CS 449 REC 4

QUEUE LAB

ANNOUNCEMENTS

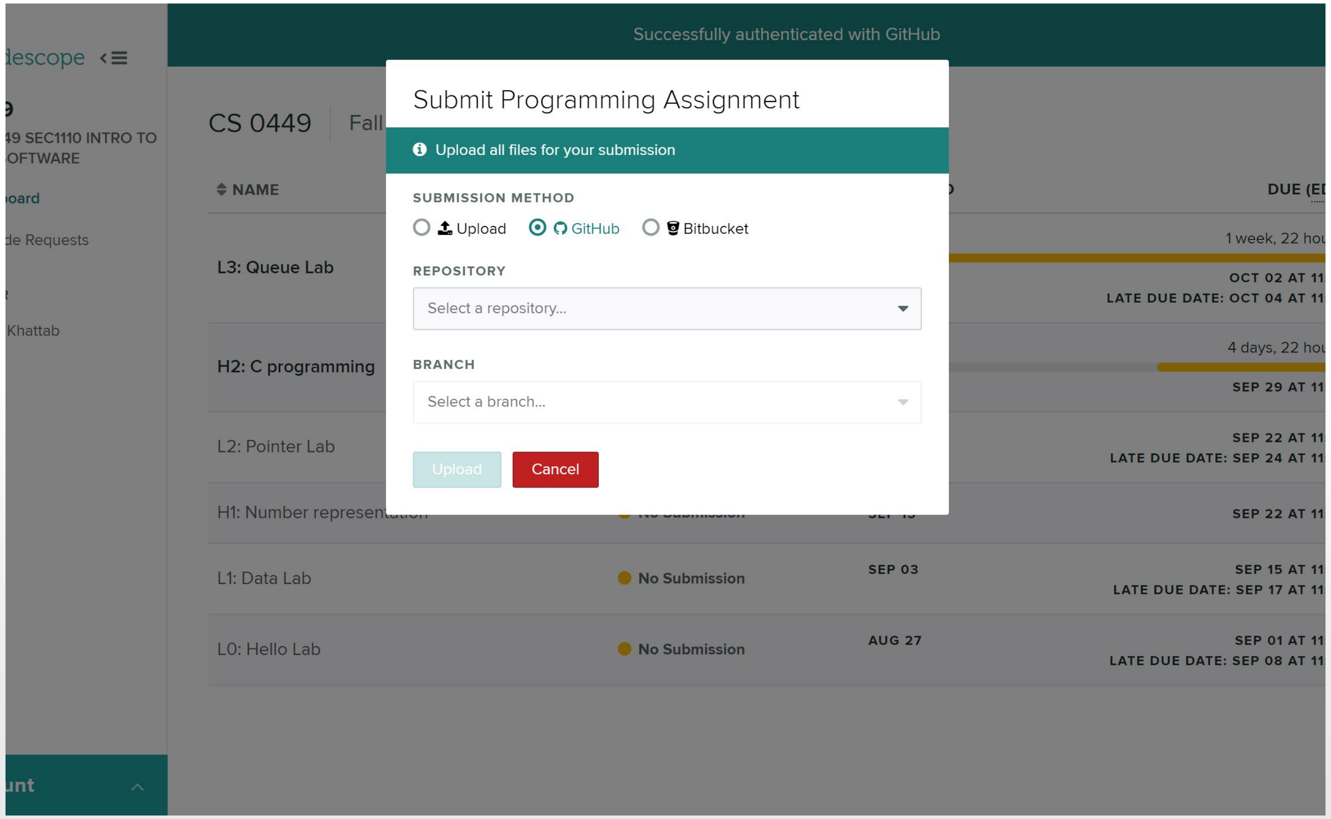
Office hour changed: Wed 2:30-4:00

Queue lab is out (start early!)

