



CS107 Lab 7 – Last One! 🎉



Optimizing, Memory Profiling, Ethics



Parting Thoughts, My CS Journey

Ben Yan, Winter 2026 ❄️

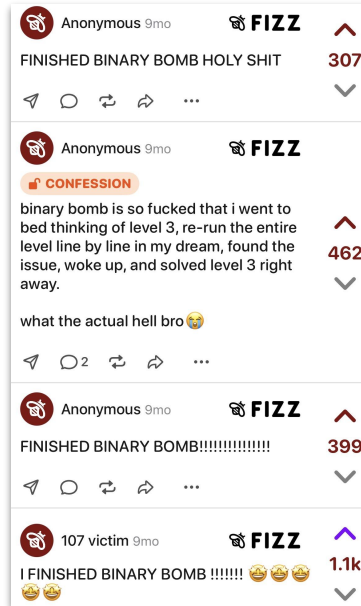
Slides Link: stanford.edu/~bbyan/cs107 :)

Thank you all so much for everything! 😊



Welcome Back to Lab! (and our last one)

And congrats on powering through Binary Bomb!!



Summarize your vault experience in one word :) _____

Welcome Back to Lab!

*Congrats on powering through Binary Bomb!
You're all doing amazing and I'm incredibly inspired*



Winter



Week 9



Spring



- **Assignment 6: Heap Alligator 🐊, the last one!, is out**
 - **Optional checkpoint (implicit allocator) due this Friday**
 - **All of it is due next Friday, March 13th**
- **Like Assign5, please please please allocate time wisely! It's also ~~notorious~~ fun but challenging and rewarding. Start early!**



Heap Allocator Implementation Gem

- ❑ In terms of speed, this is the best heap allocator I've ever seen!

```
void* malloc(size_t size) {  
    return (void*) rand();  
}
```

- ❑ As for trying to free blocks of memory that are randomly flung around:

```
void* free(void* ptr) {  
    // lol \_(ツ)_/  
}
```

Getting Setup for Lab Finale



- ❑ To get started, log into the [myth machines](#), and clone the lab starter code using the command:

```
git clone /afs/ir/class/cs107/repos/lab7/shared lab7
```

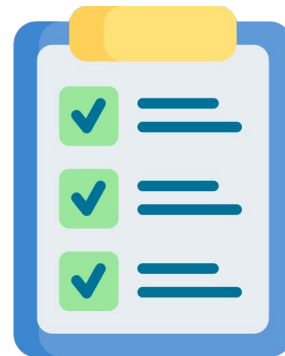
- ❑ Next, also open the **online lab check-off** linked below (or in the lab 7 handout), so you can write as you go.



[Lab Handout](#)



[Lab Check-Off Form](#)



Don't worry at all if you fall behind on the write up! I'll aim to go over everything, though just see me after lab if anything is still left



Tysm Jerry for the opportunity to teach CS!! 💖

Lecture Recap!



~10 minutes



Our last Jerry picture of the week!

Trivia: My disembodied hand is on Jerry's Facebook profile picture!



Lecture Recap: Optimization

Optimization is the task of making your program **faster and/or more efficient** with space or time!



- ❑ **GCC can perform a variety of optimizations for us** when compiling into machine code with different optimization flags 🚩 (e.g., `-O2`)
- ❑ There are a variety of different optimizations GCC does, such as constant folding, strength reduction, loop unrolling, and more (covered in lecture slides)
- ❑ **GCC can't predict or optimize everything** - there are some cases where we must do targeted optimization to improve efficiency and remove bottlenecks

```
./mult          // -O0 (no optimization)
matrix multiply 25^2: cycles 1.32M
matrix multiply 50^2: cycles 10.61M
matrix multiply 100^2: cycles 18.09M
```

```
./mult_opt      // -O2 (with optimization)
matrix multiply 25^2: cycles 0.19M (opt)
matrix multiply 50^2: cycles 2.04M (opt)
matrix multiply 100^2: cycles 13.92M (opt)
```



The Many Species of GCC Optimizations

GCC can perform a variety of optimizations for us, when compiling with different flags (–O0 for none, –O2 for nearly all reasonable optimizations, –O3 is more aggressive)



Constant Folding: Compiler can pre-compute constant values, including e.g., constant arithmetic, `sizeof(int) → 4`, `strlen("AliceInWonderland") → 17`



Common subexpression elimination: Avoids recalculating same result multiple times



Strength reduction: Avoids multiplying/dividing by shifting and adding instead



Dead code elimination: If a piece of code can never be reached (e.g., code in a function after a return statement), the compiler can just remove it



Code motion: Rearrange code for better performance



Loop unrolling: Avoid many expensive conditions and jumps by copy-pasting the loop body (i.e. less iterations, with more work per iteration)



Tail Recursion: Alters some recursive patterns to be iterative instead



The Many Species of GCC Optimizations

GCC can perform a variety of optimizations for us, when compiling with different flags (–O0 for none, –O2 for nearly all reasonable optimizations, –O3 is more aggressive)



Constant Folding: Compiler can pre-compute constant values, including e.g., constant arithmetic, `sizeof(int) → 4`, `strlen("AliceInWonderland") → 17`



Common subexpression elimination: Avoids recalculating same result multiple times



Strength reduction: Avoids multiplying by constants



Dead code elimination: Removes code that never runs

For more details and concrete examples of each GCC optimization, see the **Lecture slides!**



Code generation: Generates code for better performance



Loop unrolling: Avoid many expensive conditions and jumps by copy-pasting the loop body (i.e. less iterations, with more work per iteration)



Tail Recursion: Alters some recursive patterns to be iterative instead



Callgrind! Because we all loved Valgrind so much

Callgrind is a profiling tool: observes an executing program to gather detailed statistics about resource consumption, e.g., time spent, instructions executed, memory usage, to **evaluate program efficiency** 🚄





Callgrind! Because we all loved Valgrind so much

Callgrind is a profiling tool: observes an executing program to gather detailed statistics about resource consumption, e.g., time spent, instructions executed, memory usage, to **evaluate program efficiency** 🚄



Callgrind is part of the valgrind suite of tools, and **useful for our heap allocators** (we want to eventually make them as efficient as possible)



```
bbyan@myth61:~/CS107_TA/assignments/lab7$ valgrind --tool=callgrind --toggle-collect='*power*' ./trials
==3252945== Callgrind, a call-graph generating cache profiler
==3252945== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==3252945== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3252945== Command: ./trials
==3252945==
==3252945== For interactive control, run 'callgrind_control -h'.
==3252945==
==3252945== Events      : Ir
==3252945== Collected : 427
==3252945==
==3252945== I      refs:      427
bbyan@myth61:~/CS107_TA/assignments/lab7$ callgrind_annotate --auto=yes callgrind.out.3252945
```



Results saved to this file!





Check-Off Q: Callgrind vs Callgrind Annotate

Q: `callgrind` and `callgrind_annotate`, what's the difference?



Check-Off Q: Callgrind vs Callgrind Annotate

Q: `callgrind` and `callgrind_annotate`, what's the difference?

