Memory Allocation III

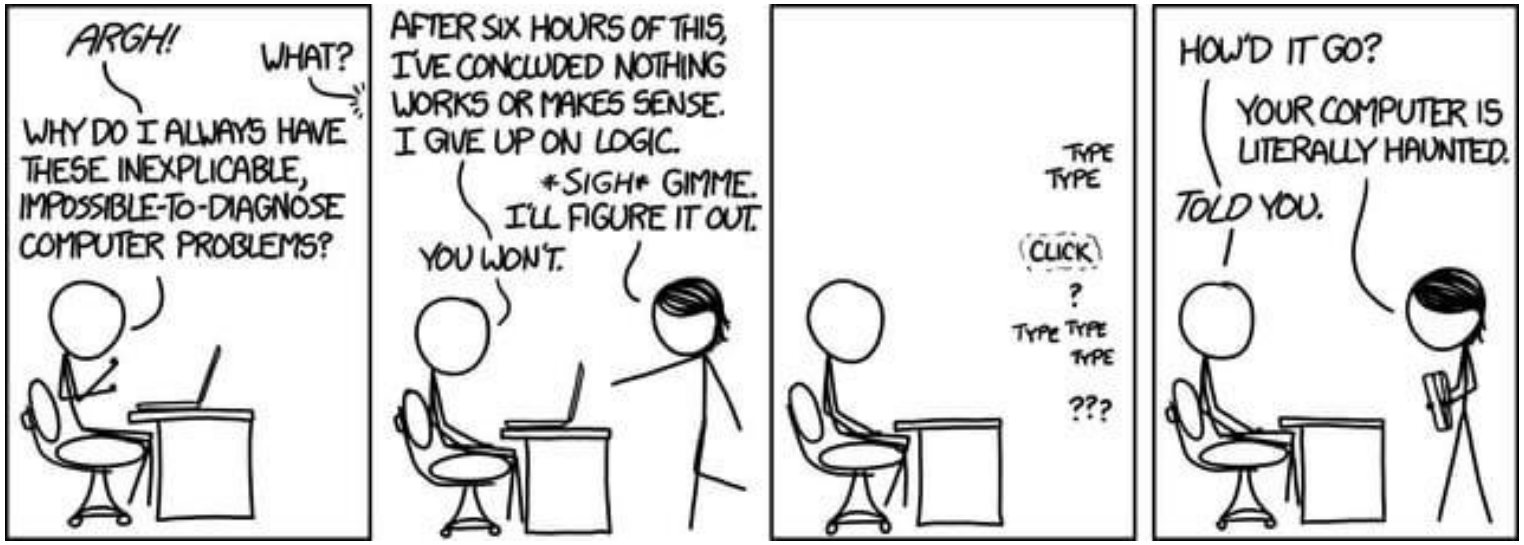
CSE 351 Winter 2024

Guest Lecturer:
Aman Mohammed

Instructor:
Justin Hsia

Teaching Assistants:

Adithi Raghavan	Connie Chen
Malak Zaki	Jiawei Huang
Naama Amiel	Nikolas McNamee
Nathan Khuat	Pedro Amarante
Eyoel Gebre	Will Robertson



Relevant Course Information

- HW19 due tonight
- HW20 due Wednesday (2/28)
- HW21 due Friday (3/1)

- Lab 4 due Friday (3/1)
- Lab 5 due Friday (3/8)
  - Section this week: lab 5 preparation!

- Looking ahead
  - Final March 11 – 13, more info on this to come
  - Check your grades in Canvas as we go

