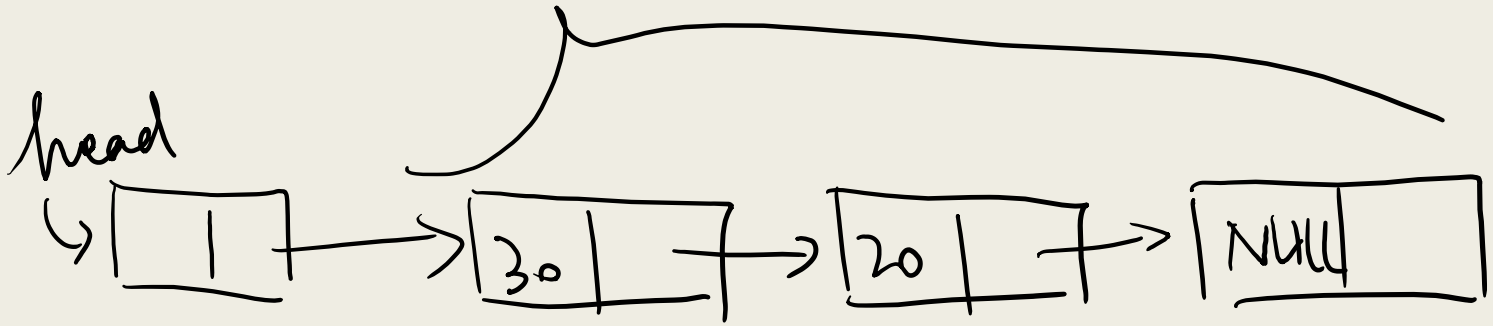
CS 449 REC 5

Announcements

- NEW!: Q&A session, and C scaffolding
- REC SLIDEs ARE NOW HERE: https://github.com/wenyuli08/CS449_Rec_Fall2020
  - *I also sent these materials to Dr. Khattab and you'll see it on Canvas*
  - *Please read them if you're struggling with the basics!*