Check off meeting

Email me if you want (possibly) faster reply (not on Slack, sorry)

BTW…YOU CAN NOW UPLOAD TO GRADESCOPE USING GITHUB



Successfully authenticated with GitHub

escope

49 SEC1110 INTRO TO
OFTWARE

oard
de Requests
Khattab

unt

CS 0449 | Fall

NAME			DUE (ED
L3: Queue Lab			1 week, 22 hou OCT 02 AT 11 LATE DUE DATE: OCT 04 AT 11
H2: C programming			4 days, 22 hou SEP 29 AT 11
L2: Pointer Lab			SEP 22 AT 11 LATE DUE DATE: SEP 24 AT 11
H1: Number represen			SEP 22 AT 11
L1: Data Lab	No Submission	SEP 03	SEP 15 AT 11 LATE DUE DATE: SEP 17 AT 11
L0: Hello Lab	No Submission	AUG 27	SEP 01 AT 11 LATE DUE DATE: SEP 08 AT 11

Submit Programming Assignment

Upload all files for your submission

SUBMISSION METHOD

Upload [✓ GitHub] Bitbucket

REPOSITORY

Select a repository...

BRANCH

Select a branch...

REVIEW + LAB 2

- Int arr[5] = {1,3,7,5,2};
- *(arr+3) = ?

WHAT'S WRONG WITH THIS CODE?

```
void foo(int *p){
        int j;
        *p = j;
}
void bar() {
        int i = 10;
        foo(&i);
        printf("i = %d\n", i);
}
```

Credits to Berkeley CS 61C by Nick Riasanovsky

```
void foo(int *p) {
   int j;
   *p = j;
}

void bar() {
   int i=10;
   foo(&i);
   printf("i = %d\n", i);
}
```

`j` is uninitialized (garbage), copied into `*p`

Using `i` which now contains garbage

FROM STACK TO HEAP

```c
#include <stdio.h>

double multiplyByTwo (double input) {
  double twice = input * 2.0;
  return twice;
}

int main (int argc, char *argv[])
{
  int age = 30;
  double salary = 12345.67;
  double myList[3] = {1.2, 2.3, 3.4};

  printf("double your salary is %.3f\n", multiplyByTwo(salary));

  return 0;
}
```