Memory Allocation III

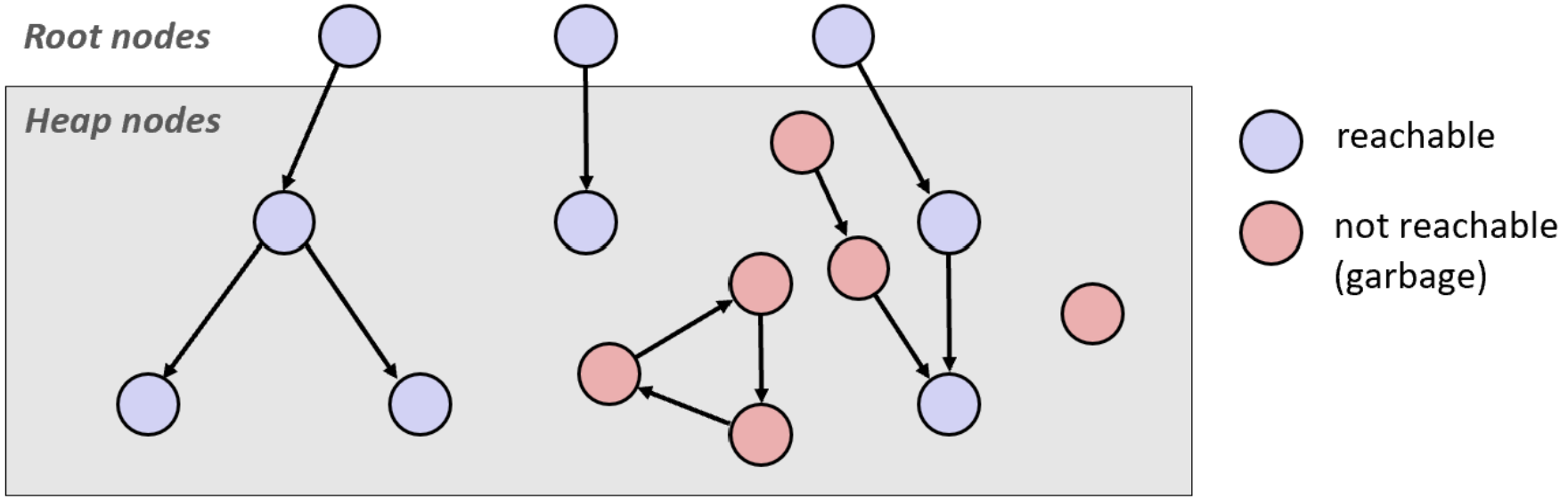
Lesson Summary (1/3)

- **Garbage Collection:** automatically freeing space on the heap when no longer needed
  - Part of an **implicit** memory allocator
  - Free any memory no longer reachable by the program's local variables
  - Runs periodically throughout the lifetime of your program
- Done in many languages (Java, Python, etc.), but not *C!*
  - Why not? – C's flexibility comes at a cost
    - Hard to tell what is a pointer and what isn't (casting)
    - Pointers don't always point to the beginning of blocks (pointer arithmetic)
  - Garbage collectors for C exist, but they don't catch everything
    - Not part of the standard library

Lesson Summary (2/3)

- **Mark-and-Sweep** is a common garbage collection algorithm
  - Stores a **mark bit** for each heap block

1. Start at root nodes (all variables in scope – global variables, stack variables, etc.)
2. Mark all "reachable" heap blocks (from all root nodes)
3. Look through all heap blocks in order, **free any unmarked blocks**



Lesson Summary (3/3)

- Common **malloc-specific** bugs:
  - **Memory leak:** allocating space with malloc, but never freeing it
  - **Double-free:** freeing the same block twice
  - **Accessing a freed block:** using a block after it's been freed
  - **Wrong allocation size:** not allocating enough space for your data

- **Debug smarter, not harder.**
  - Start from the symptoms and work backwards
    - e.g., use backtrace on a segmentation faults
  - GDB is your friend - helpful for lab 5 and beyond

Lesson Q&A

- Terminology:
  - Garbage collection: mark-and-sweep
  - Memory-related issues in C

- Learning Objectives:
  - Explain **the tradeoffs between different allocator implementations**, policies, and strategies.
  - **Identify and debug issues** such as memory leaks, incorrect pointer use, or buffer overflow in C programs.

- What lingering questions do you have from the lesson?

Memory Allocation III – Practice

Find That Bug! (Slide 9)

```
char s[8]; 8 bytes
int i;

gets(s);   /* reads "123456789" from stdin */
```

null terminator → 10 bytes

↳ ~~Use~~ not memory safe

program may stop if you change ret address to non-accessible memory. (seg fault).

Error:

Program stop?

Fix:

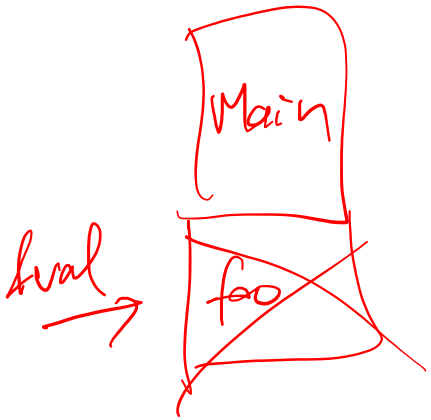
Find That Bug!

```
char s[8];
int i;

gets(s);  /* reads "123456789" from stdin */
```

Error: no bounds checking **Program stop?** Sometimes **Fix:** check bounds with fgets

Find That Bug! (Slide 11)

```
int* foo() {
    int val = 0;

    return &val;
}
```



GCC nullifies returned stack ptr! Crash early. Why is this good?

