockel others 115 Sep 26 13:24 .
drwxr-xr-x. 6 j.knockel others 54 Sep 24 23:02 ..
-rw-----. 1 j.knockel others 3280 Sep 26 13:22 command.c
-rw-----. 1 j.knockel others 731 Sep 20 13:20 command.h
-rw-r--r--. 1 j.knockel others 8968 Sep 23 16:04 command.o
-rwxr-xr-x. 1 j.knockel others 23056 Sep 23 16:05 command_test
-rw-----. 1 j.knockel others 3307 Sep 23 16:05 command_test.c
-rw-----. 1 j.knockel others 434 Sep 27 09:13 Makefile
j.knockel@krebs:~/assignments/lab2$ ls -l -a
total 52
drwxr-xr-x. 2 j.knockel others 115 Sep 26 13:24 .
drwxr-xr-x. 6 j.knockel others 54 Sep 24 23:02 ..
-rw-----. 1 j.knockel others 3280 Sep 26 13:22 command.c
-rw-----. 1 j.knockel others 731 Sep 20 13:20 command.h
-rw-r--r--. 1 j.knockel others 8968 Sep 23 16:04 command.o
-rwxr-xr-x. 1 j.knockel others 23056 Sep 23 16:05 command_test
-rw-----. 1 j.knockel others 3307 Sep 23 16:05 command_test.c
-rw-----. 1 j.knockel others 434 Sep 27 09:13 Makefile
j.knockel@krebs:~/assignments/lab2$
```

Lab 2

A terminal window titled 'jeffrey@polaris: ~' with standard window controls. The terminal shows a sequence of commands and their outputs. The first command 'time sleep 5' is followed by timing statistics. The second command 'time sleep 5 &' is followed by a job ID '[1] 1593475'. The third command 'echo 'hello world'' outputs 'hello world'. The fourth command 'time sleep 5' is followed by another set of timing statistics. The prompt '[1]+ Done' indicates the background process has finished. The final prompt 'jeffrey@polaris:~\$' is followed by a cursor.

```
jeffrey@polaris: ~
File Edit View Terminal Tabs Help
jeffrey@polaris:~$ time sleep 5

real    0m5.004s
user    0m0.002s
sys     0m0.003s
jeffrey@polaris:~$ time sleep 5 &
[1] 1593475
jeffrey@polaris:~$ echo 'hello world'
hello world
jeffrey@polaris:~$
real    0m5.005s
user    0m0.002s
sys     0m0.003s

[1]+  Done                  time sleep 5
jeffrey@polaris:~$ █
```

Lab 2

```
Python 3.12.3 (main, Aug 31 2026, 10:18:26) [GCC 13.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> ' some command line '.split()
['some', 'command', 'line']
>>>
```

- And then decide if the invocation is foreground or background (has a '&' at the end)

math.c

Debugging

- We will learn some dedicated debugging tools:
 - gdb (GNU debugger)
 - valgrind
- Other languages have debugging tools too, but C needs them more than others

debugdemo.c

Lab D

- Fixing five bugs in five programs
- Using valgrind, gdb to help debug them
- Find the lab on the course web page!

TODO

- Assignment 2 assigned due October 7