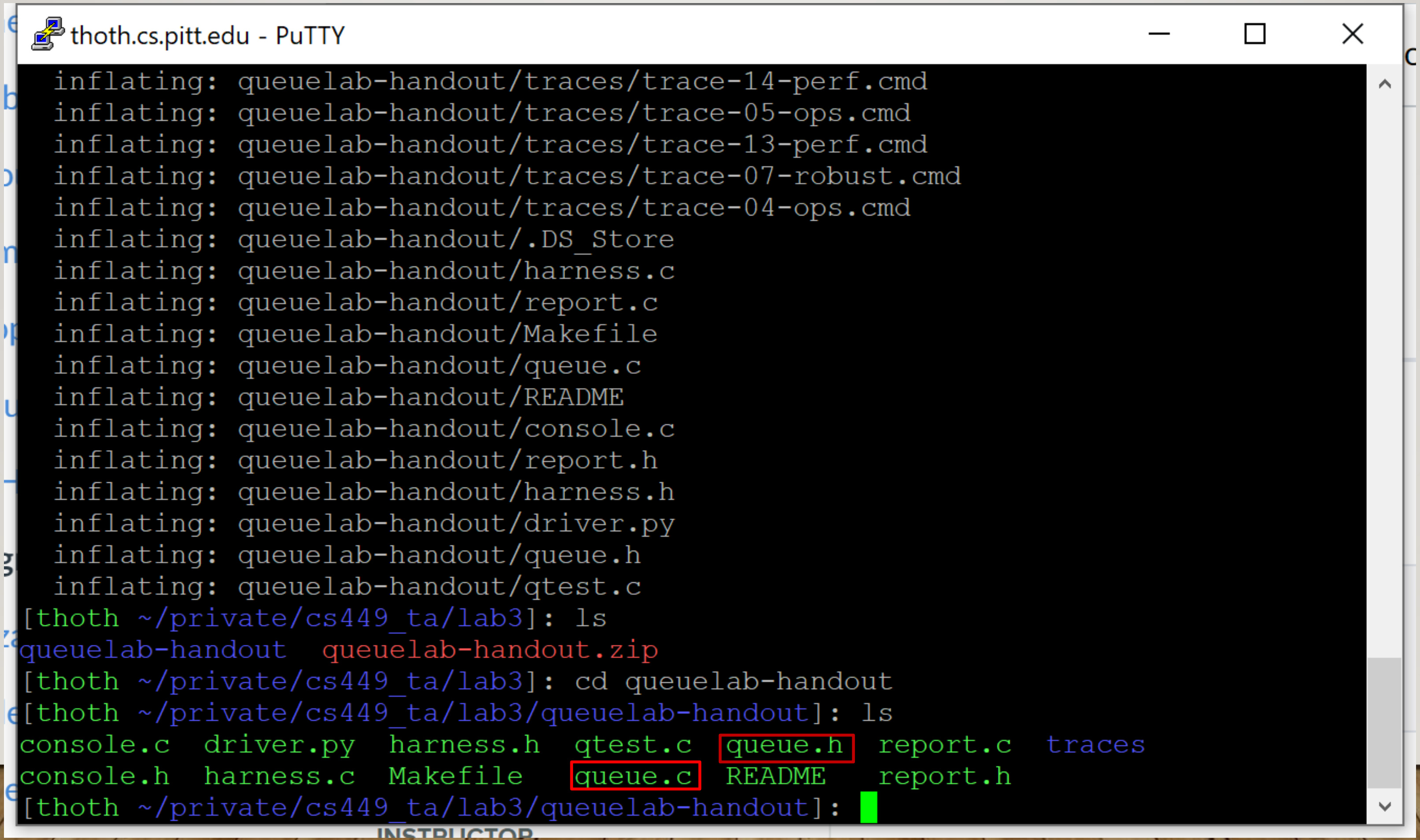
```c
#include <stdio.h>
#include <stdlib.h>

double *multiplyByTwo (double *input) {
  double *twice = malloc(sizeof(double));
  *twice = *input * 2.0;
  return twice;
}

int main (int argc, char *argv[])
{
  int *age = malloc(sizeof(int));
  *age = 30;
  double *salary = malloc(sizeof(double));
  *salary = 12345.67;
  double *myList = malloc(3 * sizeof(double));
  myList[0] = 1.2;
  myList[1] = 2.3;
  myList[2] = 3.4;

  double *twiceSalary = multiplyByTwo(salary);

  printf("double your salary is %.3f\n", *twiceSalary);

  free(age);
  free(salary);
  free(myList);
  free(twiceSalary);

  return 0;
}
```

STACK VS HEAP

Stack	Heap
• Local variable only • Fast access • Memory will not be fragmented • No need to free • Linear	• Malloc and free • Slower access • Memory fragmentation • YOU are in charge of variable space • Hierarchical

BASICS WALKTHROUGH

COPY THE FILES ON PUTTY


thoth.cs.pitt.edu - PuTTY

```
  inflating: queuelab-handout/traces/trace-14-perf.cmd
  inflating: queuelab-handout/traces/trace-05-ops.cmd
  inflating: queuelab-handout/traces/trace-13-perf.cmd
  inflating: queuelab-handout/traces/trace-07-robust.cmd
  inflating: queuelab-handout/traces/trace-04-ops.cmd
  inflating: queuelab-handout/.DS_Store
  inflating: queuelab-handout/harness.c
  inflating: queuelab-handout/report.c
  inflating: queuelab-handout/Makefile
  inflating: queuelab-handout/queue.c
  inflating: queuelab-handout/README
  inflating: queuelab-handout/console.c
  inflating: queuelab-handout/report.h
  inflating: queuelab-handout/harness.h
  inflating: queuelab-handout/driver.py
  inflating: queuelab-handout/queue.h
  inflating: queuelab-handout/qtest.c
[thoth ~/private/cs449_ta/lab3]: ls
queuelab-handout  queuelab-handout.zip
[thoth ~/private/cs449_ta/lab3]: cd queuelab-handout
[thoth ~/private/cs449_ta/lab3/queuelab-handout]: ls
console.c  driver.py  harness.h  qtest.c  queue.h  report.c  traces
console.h  harness.c  Makefile   queue.c  README   report.h
[thoth ~/private/cs449_ta/lab3/queuelab-handout]:
```

```
/************** Operations on queue ************************/

/*
  Create empty queue.
  Return NULL if could not allocate space.
*/
queue_t *q_new();

/*
  Free ALL storage used by queue.
  No effect if q is NULL
*/
void q_free(queue_t *q);
```