Callgrind is a profiler: it **gathers raw statistics** on an executing program—and writes them to a file

Think: A measuring device / instrument



```
bbyan@myth61:~/CS107_TA/assignments/lab7$ valgrind --tool=callgrind --toggle-collect
==3252945== Callgrind, a call-graph generating cache profiler
==3252945== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==3252945== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3252945== Command: ./trials
==3252945==
==3252945== For interactive control, run 'callgrind_control -h'.
==3252945==
==3252945== Events      : Ir
==3252945== Collected : 427
==3252945==
==3252945== I    refs:      427
bbyan@myth61:~/CS107_TA/assignments/lab7$ callgrind_annotate --auto=yes callgrind.ou
```



Check-Off Q: Callgrind vs Callgrind Annotate

Q: `callgrind` and `callgrind_annotate`, what's the difference?

Callgrind is a profiler: it gathers raw statistics on an executing program—and writes them to a file

Think: A measuring device / instrument 

```
bbyan@myth61:~/CS107_TA/assignments/lab7$ valgrind --tool=callgrind --toggle-collect
==3252945== Callgrind, a call-graph generating cache profiler
==3252945== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==3252945== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3252945== Command: ./trials
==3252945==
==3252945== For interactive control, run 'callgrind_control -h'.
==3252945==
==3252945== Events      : Ir
==3252945== Collected : 427
==3252945== I refs:      427
bbyan@myth61:~/CS107_TA/assignments/lab7$ callgrind_annotate --auto=yes callgrind.ou
```

`callgrind_annotate` is an annotator / analyzer:
it separately reads the file or report created by `callgrind`, and **prints useful info, e.g., annotating the code** with # of dynamic instructions per line

Think: Bar charts / data visualization 

```
. // Use the built in library function
1 long two_to_power_A(unsigned int exp) {
7     return pow(2, exp);
126 => /build/glibc-LcI20x/glibc-2.31/math/./w_pow_tem
2 }
.
. // is specialized 2^exp less work than general pow
1 long two_to_power_B(unsigned int exp) {
6     return exp2(exp);
42 => /build/glibc-LcI20x/glibc-2.31/math/./sysdeps/
2 }
```



Check-Off Q: Callgrind Hot Spots

- What is a “hot spot”? (think code eruption)
- How do you identify a **hot spot** in the annotated source code? 🧭
- How does finding a **hot spot** help you optimize your code? 🔧





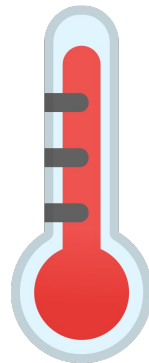
Check-Off Q: Callgrind Hot Spots

- What is a “hot spot”? (think code eruption)
- How do you identify a **hot spot** in the annotated source code? 🧭
- How does finding a **hot spot** help you optimize your code? 🔧



🔥 A hot spot is a line / group of lines in the code that is responsible for a large # of dynamic instructions.

- To spot them, see instruction counts on the annotations.
- Knowing where the hot spots are can help you **focus your optimization efforts** (e.g., by removing / modifying the lines causing the most instructions)





Exercise 1: Profiling with Callgrind

In your **lab7** directory, go through the **trials.c** and **copy.c** programs:

- Run them under callgrind as outlined in the lab handout :)
- Compare the efficiencies (e.g., # of dynamic instructions), see what you find!

endar Handouts ▾ Lectures ▾ Assignments ▾ Labs ▾ Exams ▾ Gradebook Getting Help

Valgrind Callgrind

Written by Nate Hardison and Julie Zelenski, with modifications by Nick Troccoli

Valgrind Callgrind is a program that can profile your code and report on its resources usage. It is another tool provided by Valgrind, which also helps detect [memory issues](#). In fact, the Valgrind framework supports a variety of runtime analysis tools, including [memcheck](#) (detects memory errors/leaks), [massif](#) (reports on heap usage), [helgrind](#) (detects multithreaded race conditions), and [callgrind/cachegrind](#) (profiles CPU/cache performance). This tool guide will introduce you to the Valgrind [callgrind](#) profiling tool.



~ 35 min + 10 min all-lab discussion





2^{exp} Version **A** , Static Instruction Count



C source code

```
long two_to_power_A(unsigned int exp) {  
    return pow(2, exp);  
}
```

Using **built-in library function pow** for general exponents



2^{exp} Version **A** , Static Instruction Count



C source code

```
long two_to_power_A(unsigned int exp) {  
    return pow(2, exp);  
}
```

Using **built-in library function pow** for general exponents



Compiler-generated assembly

```
000000000040118b <two_to_power_A>:  
40118b: 48 83 ec 08      sub    $0x8,%rsp  
40118f: 89 ff            mov    %edi,%edi  
401191: 66 0f ef c9      pxor   %xmm1,%xmm1  
401195: f2 48 0f 2a cf    cvtsi2sd %rdi,%xmm1  
40119a: f2 0f 10 05 de 0e 00 movsd  0xede(%rip),%xmm0  
4011a1: 00  
4011a2: e8 99 fe ff ff    call   401040 <pow@plt>  
4011a7: f2 48 0f 2c c0    cvttsd2si %xmm0,%rax  
4011ac: 48 83 c4 08      add    $0x8,%rsp  
4011b0: c3              ret
```

9 instructions!

But we're not counting how many are in the **pow** library function



2^{exp} Version **B**, Static Instruction Count



C source code

```
long two_to_power_B(unsigned int exp) {  
    return exp2(exp);  
}
```

Using a **specialized function** **exp2 ()**
for powers of 2



2^{exp} Version **B**, Static Instruction Count



C source code

```
long two_to_power_B(unsigned int exp) {  
    return exp2(exp);  
}
```

Using a **specialized function** **exp2 ()** for powers of 2



Compiler-generated assembly

```
0000000004011b1 <two_to_power_B>:  
4011b1: 48 83 ec 08      sub    $0x8,%rsp  
4011b5: 89 ff            mov    %edi,%edi  
4011b7: 66 0f ef c0      pxor   %xmm0,%xmm0  
4011bb: f2 48 0f 2a c7    cvtsi2sd %rdi,%xmm0  
4011c0: e8 6b fe ff ff    call   401030 <exp2@plt>  
4011c5: f2 48 0f 2c c0    cvttsd2si %xmm0,%rax  
4011ca: 48 83 c4 08      add    $0x8,%rsp  
4011ce: c3              ret
```

8 instructions!

But we're not counting how many are in the **exp2** library function



2^{exp} Version **C**, Static Instruction Count



C source code

```
long two_to_power_C(unsigned int exp){  
    long result = 1;  
    for (int i = 1; i <= exp; i++) {  
        result *= 2;  
    }  
    return result;  
}
```

Using **loop to repeatedly multiply by 2**, iterates **exp** number of times



2^{exp} Version **C**, Static Instruction Count



C source code

```
long two_to_power_C(unsigned int exp){  
    long result = 1;  
    for (int i = 1; i <= exp; i++) {  
        result *= 2;  
    }  
    return result;  
}
```

Using **loop** to repeatedly multiply by 2, iterates **exp** number of times



Compiler-generated assembly

```
000000000401166 <two_to_power_C>:  
401166:    b8 01 00 00 00      mov     $0x1,%eax  
40116b:    ba 01 00 00 00      mov     $0x1,%edx  
401170:    eb 06               jmp     401178 <two_to_power_C+0x12>  
401172:    48 01 d2            add     %rdx,%rdx  
401175:    83 c0 01            add     $0x1,%eax  
401178:    39 c7               cmp     %eax,%edi  
40117a:    73 f6               jae     401172 <two_to_power_C+0xc>  
40117c:    48 89 d0            mov     %rdx,%rax  
40117f:    c3                 ret
```

8 instructions but wait...

