Midterm Review

1. Modulo Arithmetic and Character I/O

```
void f(unsigned int n) {
    do {
        putchar('0' + (n % 10));
    } while (n /= 10);
    putchar('\n');
}
```

- What does f(837) produce?
- What does this function do?

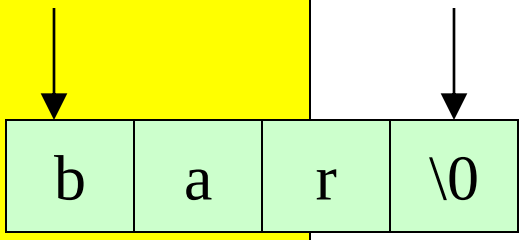
1. Modulo Arithmetic and Character I/O

```
void f(unsigned int n){
   for ( ; n; n /= 10)
      putchar('0' + (n % 10));
   putchar('\n');
}
```

- When is the answer different?

2. Pointers and Strings

```
void f(char *s) {
  char *p = s;
  while (*s)
    s++;
  for (s--; s>p; s--,p++) {
    char c = *s;
    *s = *p;
    *p = c;
  }
}
```



- What does this function do?

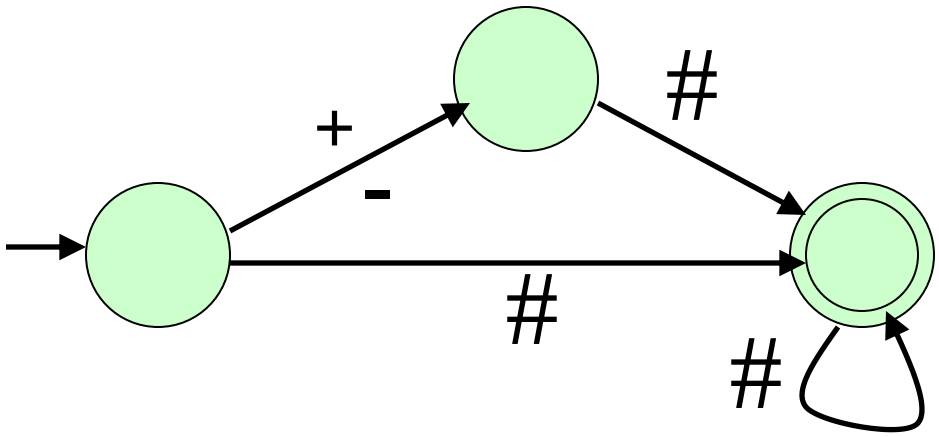
3. Deterministic Finite Automata

Identify whether or not a string is a floating-point number

- Valid numbers
  - "-34"
  - "78.1"
  - "+298.3"
  - "-34.7e-1"
  - "34.7E-1"
  - "7."
  - ".7"
  - "999.99e99"
- Invalid numbers
  - "abc"
  - "-e9"
  - "1e"
  - "+"
  - "17.9A"
  - "0.38+"
  - "."
  - "38.38f9"