- Let me know how to better help you!

Quiz

- Where do the rest of the queue go if I do head = NULL;?

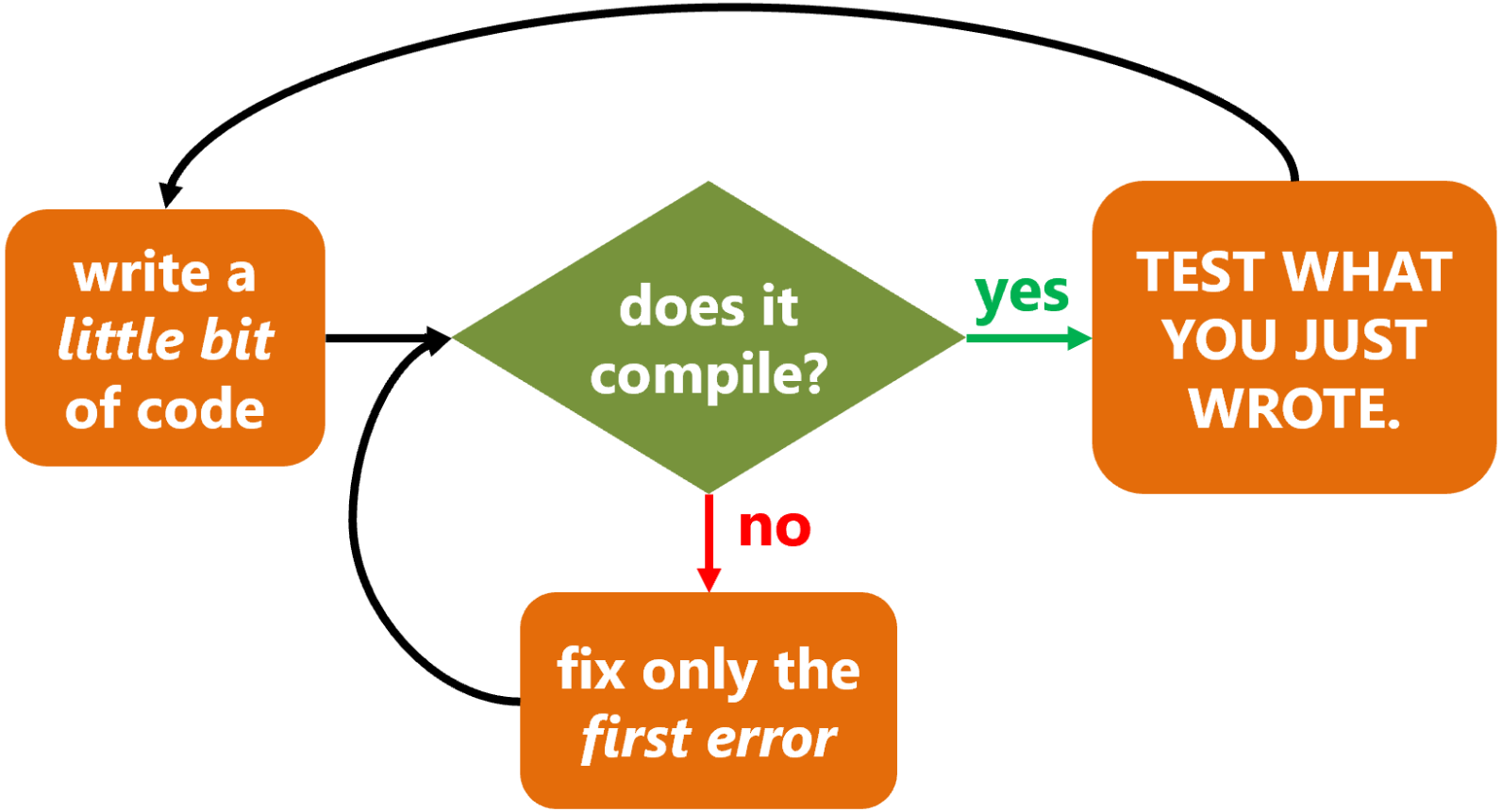


```
remove_head() {
...
    list_ele_t *node;
    node = q->head;
    q->head = q->head->next;
    free(_____);
}
```