We have a loop, so this could be misleading / lead to a very different dynamic count

← 2^{exp} Version **D** , Static Instruction Count



C source code

```
long two_to_power_D(unsigned int exp) {  
    return 1L << exp;  
}
```

Since base is known to be 2, **simple bit shift in binary** is perfect!

← 2^{exp} Version **D** , Static Instruction Count



C source code

```
long two_to_power_D(unsigned int exp) {  
    return 1L << exp;  
}
```

Since base is known to be 2, **simple bit shift in binary** is perfect!



Compiler-generated assembly

```
0000000000401180 <two_to_power_D>:  
401180:      b8 01 00 00 00      mov     $0x1,%eax  
401185:      89 f9                mov     %edi,%ecx  
401187:      48 d3 e0             shl     %cl,%rax  
40118a:      c3                  ret
```

4 instructions slay
Seems legit: no loops,
other functions called



To gauge true performance, use Callgrind!

To measure **dynamic instruction counts**: # instructions actually executed in total

```
bbyan@myth61:~/CS107_TA/assignments/lab7$ valgrind --tool=callgrind --toggle-collect='*power*' ./trials
==3252945== Callgrind, a call-graph generating cache profiler
==3252945== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==3252945== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3252945== Command: ./trials
==3252945==
==3252945== For interactive control, run 'callgrind_control -h'.
==3252945==
==3252945== Events      : Ir
==3252945== Collected : 427
==3252945==
==3252945== I    refs:      427
```



To gauge true performance, use Callgrind!

To measure **dynamic instruction counts**: # instructions actually executed in total



Program being executed (`./trials`),
and `--toggle-collect` to examine all
functions with **power** in the name

```
bbyan@myth61:~/CS107_TA/assignments/lab7$ valgrind --tool=callgrind --toggle-collect='*power*' ./trials
==3252945== Callgrind, a call-graph generating cache profiler
==3252945== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==3252945== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3252945== Command: ./trials
==3252945==
==3252945== For interactive control, run 'callgrind_control -h'.
==3252945==
==3252945== Events      : Ir
==3252945== Collected : 427
==3252945==
==3252945== I    refs:      427
```



To gauge true performance, use Callgrind!

To measure **dynamic instruction counts**: # instructions actually executed in total



Program being executed (`./trials`),
and `--toggle-collect` to examine all
functions with **power** in the name

```
bbyan@myth61:~/CS107_TA/assignments/lab7$ valgrind --tool=callgrind --toggle-collect='*power*' ./trials
==3252945== Callgrind, a call-graph generating cache profiler
==3252945== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==3252945== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3252945== Command: ./trials
==3252945==
==3252945== For interactive control, run 'callgrind_control -h'.
==3252945==
==3252945== Events      : Ir
==3252945== Collected : 427
==3252945==
==3252945== I    refs:    427
```



Total count of instructions executed when running program!



To gauge true performance, use Callgrind!

To measure **dynamic instruction counts**: # instructions actually executed in total



Callgrind writes detailed info to a file named **callgrind.out. [pid]**

➤ Here, **[pid]** is **3252945**



Program being executed (**./trials**), and **--toggle-collect** to examine all functions with **power** in the name

```
bbyan@myth61:~/CS107_TA/assignments/lab7$ valgrind --tool=callgrind --toggle-collect='*power*' ./trials
==3252945== Callgrind, a call-graph generating cache profiler
==3252945== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==3252945== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3252945== Command: ./trials
==3252945==
==3252945== For interactive control, run 'callgrind_control -h'.
==3252945==
==3252945== Events      : Ir
==3252945== Collected : 427
==3252945==
==3252945== I    refs:      427
```



Total count of instructions executed when running program!



To gauge true performance, use Callgrind!

To measure **dynamic instruction counts**: # instructions actually executed in total



Callgrind writes detailed info to a file named **callgrind.out. [pid]**

➤ Here, **[pid]** is **3252945**



Program being executed (**./trials**), and **--toggle-collect** to examine all functions with **power** in the name

```
bbyan@myth61:~/CS107_TA/assignments/lab7$ valgrind --tool=callgrind --toggle-collect='*power*' ./trials
==3252945== Callgrind, a call-graph generating cache profiler
==3252945== Copyright (C) 2002-2017, and GNU GPL'd, by Josef Weidendorfer et al.
==3252945== Using Valgrind-3.15.0 and LibVEX; rerun with -h for copyright info
==3252945== Command: ./trials
==3252945==
==3252945== For interactive control, run 'callgrind_control -h'.
==3252945==
==3252945== Events      : Ir
==3252945== Collected : 427
==3252945==
==3252945== I    refs:      427
```



Let's go take a look inside that **.out** file that callgrind just wrote to!



Total count of instructions executed when running program!



Read Callgrind file with callgrind_annotate

```
bbyan@myth61:~/CS107_TA/assignments/lab7$ callgrind_annotate --auto=yes callgrind.out.3252945
```

```
-----  
Profile data file 'callgrind.out.3252945' (creator: callgrind-3.15.0)  
-----
```



Read Callgrind file with callgrind_annotate



We use `callgrind_annotate` to print out key information from the output file `callgrind.out.[pid]`, where `[pid]` is `3252945` here.



```
bbyan@myth61:~/CS107_TA/assignments/lab7$ callgrind_annotate --auto=yes callgrind.out.3252945
```

```
-----  
Profile data file 'callgrind.out.3252945' (creator: callgrind-3.15.0)  
-----
```



Read Callgrind file with callgrind_annotate



We use `callgrind_annotate` to print out key information from the output file `callgrind.out.[pid]`, where `[pid]` is `3252945` here.

```
bbyan@myth61:~/CS107_TA/assignments/lab7$ callgrind_annotate --auto=yes callgrind.out.3252945
```

```
-----  
Profile data file 'callgrind.out.3252945' (creator: callgrind-3.15.0)  
-----
```

Sample Annotated Printout

```
.           // Use the built in library  
1 ( 0.26%)  long two_to_power_A(unsigned  
7 ( 1.81%)          return pow(2, exp);  
130 (33.68%) => ./math/./w_pow_template  
2 ( 0.52%)  }
```



Read Callgrind file with callgrind_annotate



We use `callgrind_annotate` to print out key information from the output file `callgrind.out.[pid]`, where `[pid]` is `3252945` here.

```
bbyan@myth61:~/CS107_TA/assignments/lab7$ callgrind_annotate --auto=yes callgrind.out.3252945
```

```
-----  
Profile data file 'callgrind.out.3252945' (creator: callgrind-3.15.0)  
-----
```

Note: The `=>` mark separates the:

- Count of instructions for **setting up + calling a function** (here, 7)
- Count of instructions for **executing the function itself** (here, 130)