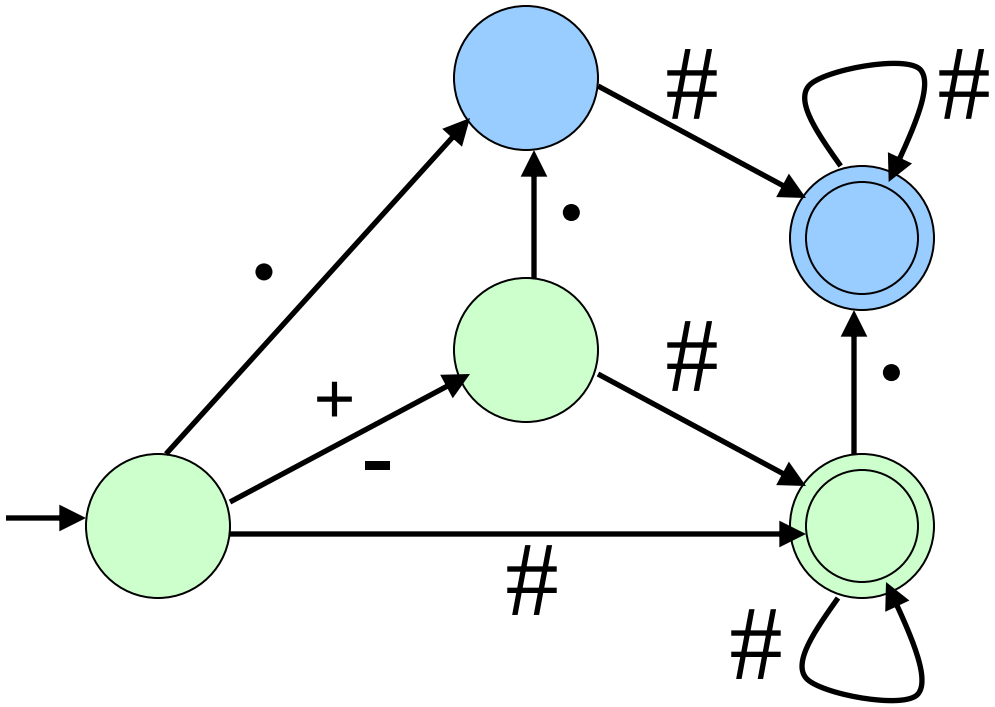
4. Deterministic Finite Automata

- Optional "+" or "-"
- Zero or more digits



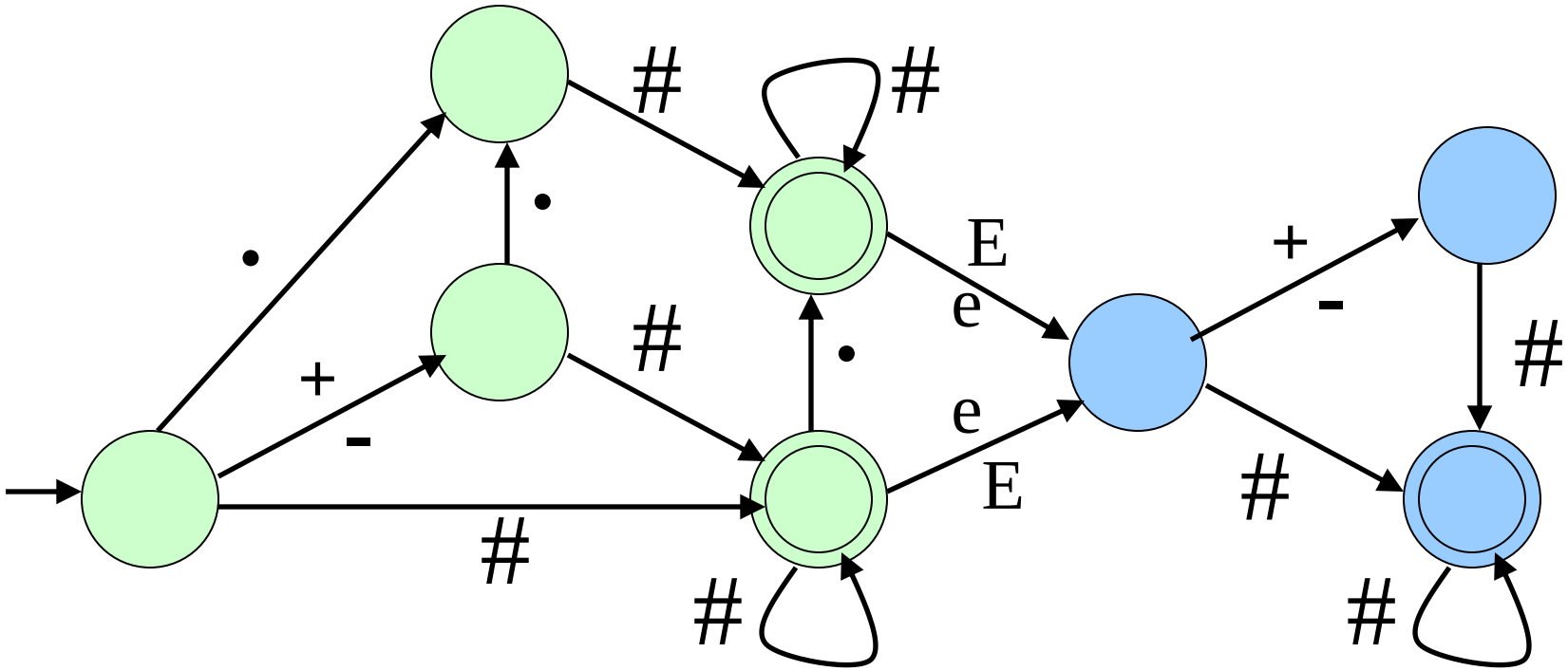
4. Deterministic Finite Automata

- Optional "+" or "-"
- Zero or more digits
- Optional decimal point
  - Followed by zero or more digits



4. Deterministic Finite Automata

- Optional "+" or "-"
- Zero or more digits
- Optional decimal point
  - Followed by zero or more digits
- Optional exponent "E" or "e"
  - Followed by optional "+" or "-"
  - Followed by one or more digits



4: Abstract Data Types

- Interface for a Queue (a first-in-first-out data structure)

```
#ifndef QUEUE_INCLUDED
#define QUEUE_INCLUDED
typedef struct Queue_t *Queue_T;


Queue_T Queue_new(void);
int Queue_empty(Queue_T queue);
void Queue_add(Queue_T queue, void* item);
void* Queue_remove(Queue_T queue);
#endif
```

4: Abstract Data Types

- An implementation for a Queue (in queue.c)

```
#include <stdlib.h>
#include <assert.h>
#include "queue.h"
struct list {
  void* item;
  struct list *next;
};
struct Queue_t {
  struct list *head;
  struct list *tail;
};
```

Why void*?

Why declared here and not in queue.h?

4: Abstract Data Types

- An implementation for a Queue_new

```
Queue_T Queue_new(void) {
  Queue_T queue = malloc(sizeof *queue);
  assert(queue != NULL);
  queue->head = NULL;
  queue->tail = NULL;
  return queue;
}
```

Implement a check for whether the queue is empty.

