CSCI 331:
Introduction to Computer Security

Lecture 3: More C

Instructor: Dan Barowy

Williams

Topics

More C

Announcements

- CS Colloquium **tomorrow @ 2:35pm in Wege Auditorium (TCL 123)**

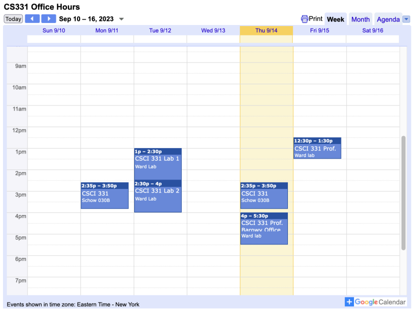
David Mimno (Cornell)

The data in data science: measuring the impact of data curation on large language model pretraining

Large language models like BERT and ChatGPT are fundamentally a reflection of the data used to train them. Putting together millions of documents from diverse sources requires innumerable choices. But because of the time and expense of the initial, general-purpose "pretraining" phase of model training, many of these choices are made heuristically without any systematic evidence-based justification. We train models to measure the effects of three common curation decisions: document age, quality and toxicity filtering, and data sources. We find that these choices have significant, noticeable effects that cannot be fully overcome by additional training.

Your to-dos

1. Second lab (Lab 1) is posted, **due 9/24**.
   i. Read chapters on C if you feel like you need a refresher.
2. I have office hours today after class.



CS331 Office Hours

Sep 10 – 16, 2023

2:35p – 3:50p CSCI 331 Schow 030B

1p – 2:30p CSCI 331 Lab 1 Ward Lab

2:30p – 4p CSCI 331 Lab 2 Ward Lab

2:35p – 3:50p CSCI 331 Schow 030B

4p – 5:30p CSCI 331 Prof. Barowy Office Ward lab

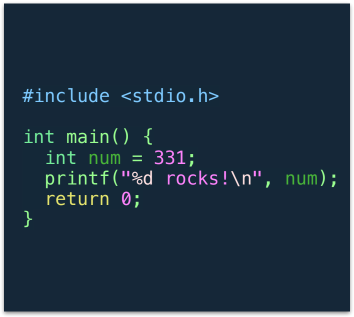
12:30p – 1:30p CSCI 331 Prof. Ward lab

Events shown in time zone: Eastern Time - New York

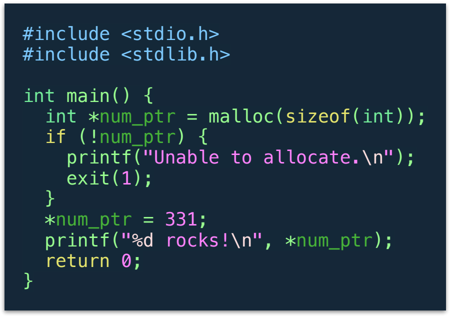
C rules from last class

0. Pointers are for **pointing at** other values in **memory**.
1. Whenever you **store** a **variable**, you **always** ask C to **reserve memory** for some **duration**.

Activity: What **effect** do these programs have on **memory**?



```
#include <stdio.h>

int main() {
  int num = 331;
  printf("%d rocks!\n", num);
  return 0;
}
```

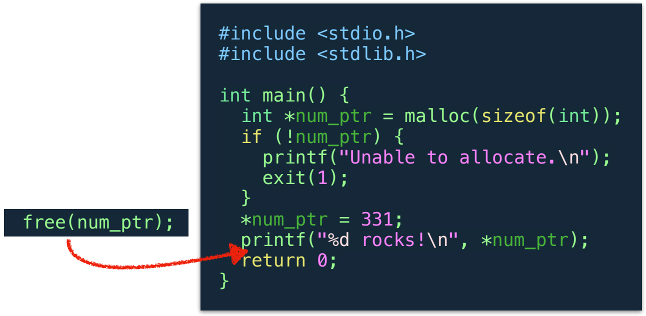


```
#include <stdio.h>
#include <stdlib.h>

int main() {
  int *num_ptr = malloc(sizeof(int));
  if (!num_ptr) {
    printf("Unable to allocate.\n");
    exit(1);
  }
  *num_ptr = 331;
  printf("%d rocks!\n", *num_ptr);
  return 0;
}
```

Rule 2:

All long duration storage needs to be both **allocated** and **deallocated**.