_____ = ?

Schedule

- A bit of debugging…
- More queue lab
- Bomb lab


write a
little bit
of code
does it
compile?
yes
TEST WHAT
YOU JUST
WROTE.
no
fix only the
first error

Warnings & Errors

- -Wall: enable all warnings
- -Werror: make all warnings into errors
- Gcc -g
- https://gcc.gnu.org/onlinedocs/gcc/

Assert(cond)

- True: do nothing; False: stderr & abort
- #include <assert.h>
- Only to test things that should be impossible/certain to happen at all times
  - *i.e. do NOT use it on function parameters*

Break point & Watch point (gdb)

- Break point
  - **b func** will pause whenever **func** is called.
  - **b mymalloc.c:45** will pause when line 45 of mymalloc.c is reached.
  - **b *0x8004030** will pause when the PC gets to address 0x8004030.
  - **b** *location* **if x == 5** will pause at a location but only if the condition is satisfied. location can be any of the above.
  - **tb** *location* is a breakpoint that only happens once – it's deleted after the first time it's hit. (you can make these conditional too.)
- Watch point
  - **watch globalvar** will pause when a global variable is changed.
  - **watch localvar** will only work when you are paused in a function, and it will last until the local variable goes out of scope.
  - **rwatch** and **awatch** work the same, except they pause when a variable is **r**ead (rwatch) or on **a**ll accesses (awatch).

```c
int main() {
    // The comments at the ends of the lines show what list_print should output.
    Node* head = create_node(1);
    list_print(head);                   // 1
    Node* end = list_append(head, 2);
    list_print(head);                   // 1 -> 2
    end->next = create_node(3);
    list_print(head);                   // 1 -> 2 -> 3
    head = list_prepend(head, 0);
    list_print(head);                   // 0 -> 1 -> 2 -> 3
    list_append(head, 4);
    list_print(head);                   // 0 -> 1 -> 2 -> 3 -> 4
    list_append(head, 5);
    list_print(head);                   // 0 -> 1 -> 2 -> 3 -> 4 -> 5

    head = list_remove(head, 5);
    list_print(head);                   // 0 -> 1 -> 2 -> 3 -> 4
    head = list_remove(head, 3);
    list_print(head);                   // 0 -> 1 -> 2 -> 4
    head = list_remove(head, 0);
    list_print(head);                   // 1 -> 2 -> 4
    list_free(head);

    return 0;
}
```