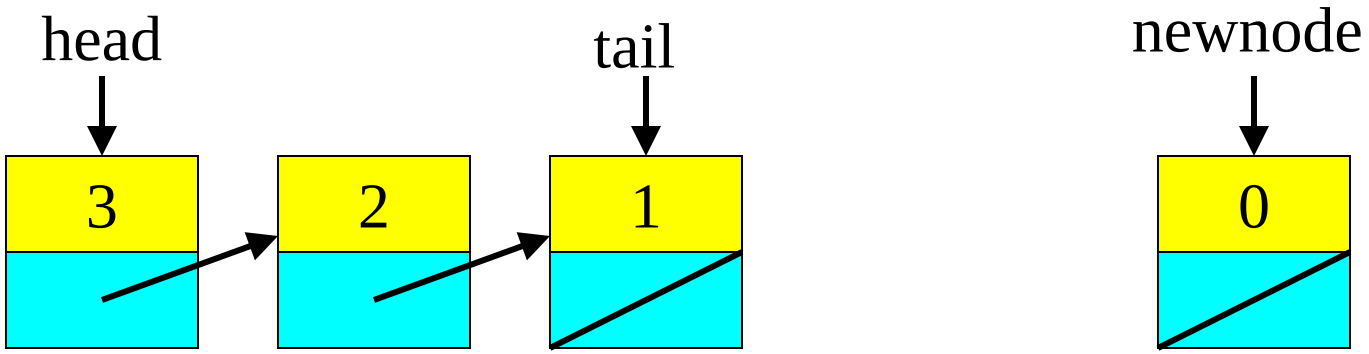
4: Abstract Data Types

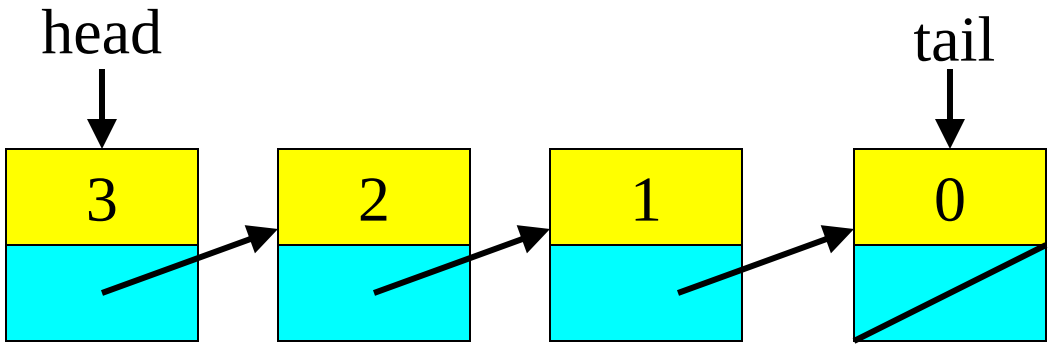
- An implementation for a `Queue_empty`

```
int Queue_empty(Queue_T queue) {
  assert(queue != NULL);
  return queue->head == NULL;
}
```

4: Abstract Data Types

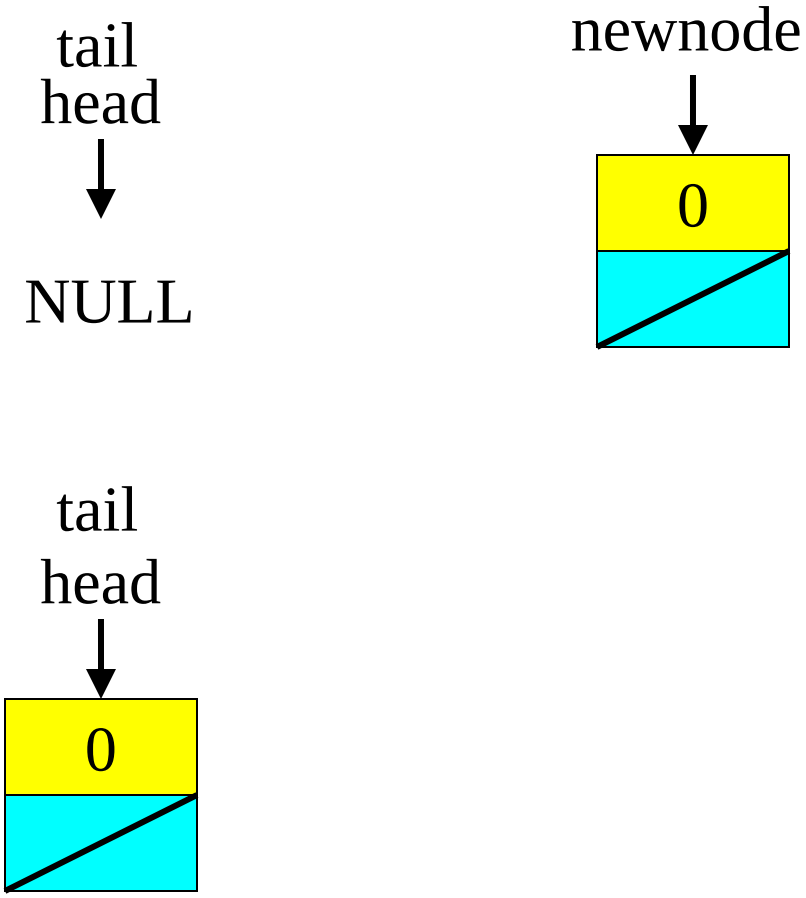
- An implementation for a `Queue_add`





4: Abstract Data Types

- An implementation for a `Queue_add`



Queue_add() Implementation

```
void Queue_add(Queue_T queue, void *item) {
   struct list *newnode;
   assert(queue != NULL);
   newnode = (struct list*)malloc(sizeof(*newnode));
   assert(newnode != NULL);
   newnode->item = item;
   newnode->next = NULL;
   if (queue->tail == NULL)
      queue->head = newnode;
   else
      queue->tail->next = newnode;
   queue->tail = newnode;
}
```