Last class we spotted what was wrong here…



```
#include <stdio.h>
#include <stdlib.h>

int main() {
  int *num_ptr = malloc(sizeof(int));
  if (!num_ptr) {
    printf("Unable to allocate.\n");
    exit(1);
  }
  *num_ptr = 331;
  printf("%d rocks!\n", *num_ptr);
  return 0;
}
```

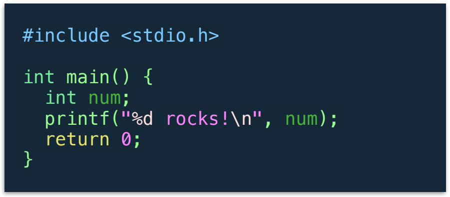
```
free(num_ptr);
```

Does this bug “matter” for this program?

Rule 3:

Always initialize variables.

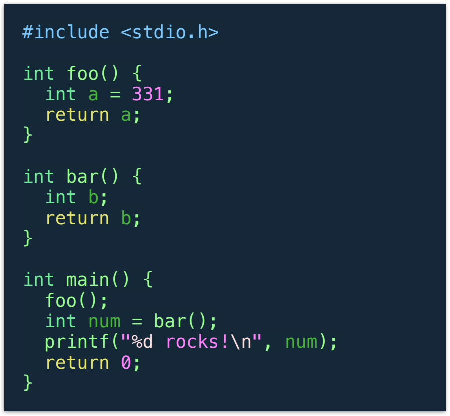
What does this program print?



```
#include <stdio.h>

int main() {
  int num;
  printf("%d rocks!\n", num);
  return 0;
}
```

(always? are you sure?)

This program prints “331 rocks!”

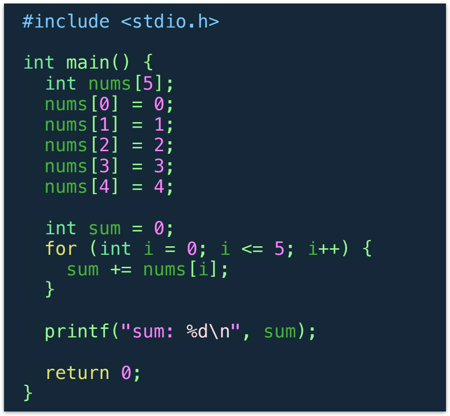


```
#include <stdio.h>

int foo() {
  int a = 331;
  return a;
}

int bar() {
  int b;
  return b;
}

int main() {
  foo();
  int num = bar();
  printf("%d rocks!\n", num);
  return 0;
}
```

Please do not write code like this!

Rule 4:

Watch out for **off-by-one** errors.



```
#include <stdio.h>

int main() {
  int nums[5];
  nums[0] = 0;
  nums[1] = 1;
  nums[2] = 2;
  nums[3] = 3;
  nums[4] = 4;

  int sum = 0;
  for (int i = 0; i <= 5; i++) {
    sum += nums[i];
  }

  printf("sum: %d\n", sum);

  return 0;
}
```

Effects range from **subtle** to **catastrophic**!

Rule 5:

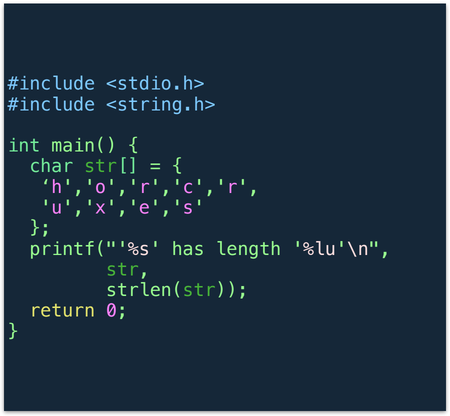
Always **null-terminate** “**C strings**.”

C has **no `String` data type**.
Instead, it has **character arrays**.
Character arrays must always be **null-terminated**.