Sample Annotated Printout

.		// Use the built in library
1 (0.26%)		long two_to_power_A(unsigned
7 (1.81%)		return pow(2, exp);
130 (33.68%)	=>	./math/./w_pow_template
2 (0.52%)		}



Annotator: 2^{exp} Dynamic Instruction Counts



Callgrind annotation

```
. // Use the built in library functi
1 ( 0.26%) long two_to_power_A(unsigned int e
7 ( 1.81%)     return pow(2, exp);
130 (33.68%) => ./math/./w_pow_template.c:pow@@
2 ( 0.52%) }

.
. // is specialized 2^exp less work
1 ( 0.26%) long two_to_power_B(unsigned int e
6 ( 1.55%)     return exp2(exp);
42 (10.88%) => ./math/./sysdeps/ieee754/dbl-6
2 ( 0.52%) }

.
. // loop to repeatedly multiply by
. long two_to_power_C(unsigned int e
1 ( 0.26%)     long result = 1;
142 (36.79%)     for (int i = 1; i <= exp; i++)
46 (11.92%)         result *= 2;
.         }
.         return result;
2 ( 0.52%) }

.
. // base is known to be 2, simple b
. long two_to_power_D(unsigned int e
3 ( 0.78%)     return 1L << exp;
1 ( 0.26%) }
```



Annotator: 2^{exp} Dynamic Instruction Counts

A 140 instructions

 **Callgrind annotation**

```
. // Use the built in library functi
1 ( 0.26%) long two_to_power_A(unsigned int e
7 ( 1.81%)     return pow(2, exp);
130 (33.68%) => ./math/./w_pow_template.c:pow@@
2 ( 0.52%) }
```

B 51 instructions

```
. // is specialized 2^exp less work
1 ( 0.26%) long two_to_power_B(unsigned int e
6 ( 1.55%)     return exp2(exp);
42 (10.88%) => ./math/./sysdeps/ieee754/dbl-6
2 ( 0.52%) }
```

C 191 instructions

```
. // loop to repeatedly multiply by
. long two_to_power_C(unsigned int e
1 ( 0.26%)     long result = 1;
142 (36.79%)     for (int i = 1; i <= exp; i++)
46 (11.92%)         result *= 2;
.         }
.         return result;
2 ( 0.52%) }
```

D 4 instructions ★

```
. // base is known to be 2, simple b
. long two_to_power_D(unsigned int e
3 ( 0.78%)     return 1L << exp;
1 ( 0.26%) }
```



Callgrind Annotator: 2^{exp} Callgrind Hotspots

A 140 instructions

B 51 instructions

C 191 instructions

D 4 instructions ★



Callgrind annotation

```
. // Use the built in library functi
1 ( 0.26%) long two_to_power_A(unsigned int e
7 ( 1.81%)     return pow(2  exp);
130 (33.68%) => ./math/./w_pow_template.c:pow@@
2 ( 0.52%) }
```

```
.
. // is s
1 ( 0.26%) long tw
6 ( 1.55%)     ret
42 (10.88%) => ./ma
2 ( 0.52%) }
```

```
.
. // loop to repeatedly multiply by
. long two_to_power_C(unsigned int e
1 ( 0.26%)     long result = 1;
142 (36.79%)     for (int i = 1; i <= exp; i++)
46 (11.92%)         result *= 2;
.
.     }
.     return
2 ( 0.52%) }
```

```
.
. // base is
. long two_to_power_D(unsigned int e
3 ( 0.78%)     return 1L << exp;
1 ( 0.26%) }
```

🔥 Also hot spot due to use of expensive, general pow () library function

🔥🔥 Largest hot spot due to expensive loop



Check-Off Q: Memory Copying

About **how much faster** is a 📦 block memcpy copy versus a loop copying chunk by chunk using memcpy? (i.e. compare # of dynamic instructions)



Check-Off Q: Memory Copying

About **how much faster** is a  block memcpy copy versus a loop copying chunk by chunk using memcpy? (i.e. compare # of dynamic instructions)

```
7 ( 0.07%) void *copy_memloop(void *dst, const void *src,
2 ( 0.02%)     size_t n_iterations = nbytes / MEMLOOP_CHUNK,
.
.
.           // Copy one chunk at a time
388 ( 3.82%)    for (long i = 0; i < n_iterations; i++) {
640 ( 6.30%)        memcpy((char *)dst + i * MEMLOOP_CHUNK,
1,664 (16.38%) => ./string/../sysdeps/x86_64/multiarch/memmove.c
128 ( 1.26%)        (char *)src + i * MEMLOOP_CHUNK,
.
.           }
.           return dst;
7 ( 0.07%) }
```

```
1 ( 0.01%) void *copy_memcpy(void *dst, const void *src,
2 ( 0.02%)     return memcpy(dst, src, nbytes);
125 ( 1.23%) => ./string/../sysdeps/x86_64/multiarch/memmove.c
2 ( 0.02%) }
```



Check-Off Q: Memory Copying

About **how much faster** is a  block memcpy copy versus a loop copying chunk by chunk using memcpy? (i.e. compare # of dynamic instructions)

Solution: Block memcpy is an order of magnitude faster,

➤ **copy_memloop** is **2836** instructions

➤ **copy_memcpy** is **130** instructions

~ 22 ish times faster

```
7 ( 0.07%) void *copy_memloop(void *dst, const void *src,
2 ( 0.02%)     size_t n_iterations = nbytes / MEMLOOP_CHUNK
.
.
. // Copy one chunk at a time
388 ( 3.82%) for (long i = 0; i < n_iterations; i++) {
640 ( 6.30%)     memcpy((char *)dst + i * MEMLOOP_CHUNK,
1,664 (16.38%) => ./string/./sysdeps/x86_64/multiarch/memmove
128 ( 1.26%)     (char *)src + i * MEMLOOP_CHUNK,
.
.     }
.     return dst;
7 ( 0.07%) }
```

```
1 ( 0.01%) void *copy_memcpy(void *dst, const void *src,
2 ( 0.02%)     return memcpy(dst, src, nbytes);
125 ( 1.23%) => ./string/./sysdeps/x86_64/multiarch/memmove
2 ( 0.02%) }
```

Happy to answer any questions!

callgrind, valgrind, the Stanford grind





Exercise 2: Ethics Case Study

In lab today, we'll consider a **case study of one of the worst software bugs in history**: a lethal race condition in the software of a medical radiation device, the Therac-25, between 1985-1987.

Go ahead and read through the article here [[link](#)], discuss the article + lab questions in groups of 4-6, and we'll reconvene for a lab-wide discussion.



~ 10 min + 10 min all-lab discussion



PATIENT NAME: JOHN	BEAM TYPE: X	ENERGY (KeV):	10
TREATMENT MODE: FIX			
	ACTUAL	PRESCRIBED	
UNIT RATE/MINUTE	0.000000	0.000000	
MONITOR UNITS	200.000000	200.000000	
TIME (MIN)	0.270000	0.270000	
GANTRY ROTATION (DEG)	0.000000	0.000000	VERIFIED
COLLIMATOR ROTATION (DEG)	359.200000	359.200000	VERIFIED
COLLIMATOR X (CM)	14.200000	14.200000	VERIFIED
COLLIMATOR Y (CM)	27.200000	27.200000	VERIFIED
WEDGE NUMBER	1.000000	1.000000	VERIFIED
ACCESSORY NUMBER	0.000000	0.000000	VERIFIED
DATE: 2012-04-16	SYSTEM: BEAM READY	OP.MODE: TREAT	AUTO
TIME: 11:49:58	TREAT: TREAT PAUSE	X-RAY	179777
OP# ID: 033-1100p	REASON: OPERATOR	COMMAND: █	



Exercise 2: Ethics Case Study

Responsibility: Who should be held responsible for incidents such as the Therac-25 incident? In what way should they be held responsible?