Error:

Program stop? depends → does the user of this ptr deref? does the compiler nullify the ptr?

Fix:

Find That Bug!

```
int* foo() {
    int val = 0;

    return &val;
}
```

Error: using nonexistent var **Program stop?** Sometimes **Fix:** malloc

Find That Bug! (Slide 13)

```
int** p;

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

for (int i = 0; i < N; i++) {
   p[i] = (int*)malloc( M * sizeof(int) );
}
```

- `N` and `M` defined elsewhere (`#define`)

Error: **Program stop?** **Fix:**

Find That Bug!

```
int** p;

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

for (int i = 0; i < N; i++) {
   p[i] = (int*)malloc( M * sizeof(int) );
}
```

- `N` and `M` defined elsewhere (`#define`)

Error: wrong allocation size **Program stop?** Sometimes **Fix:** malloc

Find That Bug! (Slide 15)

```
x = (int*)malloc( N * sizeof(int) );
    // manipulate x
free(x);

    ...

y = (int*)malloc( M * sizeof(int) );
    // manipulate y
free(x);
```

Error: **Program stop?** **Fix:**

Find That Bug!

```
x = (int*)malloc( N * sizeof(int) );
    // manipulate x
free(x);

    ...

y = (int*)malloc( M * sizeof(int) );
    // manipulate y
free(x);
```

Error: Double-free **Program stop?** Sometimes **Fix:** free(y)

Find That Bug! (Slide 17)

```
x = (int*)malloc( N * sizeof(int) );
   // manipulate x
free(x);

   ...

y = (int*)malloc( M * sizeof(int) );
for (i=0; i<M; i++)
   y[i] = x[i]++;
```

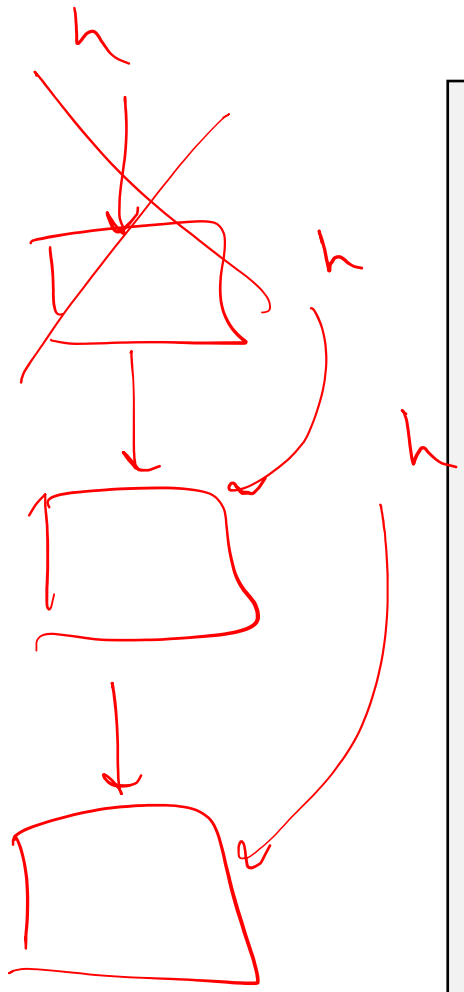
Error: **Program stop?** **Fix:**

Find That Bug!

```
x = (int*)malloc( N * sizeof(int) );
   // manipulate x
free(x);

   ...

y = (int*)malloc( M * sizeof(int) );
for (i=0; i<M; i++)
   y[i] = x[i]++;
```

Error: Access again after free **Program stop?** Sometimes **Fix:** free x after using it

Find That Bug! (Slide 19)



```
typedef struct L {
    int val;
    struct L* next;
} list;

void foo() {
    list* head = (list*) malloc( sizeof(list) );
    head->val = 0;
    head->next = NULL;
        // create and manipulate the rest of the list
        ...
    free(head);
    return;
}
```

Error: **Program stop?** **Fix:**

Find That Bug!

```
typedef struct L {
    int val;
    struct L* next;
} list;

void foo() {
    list* head = (list*) malloc( sizeof(list) );
    head->val = 0;
    head->next = NULL;
        // create and manipulate the rest of the list
        ...
    free(head);
    return;
}
```