Assuming you took 445…

- So you know linked lists: write a test program yourself, it'd be easier to manipulate
  - *A sample here*
  - *Do NOT copy paste, different assignment*

Memory Allocation

- Free everything you malloced exactly ONCE
- Int *arr = malloc(sizeof(int)*10);
  - *Allocates 40 bytes on the heap*
  - ***Malloc** makes a block of bytes at least this big, and returns a pointer to it*
- Queue *q = malloc(sizeof(Queue));
  - *≈ new in Java*
- After you free a variable on heap, any pointer to it becomes INVALID
  - *Could be where seg fault comes from*
  - *You can point to it, but do NOT access it*
- Everything you malloc'd contains garbage
  - *Use memset or Calloc**

Allocate space for the strings

Malloc	Copy	Assign	Debug
Use malloc on the size of the string, like last slide	Copy the string passed in as arg to the pointer	Assign the pointer to field “value” of the list_ele_t node	Debug using the methods mentioned above

When you free the queue (order doesn’t matter (why?))

01

Free EVERY NODE in the queue: free(node) (and strings: free(node->value))

- Access the pointers pointing to the node

02

Free the struct pointer of type queue_t (you possibly need queue_t to access strings & nodes)

QUESTIONS?

BOMB LAB

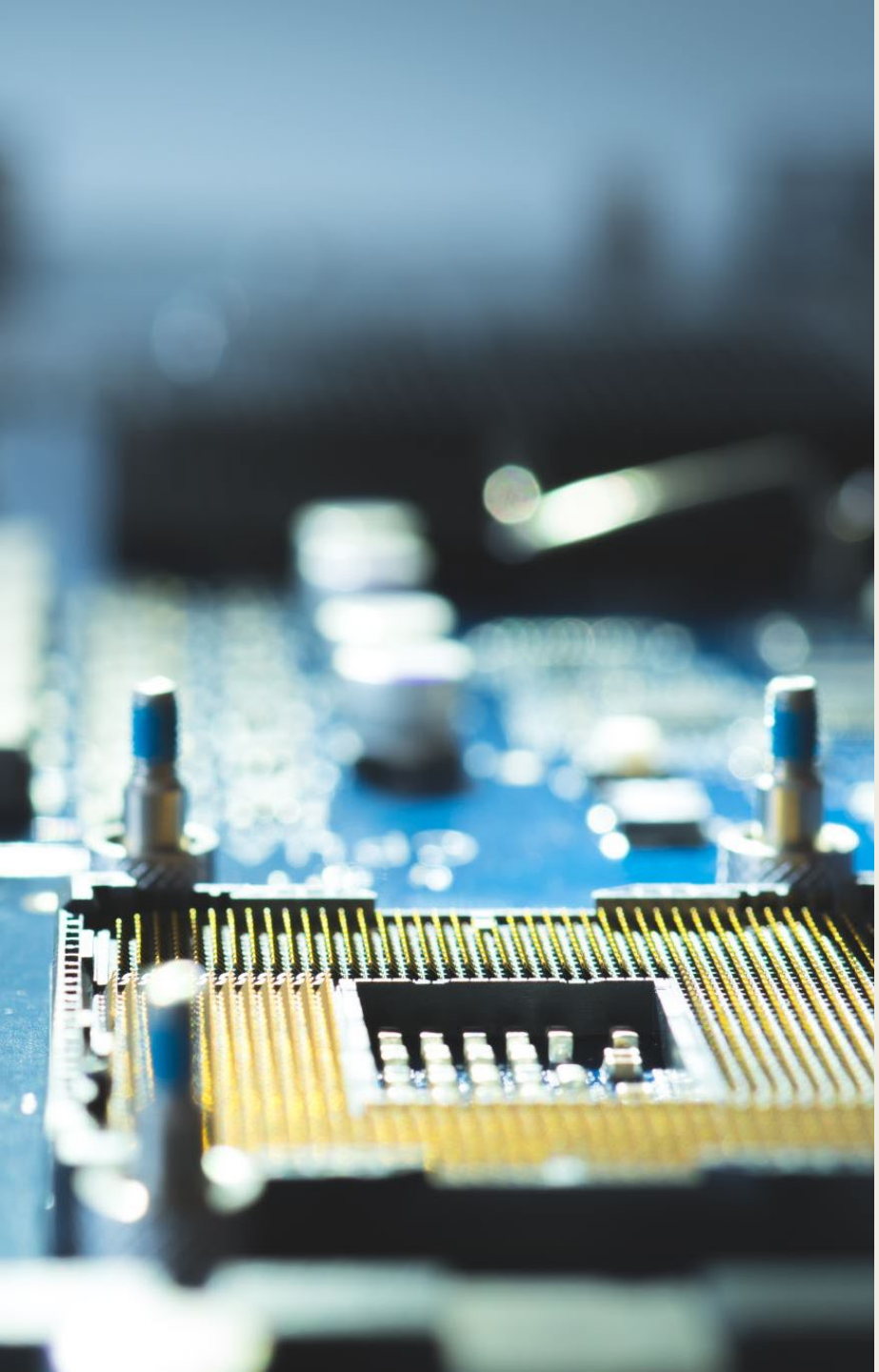
Don’t panic! It’s not assigned yet

Bomb Lab (guessing the password)

- ./bomb to start
  - *You start typing the first “password” after “Have a nice day!”*
- You can always stop the lab using ctrl+c w/o penalty…

```
[thoth ~/private/cs449_ta/lab4/bomb66]: ./bomb
Welcome to my fiendish little bomb. You have 4 phases with
which to blow yourself up. Have a nice day!
^CSo you think you can stop the bomb with ctrl-c, do you?
Well...OK. :-)
```

PDF WALKTHROUGH



Why assembly code?

- Direct manipulation of hardware
- Creating device drivers and compilers
- Debugging
  - If all you have is a core dump to work with, you can still debug
- Speed optimization
  - Total control over your hardware allows you to optimize speed and efficiency
- Cyber Security
  - Assembly code hides nothing, so you can poke around in registers and figure out what a program is doing