(otherwise **bad things** happen)

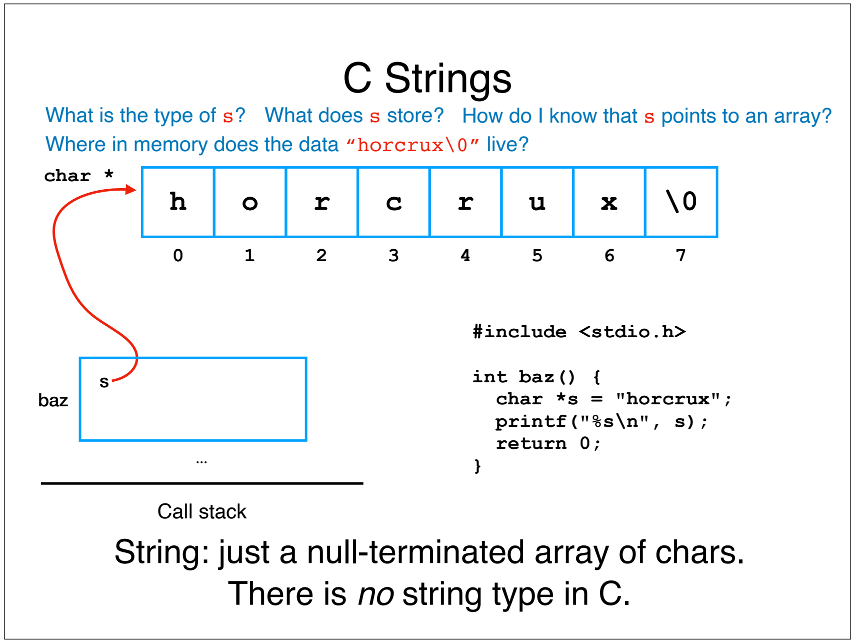
Rule 5:

Always **null-terminate** “**C strings**.”



```
#include <stdio.h>
#include <string.h>

int main() {
  char str[] = {
   ‘h','o','r','c','r',
   'u','x','e','s'
  };
  printf("'%s' has length '%lu'\n",
         str,
         strlen(str));
  return 0;
}
```

Effects range from **subtle** to **catastrophic**!



C Strings

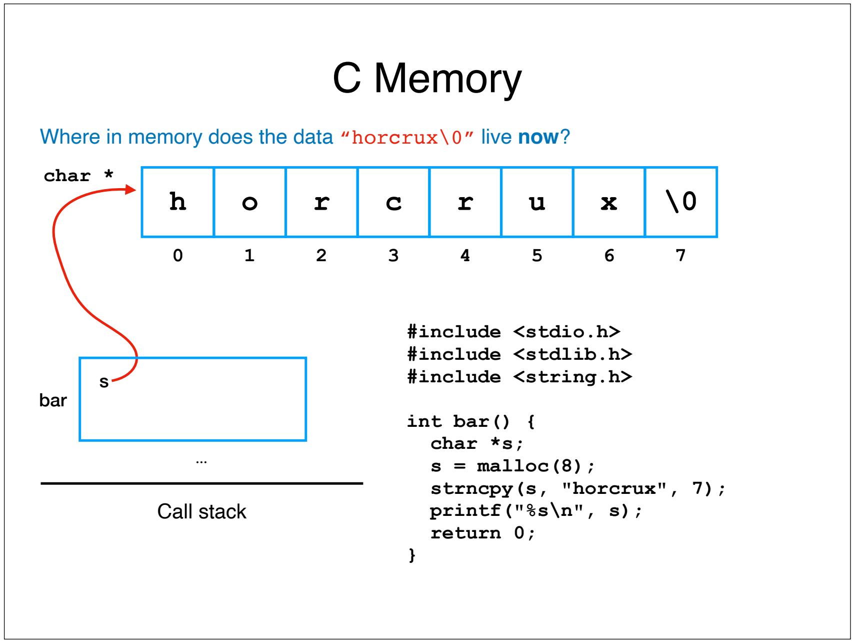
What is the type of s? What does s store? How do I know that s points to an array?
Where in memory does the data “horcrux\0” live?

char *

h	o	r	c	r	u	x	\0
0	1	2	3	4	5	6	7

```
#include <stdio.h>

int baz() {
  char *s = "horcrux";
  printf("%s\n", s);
  return 0;
}
```

String: just a null-terminated array of chars.
There is *no* string type in C.