NO NEED TO MODIFY

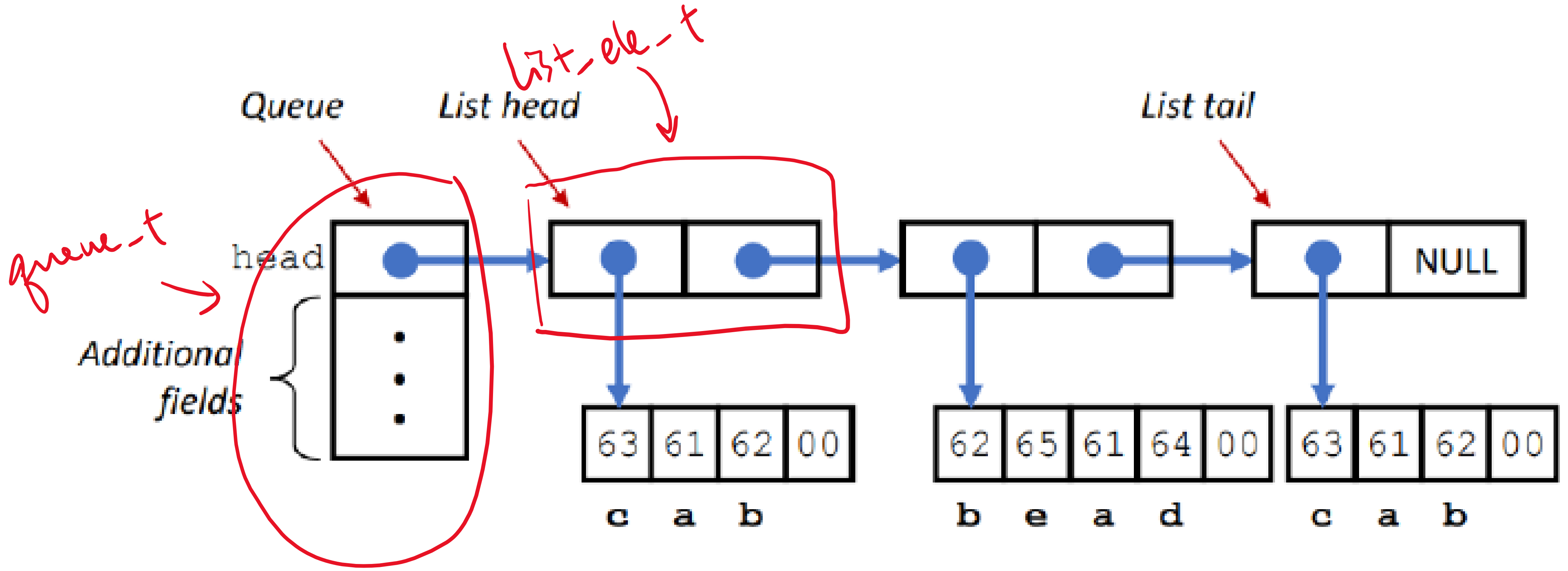


Figure 1: **Linked-list implementation of a queue.** Each list element has a value field, pointing to an array of characters (C's representation of strings), and a next field pointing to the next list element. Characters are encoded according to the ASCII encoding (shown in hexadecimal.)

