# Midterm Review
# (overview of fall 2005 midterm)

COS 217



## 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

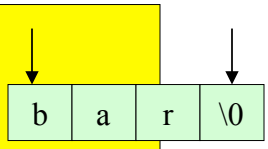
```
void f(unsigned int n){
   for ( ; n; n /= 10)
      putchar('0' + (n % 10));
   putchar('\n');
}
```

•When is the answer different?

3

## 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;
  }
}
```

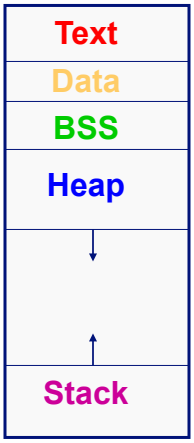



•What does this function do?

4

# 3. Short Answer

- In the memory layout for a UNIX process:
  - Why does the heap grow from the top down and the stack from the bottom up, instead of both growing from the top down or both growing from the bottom up?




5

# 4. Deterministic Finite Automata

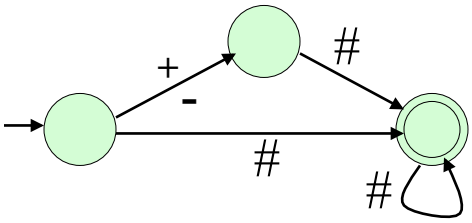
**Identify whether or not a string is a floating-point number**

- 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”

6

# 4. Deterministic Finite Automata

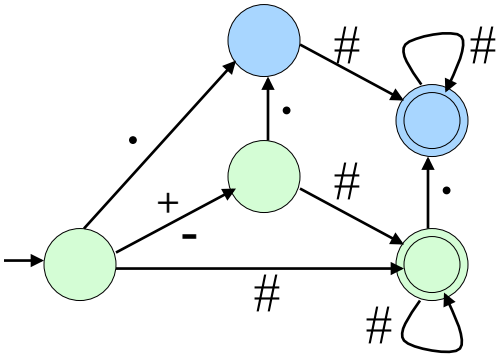
- Optional “+” or “-”
- Zero or more digits




7

# 4. Deterministic Finite Automata

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

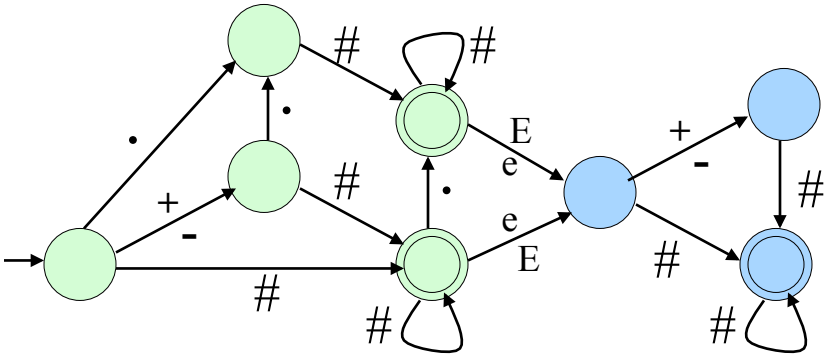



8

# 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




9

# 5: 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
```

10

# 5: Abstract Data Types

- Data structures for a Queue

```
struct list {
  void* item;
  struct list *next;
};
struct Queue_t {
  struct list *head;
  struct list *tail;
};
```

Why void*?

11

# 5: 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.

12

## 5: Abstract Data Types

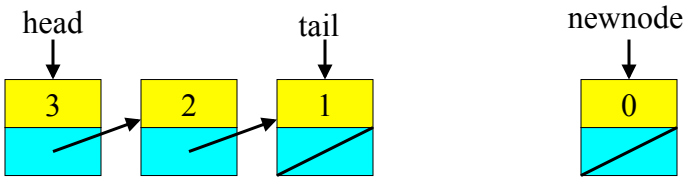
- An implementation for a `Queue_empty`

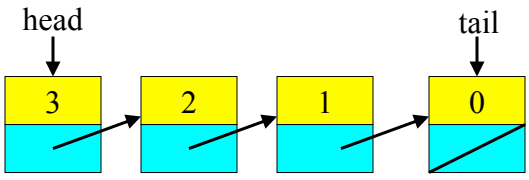
```
int Queue_empty(Queue_T queue) {
  assert(queue != NULL);
  return queue->head == NULL;
}
```

13

## 5: Abstract Data Types

- An implementation for a `Queue_add`

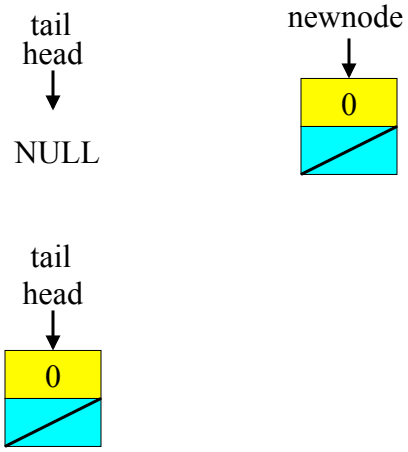



14

# 5: Abstract Data Types

- An implementation for a `Queue_add`




15

# 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;
}
```

16

# 5. 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

17

## Midterm Review
## (overview of spring 2008 midterm)

18

## 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.

19

## 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;
}
```

20

## 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**:

21

## 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;
}
```

## Common Errors for this Problem

- *Mishandle case where n is 0, which requires 1 character*
- *Assume n is unsigned, and not allocate space for the minus sign*
- *Omit the "assert(retbuf)" after the call to malloc*
- *Use "sizeof(n)" to compute the length: gives bytes in int type*
- *Extract each of the digits of n, using code similar to question 1b (but base 10). Perfectly valid, though using "sprintf" is simpler.*
- *An interesting, and clever, answer was to create a large array of characters as a local variable, use sprintf to place the string representation of n in the array, use strlen() to compute the length of the string, use malloc() to allocate the appropriate amount of space to retbuff, and then copy the string from the local variable to retbuf.*

24

# Preparing for the Exam

- Studying for the exam
  - Read through lecture and precept nodes
  - Study past midterm exams
  - Read through the relevant sections and the exercises in the book
- Taking the exam
  - Read briefly through all questions
  - Strategize regarding where you spend your time, and what you answer first (hint: the easy stuff)
- Exam logistics
  - Thursday 10-10:50am in *WATCH LISTSERV FOR UPDATES*
  - Closed book and notes. No computers or PDAs/phones allowed.
  - No questions on UNIX tools (e.g., emacs, gcc, gdb, …)

25