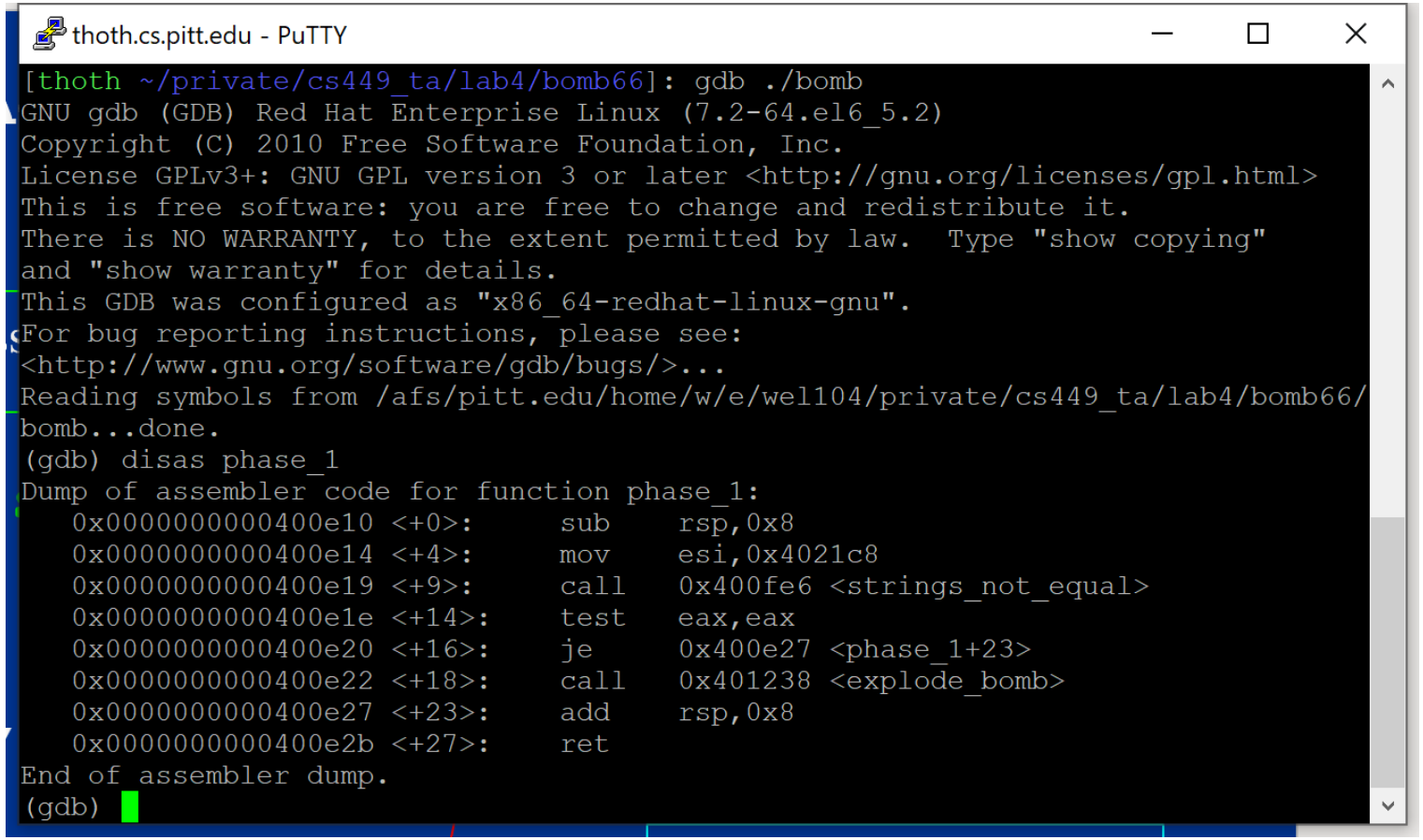
RAX	Return value (if exists and < 64 bits)
RBX	
RCX	4
RDX	3
RSP	SP = Stack Pointer (current location, growing downwards)
RBP	BP = Base Pointer (base of stack frame/start to stack)
RSI	2, SI = Source Index (for copy)
RDI	1, DI = Dest Index (for copy)
RIP	
R8-15	R8 5 R9 6

X86-64 registers

- Caller-saved registers/volatile: NOT preserved across function calls
  - RAX, RCX, RDX, R8, R9, R10, R11
- Callee-saved registers: to hold values should be preserved across function calls
  - RBX, RBP, RDI, RSI, RSP, R12, R13, R14, and R15

MOV vs LEA (load effective address)

- MOV: loads the actual value at address
  - *Mov src dest: dest = src*
- LEA: loads a pointer to the item
  - *Lea addr dest: dest = addr*


thoth.cs.pitt.edu - PuTTY
[thoth ~/private/cs449_ta/lab4/bomb66]: gdb ./bomb
GNU gdb (GDB) Red Hat Enterprise Linux (7.2-64.el6_5.2)
Copyright (C) 2010 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-redhat-linux-gnu".
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>...
Reading symbols from /afs/pitt.edu/home/w/e/wel104/private/cs449_ta/lab4/bomb66/
bomb...done.
(gdb) disas phase_1
Dump of assembler code for function phase_1:
   0x0000000000400e10 <+0>:   sub   rsp,0x8
   0x0000000000400e14 <+4>:   mov   esi,0x4021c8
   0x0000000000400e19 <+9>:   call   0x400fe6 <strings_not_equal>
   0x0000000000400e1e <+14>:   test   eax,eax
   0x0000000000400e20 <+16>:   je   0x400e27 <phase_1+23>
   0x0000000000400e22 <+18>:   call   0x401238 <explode_bomb>
   0x0000000000400e27 <+23>:   add   rsp,0x8
   0x0000000000400e2b <+27>:   ret
End of assembler dump.
(gdb)

More operations

- Cheat sheet: Stanford CS107

More on this next time (and maybe hands-on examples)