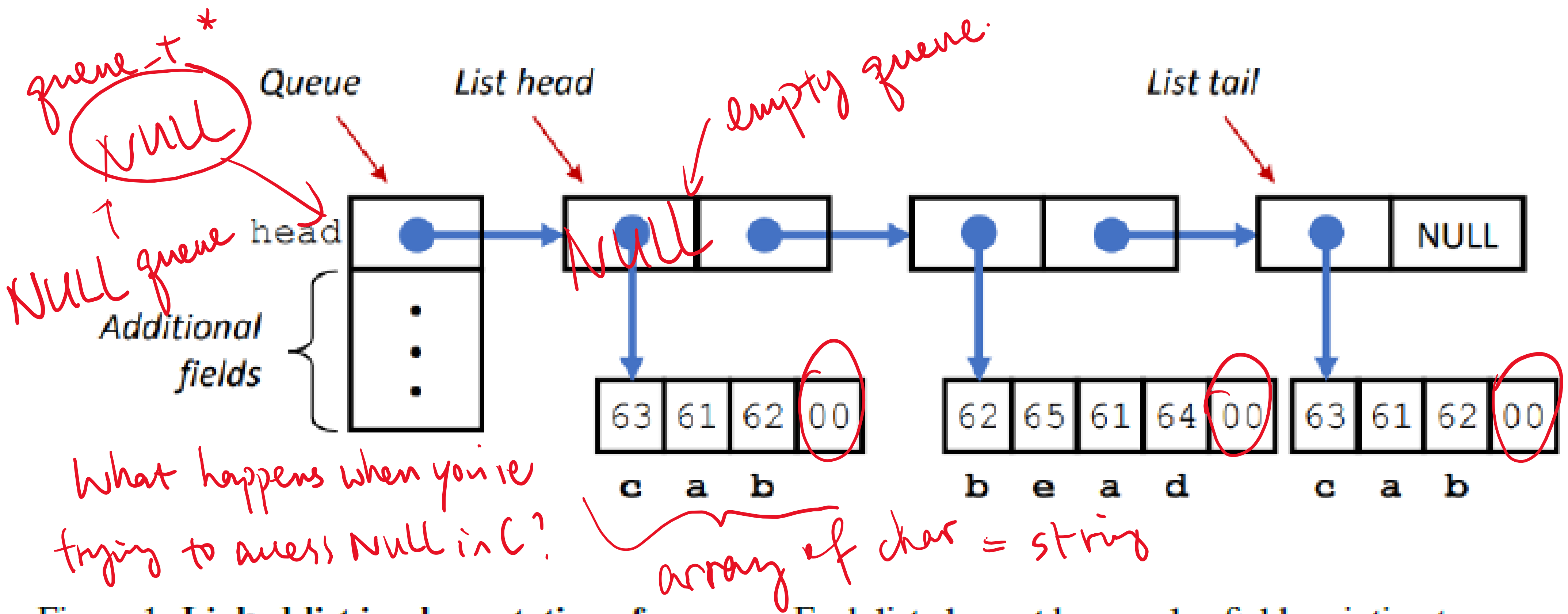
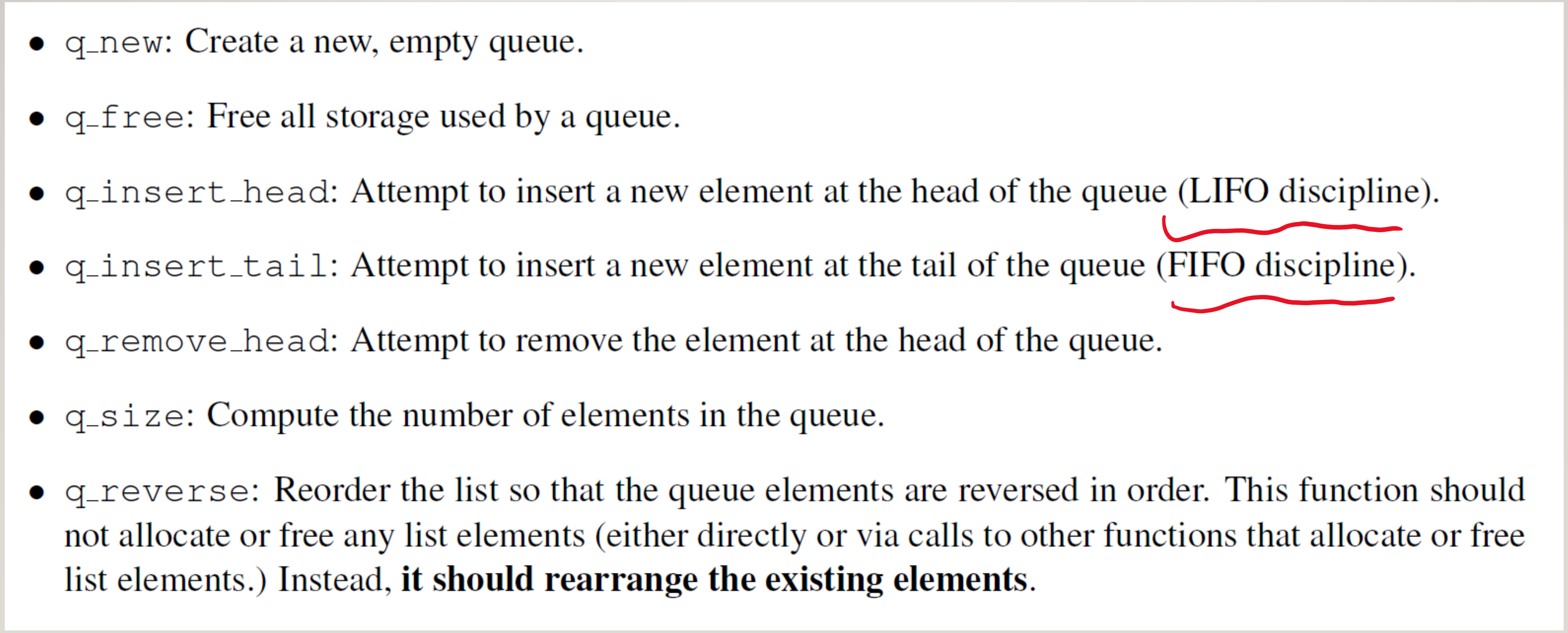