Exercise 2: Ethics Case Study

Responsibility: Who should be held responsible for incidents such as the Therac-25 incident? In what way should they be held responsible?

Many possible answers:

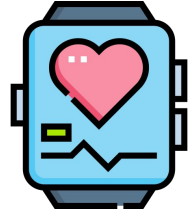
- **Perhaps company** (e.g., the corporation itself, leadership, project managers) – lawsuits, monetary damages, government penalties / inquiries
- **Perhaps shareholders** – though maybe they're not aware of the software flaws?
- **Perhaps software engineers** – though they implemented the buggy code, maybe one person alone didn't know about the scope of the issue?
- **Perhaps quality testers** – would have to probe what testing they did / didn't do
- **Perhaps the hospitals using it** – maybe safe-use protocols weren't followed?

Overarching Discussion: Is it more collective or team responsibility (for instituting proper protocols, review procedures) than individual responsibility?



Exercise 2: Ethics Case Study

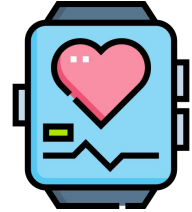
Regulation: The FDA now regulates medical devices more thoroughly than they did in the 1980s. However, most uses of software are not regulated. In addition to medical software, **what other kinds of software (if any) ought to be regulated** in order to preserve human life, health, and safety?





Exercise 2: Ethics Case Study

Regulation: The FDA now regulates medical devices more thoroughly than they did in the 1980s. However, most uses of software are not regulated. In addition to medical software, **what other kinds of software (if any) ought to be regulated** in order to preserve human life, health, and safety?



Also many possible answers:

- Stock trading or financial software? ~~e.g., SecureVault~~
- Software inside self-driving cars or just regular cars?
- Navigation apps?
- Military hardware and software?
- Health advice or personal health data apps?
- Language models, chatbots, image-generation AI?



Can be many other answers! Our main goal in lab is to facilitate a discussion and encourage thinking about the breadth of possibilities.

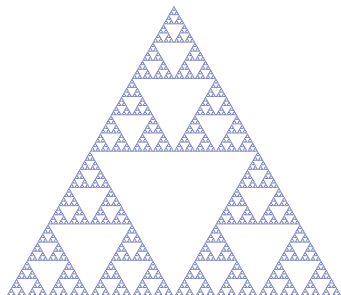


Some last bits before goodbye
👋 (for now at least!)

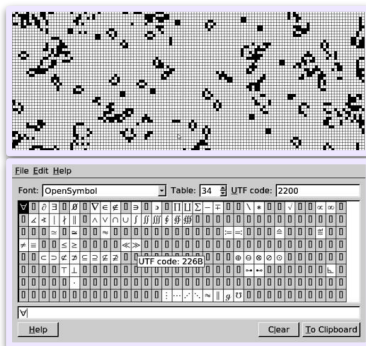




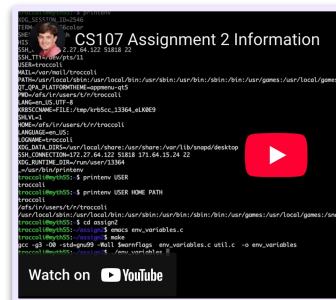
Congrats on all the beautiful work you've done!



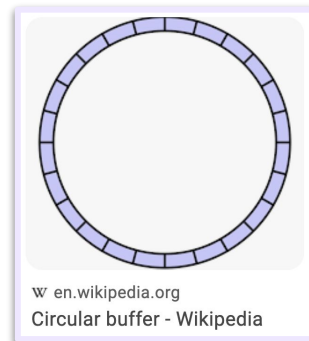
A0: Intro to Unix & C



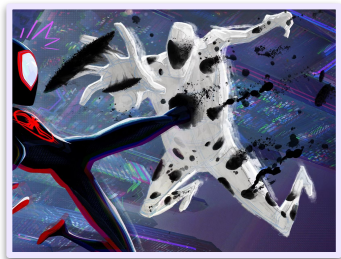
A1: A Bit of Fun



A2: C Strings



A3: A Heap of Fun



A4: Into the Void*

```
myth$ ./samples/atm 50 $USER
Please take your cash:
$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
[Transaction Log: Presented credential as
withdrew $50, new balance = $57]

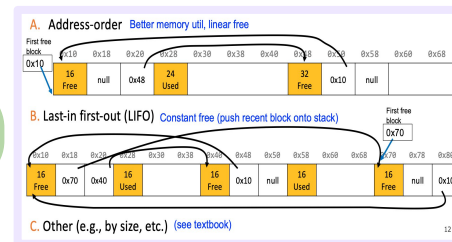
myth$ ./samples/atm 100 $USER
Request failed: insufficient funds.

myth$ ./samples/atm 50 someoneelse
Request failed: unauthorized.
```



A5: SecureVault!

And soon!



A6: Heap Allocator!!



Announcement: Course + Section Evals

They release at the start of Week 10 – I'd really appreciate and value your honest, anonymous feedback there when you have time! ❤️

Your evaluations help the course, Jerry, my fellow TAs and me immensely.

- There's a **Course** tab to give feedback for Jerry / anyone or aspect of the course! Plus any comments and pointers :) for a student taking the course in the future.
- There's a **Section** tab to give TA or lab leader specific feedback!





Recommended Read: Kate's CS/CS107 Story

Sharing this with permission from our wonder CA Kate Baker yay!



My CS Story: How I learned my Path Wasn't Set

I used to think Stanford's computer science major and its classes were untouchable. My high school's coding classes seemed to be filled with those who had already built apps, animated short films, or were just general geniuses (or at least were filled with those more comfortable boasting about it). Also, it felt like my high school friends and I were prematurely siloed into distinct disciplines based on which subjects we excelled in. It felt too late to explore anything else, particularly the black box that was coding. And that was just my tiny high school! I assumed that Stanford must have summoned the programming elites – surely the university wouldn't have time to teach a lousy freshman from scratch whose computer acumen was limited to naming a Google doc. Don't worry though – that's certainly not the case.

Then the midterm... I have never received a lower score in my life – I was humiliated. The cloud that previously enshrouded the computer science discipline returned and I couldn't see any other way out of the fog besides dropping the class. I knew that I couldn't succeed in this field – I had only just started! I began to spiral, but after discussing my dilemma with some friends, they recommended I talk with my professor to get some advice on how to improve my studying and to get a feel of my place in the class. It was an amazing decision and wonderful advice; he was willing to meet one-on-one (even in a 300+ person class) and was incredibly reassuring. He noted that assignments contribute most to the grade anyways and said that I've been succeeding in those, participating in lectures, etc, so I should be okay. My hope returned, and I continued to toil away (but a very positive, gratifying toiling) on the remaining assignments. I tried out some new study strategies and ended up doing much better on the final, securing a good grade in the class.