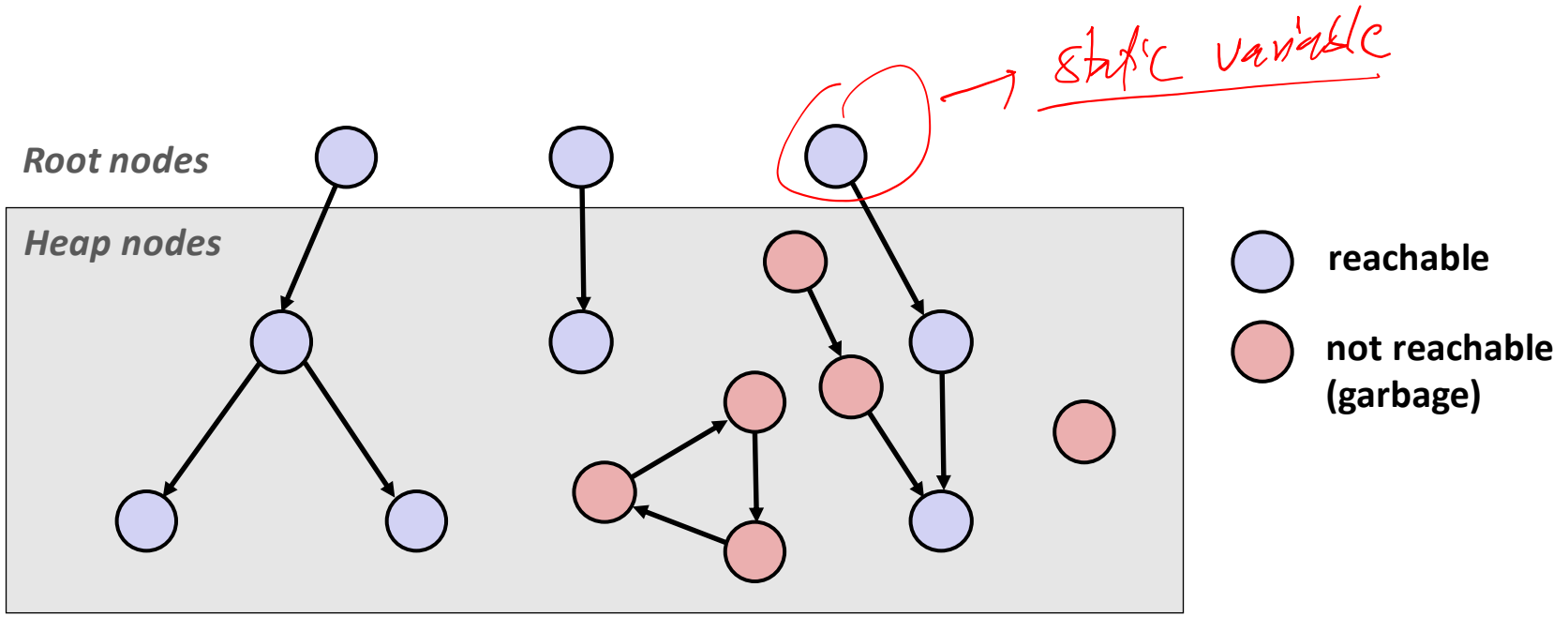
Error: memory leak **Program stop?** No **Fix:** save head-next, then free

What about Java or ML or Python or …?

- In *memory-safe languages*, most of these bugs are impossible
  - Cannot perform arbitrary pointer manipulation
  - Cannot get around the type system
  - Array bounds checking, null pointer checking
  - Automatic memory management
- But one of the bugs we saw earlier is possible. Which one?

Memory Leaks with GC

- Not because of forgotten `free` — we have GC!
- Unneeded "leftover" roots keep objects reachable
- *Sometimes* nullifying a variable is not needed for correctness but is for performance
- Example: Don't leave big data structures you're done with in a static field



Memory Allocation III – Context

Debugging

"As soon as we started programming, we found to our surprise that it wasn't as easy to get programs right as we had thought. Debugging had to be discovered. I can remember the exact instant when I realized that a large part of my life from then on was going to be spent in finding mistakes in my own programs."