Figure 1: **Linked-list implementation of a queue.** Each list element has a value field, pointing to an array of characters (C's representation of strings), and a next field pointing to the next list element. Characters are encoded according to the ASCII encoding (shown in hexadecimal.)

WHEN YOU ARE STUCK, DRAW OUT SOMETHING LIKE THAT

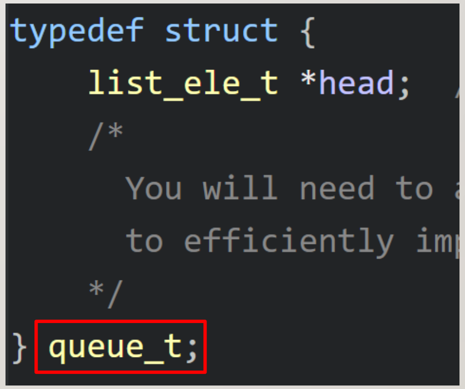
FUNCTIONS IN QUEUE.C…MORE IN COMMENTS



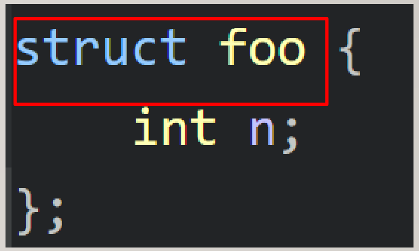
- `q_new`: Create a new, empty queue.
- `q_free`: Free all storage used by a queue.
- `q_insert_head`: Attempt to insert a new element at the head of the queue (LIFO discipline).
- `q_insert_tail`: Attempt to insert a new element at the tail of the queue (FIFO discipline).
- `q_remove_head`: Attempt to remove the element at the head of the queue.
- `q_size`: Compute the number of elements in the queue.
- `q_reverse`: Reorder the list so that the queue elements are reversed in order. This function should not allocate or free any list elements (either directly or via calls to other functions that allocate or free list elements.) Instead, **it should rearrange the existing elements**.