Both 106B and 107 taught me that Stanford provides every student, no matter their experience, an opportunity to experiment with and explore computer science, or any other major. The professors in this field are typically incredibly accommodating and are rooting for your success. I learned to never be afraid to reach out to a professor for a check-in, no matter the size of the class. And most importantly, I discovered the value in embracing the discomfort of not settling into a major too early if something else piques my interest. I'm now taking classes in both materials science and computer science, and considering switching to a computer science major (or potentially a coterm)!



Full Stanford article on the lab website, and at:

<https://advising.stanford.edu/current-students/my-stanford-story/path-not-set>

Ben's CS Story

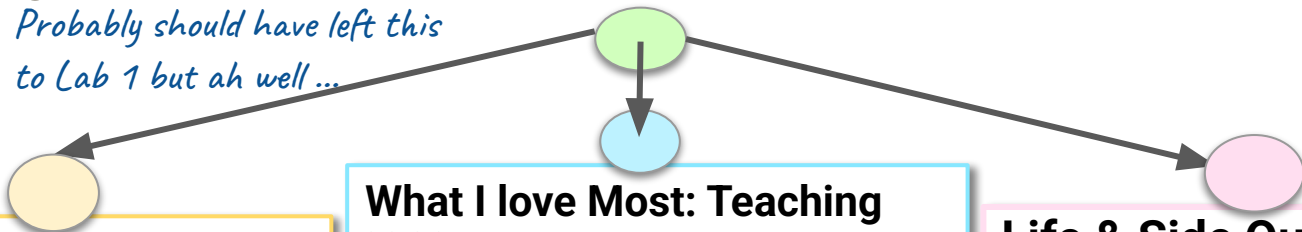
The good, the bad, the very bad, the very very good 

(featuring CS107! ~~as the villain~~)



Story as a heap – a beautiful, fragmented blessed mess

*Probably should have left this
to Lab 1 but ah well...*



Stanford (🎓'24→'26/'27)

- ♣ Majored in **CS (AI Track)**
- ♣ Also majored in **Math** cuz I'm indecisive af 🧰😞
- ♣ Also minored in **Creative Writing** (12 English classes 🤪) as **poetry >> assembly**

Summary: fun, but i got cooked

- ♠ Now CS MS (Theory Track)
- ♠ **My advisor is Jerry!** which has done wonders for my life

What I love Most: Teaching

- 💔💔 Rejected first 2 quarters I applied to be a graduate CS TA :(
- ❤️ 3rd time's a charm! **Head TA for CS106AX with Jerry** in autumn 🎉
- ❤️ Then **TAed for CS107**, love it! Also section-leading for **CS106A/B**
- 💖 Side gig: Lecturer for a 1-unit wonder **CS 106S: Coding for Social Good***, ~110 students overall 💖

Work Stuff

- ♠ Systems SWE (NVIDIA); CURIS after rejections 1st & 2nd year

Life & Side Quests

- ♦ Lived in Lantana, Kimball, Roble, EVGR, **hope SF is next!**
- ♦ **Oxford study abroad (Creative Writing Tutorial 📖)**, got role of Ben(volio) in *Romeo & Juliet* play
- ♦ My fiction writer arc ended in disgrace sadly: **my last story was infamously a PowerPoint :D**
- ♦ **Language learning** 🌐 (can read manga in Japanese, fighting for my life learning Korean rn)

*Last spring 🍀, **CS106S had a record ~40 students**, and somehow, from the course evaluations, it was the **highest-rated course in the entire Stanford CS department! (4.96/5) My students are really too kind** ❤️



The truth I guess ...



time for a lowlight
reel / noteworthy fails



```
bbyan@noMoreMyths: gdb my_time_in_CS
```

```
(gdb) disassemble frosh_year (i.e. Level 1)
```

- My first CS class (frosh autumn 🍂) was CS109, which looking back was kinda sus – prereqs are CS106B, Math 51, and CS103 (soft)
- Nevertheless, Profs Jerry Cain & Lisa Yan made it very welcoming and it felt tough but doable. **But that wasn't the only CS class I took that quarter** 😭.



BEN WHAT

```
(gdb) layout Ben's foolishness
```

- Have you ever heard of a class called CS229 🙌: **MACHINE LEARNING?** 🙌 I wish I haven't. **I got cooked.**
- Goodness, yeah, that still registers as one of the dumbest and most notorious mistakes I've made here
- It was brutal. I'm an idiot. But surely I learned my lesson right —



(gdb) `disassemble ben_in_cs107`

- My frosh year (mostly Zoom), I didn't realize there was a front page on the CS program sheet – **and ended up taking CS107 before I took 106B.** Wtf Ben
 - ➔ **How did that go?** Uhhhh
- Somehow, I was paddling above water (barely), until the bottom fell out on **Assignment 4: Into the Void**

There's More!



Assignments Submitted		Days Late	Functionality Review	
assign4: Into the void*	Sat Feb 20 23:54	3	25/92	permitted
assign5: Some Assembly Required	Sun Mar 07 23:59	2	41/84	

And I collapsed even further in Binary Bomb and Heap Alligator :(🦎

Overall Assignment percentage: 68.41%

broke down crying in OH over heap :(

- **At the end of that winter quarter, I passed just half of my units, which was quite demoralizing ngl.** A lot of time was spent trying to mount a desperate comeback in CS107, and I would have failed if not for ... ironically my perfect lab attendance 🐼

More!



(gdb) next next next next

- The first major I declared was **Physics** (believe it or not), which was a spectacular disaster, again due to Ben doing Ben things.
- Still **surprised I got to the finish line** of my undergrad, definitely didn't seem that way for like a solid four years.

It's not flattering, but it's true that **I took an entire year off from CS in undergrad, taking no classes / mentally giving up**. It was probably the most hopeless and depressed I've felt in a lot of ways (besides CS), and I know **I'm not alone in struggling with mental health here**.



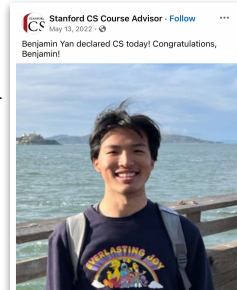
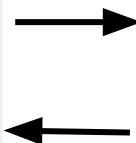
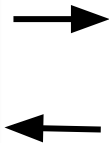
*kindness during a
very rough time*

`trigger_alarm` reached, Ben exiting CS program 🙄

- **Aight, that's most of it lol (for now)**, not trying to elevate academic self-sabotage into an art form :(

Thanks, and let me know if I can help you make some progress today or tomorrow. I don't want you feeling like everyone around you has moved on and that you don't have access to help.

Jerry



Looking on the bright side!

And continuing my final bad metaphor of the quarter

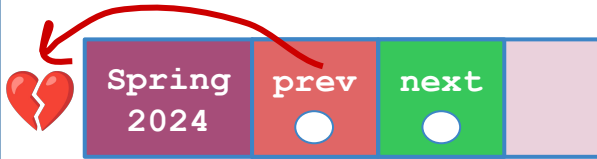


Hope as a heap allocator: linked list of memories



Hope As a Heap Allocator: Linked List of Memories

> Returned from year off as an emotional ruin, met with Jerry
> When **he proposed teaching as an option (for who, me??), I laughed at him**, which I regret.