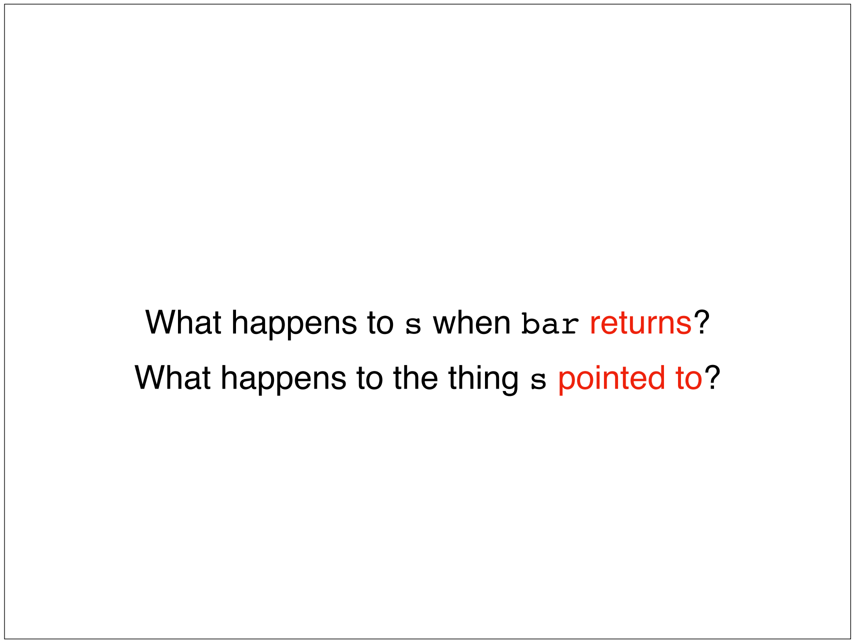
C Memory

Where in memory does the data “horcrux\0” live **now**?

char *

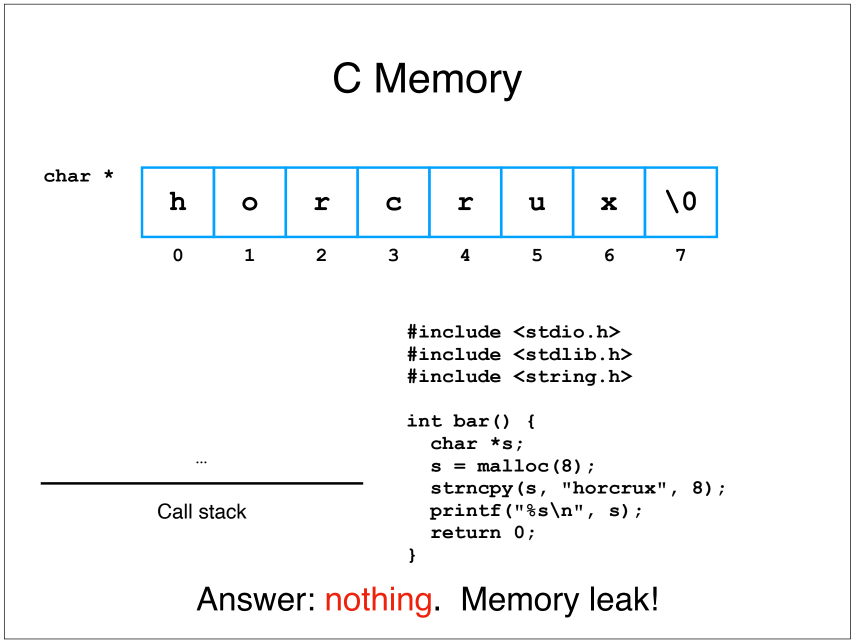
h	o	r	c	r	u	x	\0
0	1	2	3	4	5	6	7

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int bar() {
  char *s;
  s = malloc(8);
  strncpy(s, "horcrux", 7);
  printf("%s\n", s);
  return 0;
}
```



What happens to s when bar returns?

What happens to the thing s pointed to?



C Memory

char *

h	o	r	c	r	u	x	\0
0	1	2	3	4	5	6	7

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int bar() {
  char *s;
  s = malloc(8);
  strncpy(s, "horcrux", 8);
  printf("%s\n", s);
  return 0;
}
```

Answer: nothing. Memory leak!

C Rules

0. Pointers are for **pointing at** other values in **memory**.
1. Remember, when using a variable, you're **always** ask C to **reserve memory** for some **duration**.
2. **Always allocate** and **deallocate** long duration storage.
3. **Always initialize** variables.
4. **Watch out** for **off-by-one** errors.
5. **Always null-terminate** "C strings."

Recap & Next Class

Today we discussed:

More C

Next class:

Virtual memory

Segmentation Faults

Pseudoterminals