NOTES

- Printf anything you are unsure of/keep track of them
- You can define Macros/structs/functions in queue.h
  - #define Addnumbers(x, y) (x+y)
  - Struct or typedef struct?



```
typedef struct {
    list_ele_t *head;
    /*
      You will need to
      to efficiently im
    */
} queue_t;
```



```
struct foo {
    int n;
};
```

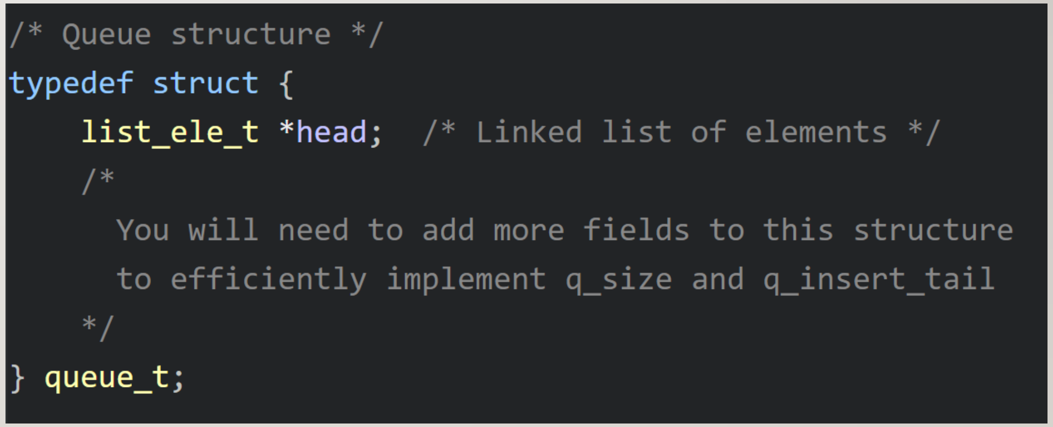
- When you are adding a new string, remember not only the string takes up memory, the node does too/the same with removing…
  - Malloc is in <stdlib.h>
  - Using sizeof
  - Using strlen/strcmp   Why are they not safe to use?
  - String manipulation functions usually start with strxxx

(CONT.)

- You can not assume an upper bound on string length! i.e. you can’t do char[constant]
- When you see a function in c, search for its document and pay attention to return values/edge cases!
- Recursion (x) loops (√)
- Most functions involve: allocate/deallocate space -> copying string -> handling edge cases
- Reverse: you are not supposed to use any malloc/free functions
  - Perhaps use a while loop…

INSERT_TAIL & SIZE: O(1)

- Naïve algorithm
- O(1) meaning?
- What fields can be added to queue_t?



```
/* Queue structure */
typedef struct {
    list_ele_t *head;  /* Linked list of elements */
    /*
      You will need to add more fields to this structure
      to efficiently implement q_size and q_insert_tail
    */
} queue_t;
```

INSERT_TAIL & SIZE: O(1)

- O(1) meaning?
- You can add anything as a field in queue_t…
  - Perhaps a pointer to point to the soon-to-be-added value
  - Perhaps a value to hold size
  - Don’t forget to update the fields

EDGE CASES

- Removing from empty or NULL queues
- Handling **head & tail**
  - Either is null
- Return value of malloc/free (i.e. could not allocate)
- Queue size == 0
- Some of them are in the comments

WHAT'S WRONG WITH THIS CODE

```
typedef struct node {
 struct node* next;
 int val;
} Node;

int findLastNodeValue(Node* head)
{
 while (head->next != NULL) {
   head = head->next;
 }
 return head->val;
}
```

Credits to Berkeley CS 61C by Nick Riasanovsky

```
typedef struct node {
  struct node* next;
  int val;
} Node;

int findLastNodeValue(Node* head) {
  while (head->next != NULL)
    head = head->next;
  return head->val;
}
```

What if `head` is NULL?

No warnings! Just Seg Fault that needs finding!

AGAIN…

Time management!

Draw the queue!

Remember printf/gdb!