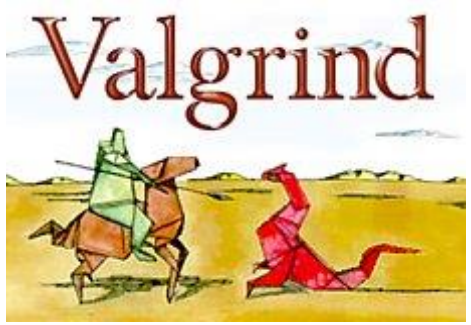
– *Memoirs of a Computer Pioneer*
by Maurice Wilkes

Quick Debugging Note

- Staring at code until you think you spot a bug is generally *not* an effective way to debug!
  - Of course it looks logically correct to you – you wrote it!
  - Language like C doesn't abstract away memory – it's part of your program state that you need to keep track of
    - Your code will only get longer and more complicated in the future: there's too much to try to keep track of mentally
- Instead, start with bad/unexpected behavior to guide your search
  - This is why we like code that crashes early
  - Search bottom-up and not top-down (exhaustive search will take forever)
  - e.g., use backtrace on seg faults as a first step

Dealing With Memory Bugs

- Make use of all of the tools available to you:
  - Pay attention to compiler warnings and errors
  - Use debuggers like GDB to track down runtime errors
    - Good for bad pointer dereferences, bad with other memory bugs
  - `valgrind` is a powerful debugging and analysis utility for Linux, especially good for memory bugs
    - Checks each individual memory reference at *runtime* (*i.e.*, only detects issues with parts of code used in a specific execution)
    - Can catch many memory bugs, including bad pointers, reading uninitialized data, double-frees, and memory leaks



Debugging Strategies

- You've got to find what works best for you
- Try a lot – your debugging technique should grow over time and some techniques will work better for different domains
  - Print debugging
  - Using a debugger
  - Visualizations
  - Generating thorough test cases/suites
  - Including sensible checks throughout your program
  - etc.
- But this isn't what we're here to talk about now…

Supporting Yourself While Debugging

- This is also a learning process!
- Why is this necessary (and difficult)?
  - CS actively encourages prolonged periods of mental concentration
    - Easy to tune everything else out when you remain immobile just a few feet from your screen (and screens are getting bigger)
    - Programmers describe sometimes being "in the zone"
    - Long coding sessions and late nights are socially and culturally encouraged
    - Hackathons are designed this way and also encourage you to ignore your bodily needs
    - Tech companies entice you to stay at work with free food and amenities

Supporting Yourself While Debugging

- **Mindfulness:** "The practice of bringing one's attention in the present moment"
  - Lots of different definitions and nuance, but we'll stick with this broad definition and not the wellness craze

- While debugging, try to be *mindful* of your emotional and physical state as well as your current approach
  - Are you focused on the task at hand or distracted?
  - Am I calm and/or rested enough to be thinking "clearly?"
  - How is my posture, breathing, and tenseness?
  - Do I have any physical needs that I should address?
  - What approach am I trying and why? Are there alternatives?

Supporting Yourself While Debugging

- Try: set a timer for <your interval of choice> (*e.g.*, 15 minutes) to evaluate your state and approach
  - Like the system timer your OS uses for context switching!

- If you're distracted, feeling negative emotions, tense, or need to address something, ***take a break!***
  - You will often find that you'll make a discovery while on a break or at least recover from setbacks
  - Breaks also vary wildly by individual and situation
    - Make sure that you actually feel rested afterward
    - *e.g.*, make tea, work out, do chores, watch a show/movie, play games, chat with friends, make art

Supporting Yourself

- There are few guarantees for support, besides the support that you can give yourself
  - Get comfortable in your own skin and stand up for yourself
  - Can also find support from peers, mentors, family, friends
- Your wellbeing is much more important than your assignment grade, your GPA, your degree, your pride, or whatever else is pushing you to finish *right now*
- Don't attach too much of your self-worth to programming and debugging
  - There's so much more that makes you a wonderful and worthwhile human being!

Discussion Question

- Discuss the following question(s) in groups of 3-4 students
  - I will call on a few groups afterwards so please be prepared to share out
  - Be respectful of others' opinions and experiences

- Reflect on the last time you were debugging something, whether it was for this class or another. What was the issue? How did you go about solving it? Is there something you would try differently?

Group Work Time

- During this time, you are encouraged to work on the following:
  1) If desired, continue your discussion
  2) Work on the homework problems
  3) Work on the current lab

- Resources:
  - You can revisit the lesson material
  - Work together in groups and help each other out
  - Course staff will circle around to provide support