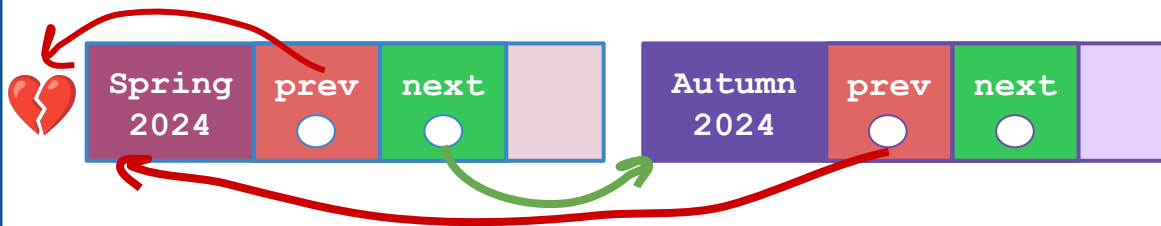




Hope As a Heap Allocator: Linked List of Memories

> Returned from year off as an emotional ruin, met with Jerry
> When **he proposed teaching as an option (for who, me??), I laughed at him**, which I regret.

> Jerry “**accidentally**” hires me **as Head TA of CS106AX** – prob the biggest surprise of my life ✨
> Learned so much, **miraculously revived the hopes I had** in CS



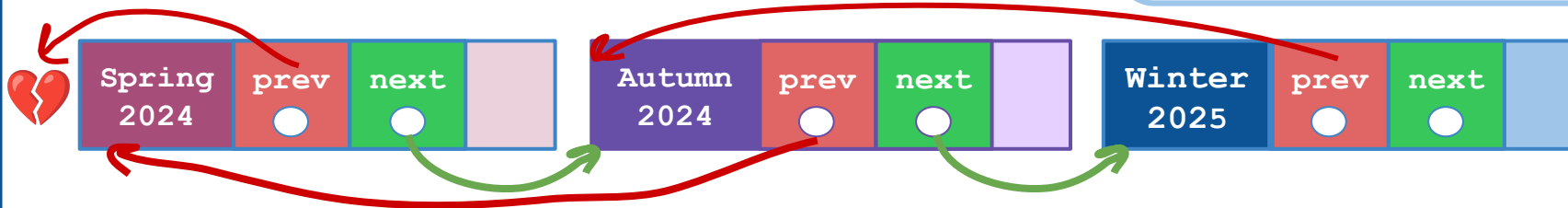


Hope As a Heap Allocator: Linked List of Memories

> Returned from year off as an emotional ruin, met with Jerry
> When **he proposed teaching as an option (for who, me??)**, I **laughed at him**, which I regret.

> **Jerry “accidentally” hires me as Head TA of CS106AX** – prob the biggest surprise of my life ✨
> Learned so much, **miraculously revived the hopes I had** in CS

> Emotional conversation with Jerry, **asking if I could TA CS107, despite barely passing the course**
> **omg a chance at redemption** 🙏
> relearned, redid all assignments!



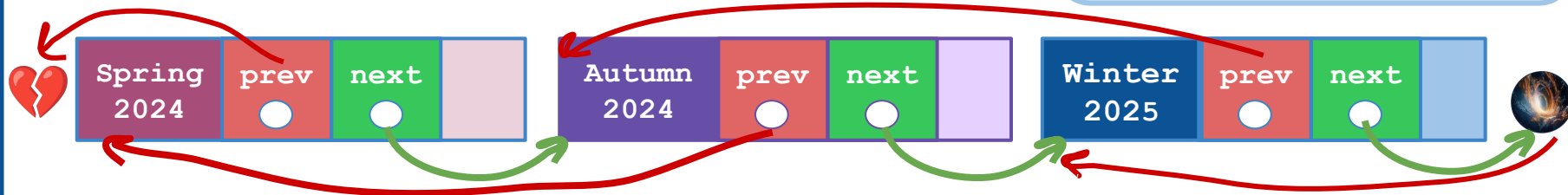


Hope As a Heap Allocator: Linked List of Memories

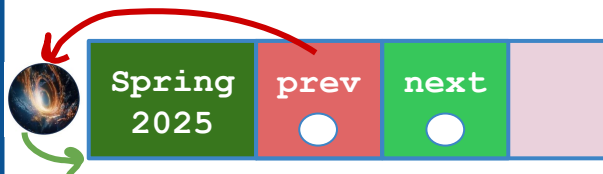
> Returned from year off as an emotional ruin, met with Jerry
> When **he proposed teaching as an option (for who, me??)**, I **laughed at him**, which I regret.

> Jerry “**accidentally**” hires me as **Head TA of CS106AX** – prob the biggest surprise of my life ✨
> Learned so much, **miraculously revived the hopes** I had in CS

> Emotional conversation with Jerry, **asking if I could TA CS107, despite barely passing the course**
> **omg a chance at redemption** 🙏
> relearned, redid all assignments!



> Gifted a 2nd time TAing 107, and **yassified my lab slides** 🌈
> Also lectured CS106S, **had 70 students in total that spring** ❤️



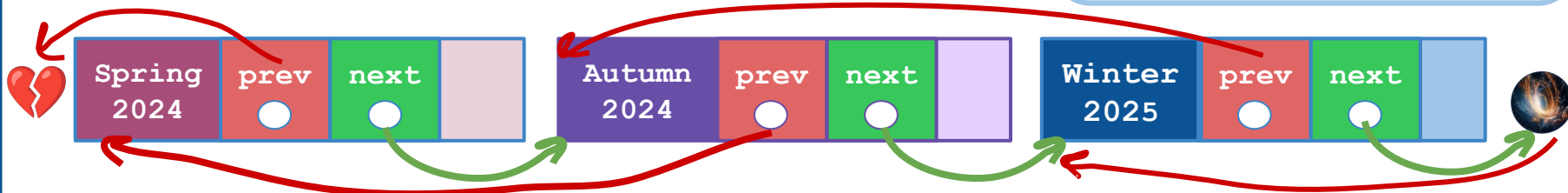


Hope As a Heap Allocator: Linked List of Memories

> Returned from year off as an emotional ruin, met with Jerry
> When **he proposed teaching as an option (for who, me??), I laughed at him**, which I regret.

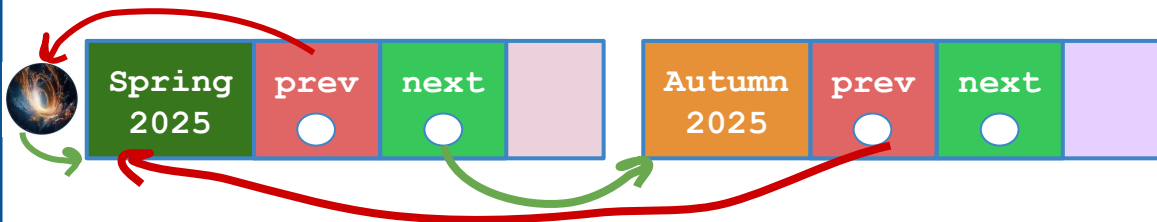
> **Jerry “accidentally” hires me as Head TA of CS106AX** – prob the biggest surprise of my life ✨
> Learned so much, **miraculously revived the hopes I had** in CS

> Emotional conversation with Jerry, **asking if I could TA CS107, despite barely passing the course**
> **omg a chance at redemption** 🙏
> relearned, redid all assignments!