4. ADT Common Mistakes

- Adding to the queue
  - Implementing a *stack* rather than a queue
    - Adding element to the head, rather than the tail
  - Not handling the case where the queue is empty
  - Missing assert() after call to malloc() for new entry
- Removing from the queue
  - Missing assert() when removing an element from an empty queue
  - Not handling removing the last item from the queue
  - Not doing a free() to return space used by the head element

5. Bit-Wise Manipulations

- Consider the following code, where **k** is an unsigned int:

```
printf("%u\n", k - ((k >> 2) << 2));
```

- What does the code do? Rewrite the line of code in a more efficient way.

- Replaces last two bits with 0
  - Same as doing: k & 3

6. What Does This Function Do?

```
char* f(unsigned int n) {
   int i, numbits = sizeof(unsigned int) * 8;
   char* ret = (char *) malloc(numbits + 1);
   for (i=numbits-1; i>=0; i--, n>>=1)
      ret[i] = '0' + (n & 1);
   ret[numbits] = '\0';
   return ret;
}
```

n = 19 00010011

7. Good Bug Hunting

- Consider this function that converts an integer to a string

```
char *itoa(int n) {
   char retbuf[5];
   sprintf(retbuf, "%d", n);
   return retbuf;
}
```

Not enough space

Temporary memory

- Where the **sprintf()** function "prints" to a formatted string, e.g., **sprintf(retbuf, "%d", 72)** places the string **"72"** starting at the location in memory indicated by the address **retbuf**:

Fixing the Bug: Rewrite

```
char *itoa(int n) {
    int size = 0;
    int temp = n;

    /* Count number of decimal digits in n */
    while (temp /= 10)
        size++;
    size++;

    /* If n is negative, add room for the "-" sign */
    if (n < 0)
        size++;
```

Fixing the Bug: Rewrite

```
    /* Allocate space for the string */
    char* retbuf = (char *) malloc(size + 1);
    assert(retbuf != NULL);

    /* Convert the number to a string of digits */
    sprintf(retbuf, "%d", n);

    return retbuf;
}
```

Void Pointer Review

- Used whenever the exact type of an object is unknown or when using a generic pointer
- A void pointer <-> Any pointer type
- No "void variable"
  - Cannot be dereferenced without typecasting

```
{
   void* vptr;
   int i = 10;
   vptr = &i;
  /*COMPILE-TIME ERROR */
   printf("%d\n", *vptr);
   printf("%d\n", *(int*)vptr);
}
```

```
{
   void* vptr;
   int i = 10;
   double d = 2.4;
   vptr = &i;
  /* Run-time bug */
   d = *(double*)vptr;
}
```

Complex Pointers

- int (*x)()
  - x is a pointer to a function returning an integer
- int *x()
  - x is a function returning an integer pointer
- int (*x)[]
  - x is a pointer to an array of integers
- int *x[]
  - x is an array of integer pointers
- int (*x[])()
  - x is an array of function pointers… all returning integers
- int *(*x)[]
  - x is a pointer to an array of integer pointers

•FUNCTION & ARRAY SYMBOLS HAVE HIGHER PRECEDENCE THAN POINTER SYMBOL

Example

```
int add(int a, int b);
int sub(int a, int b);

int main(void)
{
  int (*foo[2])(int, int);
  foo[0] = add;
  foo[1] = sub;

  printf("foo[0](1,2) = %d\n", foo[0](1,2));
  printf("foo[1](4,3) = %d\n", foo[1](4,3));
}
```

- OUTPUT –

foo[0](1,2) = 3

foo[1](4,3) = 1

Preparing for the Exam

- Studying for the exam
  - Read through lecture and precept nodes
  - Study past midterm exams
  - Read through exercises in the book
- Taking the exam
  - Read briefly through all questions
  - Strategize where you spend your time
- Exam logistics
  - 10/27 Thursday 10:30am-12:30pm in CLB, 201
  - No questions on UNIX tools (e.g., emacs, gcc, gdb, …)