> Gifted a 2nd time TAing 107, and **yassified my lab slides** 🌈
> Also lectured CS106S, **had 70 students in total that spring** ❤️

> Ran it back with Jerry's 106AX, **gave my first CS guest lecture**
> One of the amazing SLs (Diego) I managed is rn my SL for 111 🌌



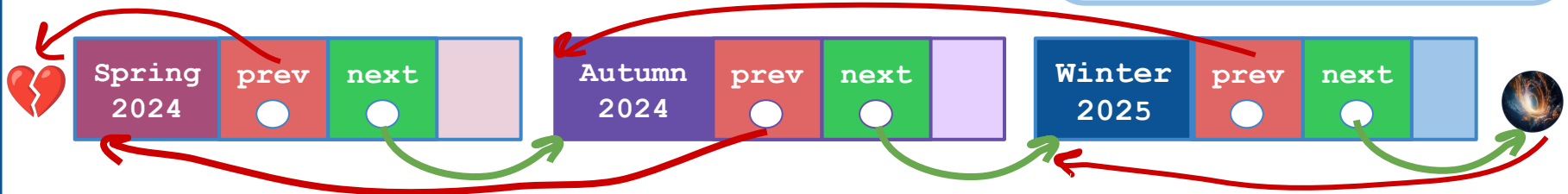


Hope As a Heap Allocator: Linked List of Memories

> Returned from year off as an emotional ruin, met with Jerry
> When **he proposed teaching as an option (for who, me??), I laughed at him**, which I regret.

> **Jerry “accidentally” hires me as Head TA of CS106AX** – prob the biggest surprise of my life ✨
> Learned so much, **miraculously revived the hopes I had** in CS

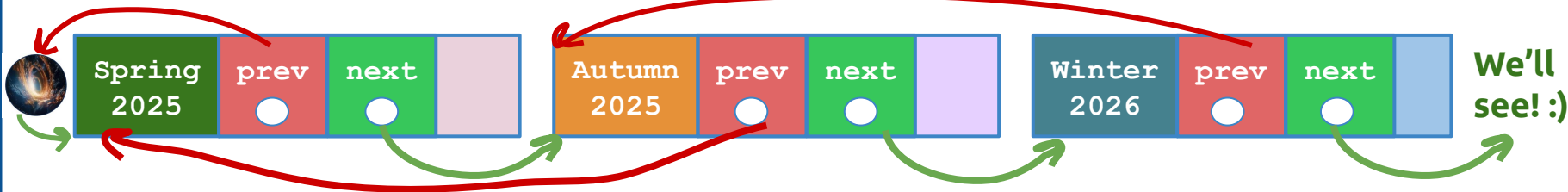
> Emotional conversation with Jerry, **asking if I could TA CS107, despite barely passing the course**
> **omg a chance at redemption** 🙏
> relearned, redid all assignments!



> Gifted a 2nd time TAing 107, and **yassified my lab slides** 🌈
> Also lectured CS106S, **had 70 students in total that spring** ❤️

> Ran it back with Jerry's 106AX, **gave my first CS guest lecture**
> One of the amazing SLs (Diego) I managed is rn my SL for 111 🌌

> **Returned to TAing CS107** 💕💕
> My disaster bi **double life starts with also section-leading 106A :D**
> My stellar students are why I teach





my ongoing “situationship with systems” arc

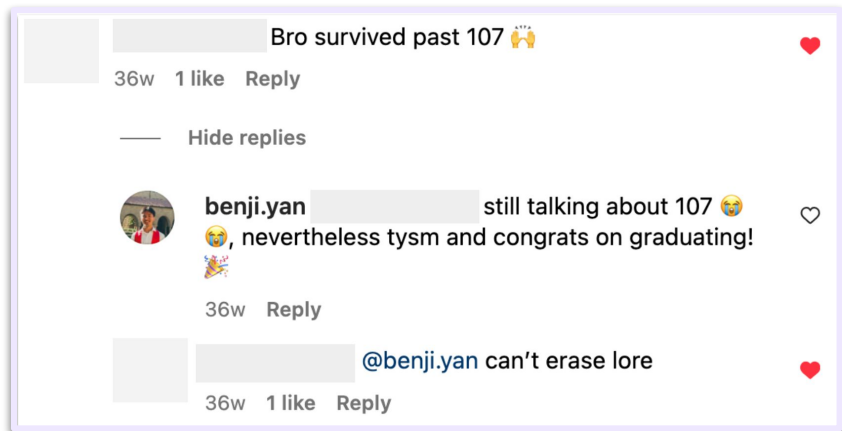
- Even after 4 years of studying CS or so, I’m still figuring out what kind of CS I really want to do lol. There’s plenty of areas I’ve decided I won’t do well in.

But my dream is to become a full-time lecturer in CS somewhere, someday. But we’ll see haha, fingers crossed, praying to the stars that things work out 🌟.



- I – and I’m sure many others – would not still be in my chosen field of study, if it weren’t for the compassion of students & others, and I can’t be more grateful.





A friend **who really helped me power through** 107 when I needed it (context: my IG grad post 🎓)

The 4 levels begin now. Good luck!
Level 1 disarmed. How about the next one?
Level 2 disabled. Keep going!
Level 3 disconnected. And lastly...
Level 4 deactivated. You are granted access to the master vault.
Congratulations! You solved all the levels!
Summarize your vault experience in one word: |

Finally finished **Binary Bomb** (as a **107 TA** 🌈); it only took 4 years!



So many blessings in the past years, and with this bright winter one of my favorite times TAing a course, **ty for making this time for special** 🥹

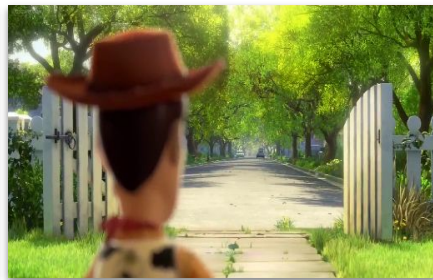
I have learned so much from teaching and from you.

So yeah, thank you!!!

I'm so grateful you're here

Final Lab Password: **thankyou**

what if the secret level was just
the friends we made along the way



Thank you for being here, and keep in touch!

➔ Happy to chat about CS107, heap allocators, final review! And also:

- 🌲 CS Coterm, CS or Math major
- 📚 Pursuing Creative Writing
- ✈️ Studying abroad, e.g., Oxford
- 👩🏫 CS classes, teaching, activities
- 🖼️ Anime, manga, films, Toy Story :)
- 🏯 S3 of Jujutsu Kaisen (my special)
- 💬 Language learning, Jerry Cain lore
- ❤️ Anything really! (life, Stanford, etc.)

➔ Please feel free to reach out anytime, and lmk if you need anything!

✉️ bbyan@stanford.edu, 📞 507-244-0751, or 📷 IG at @benji.yan