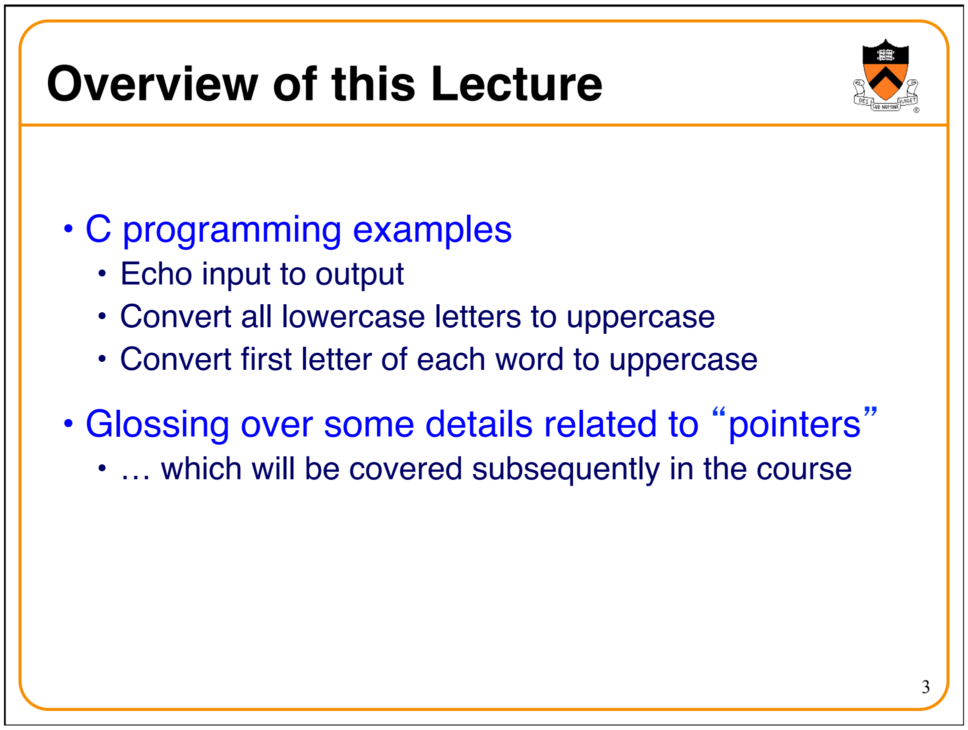
C Programming Examples

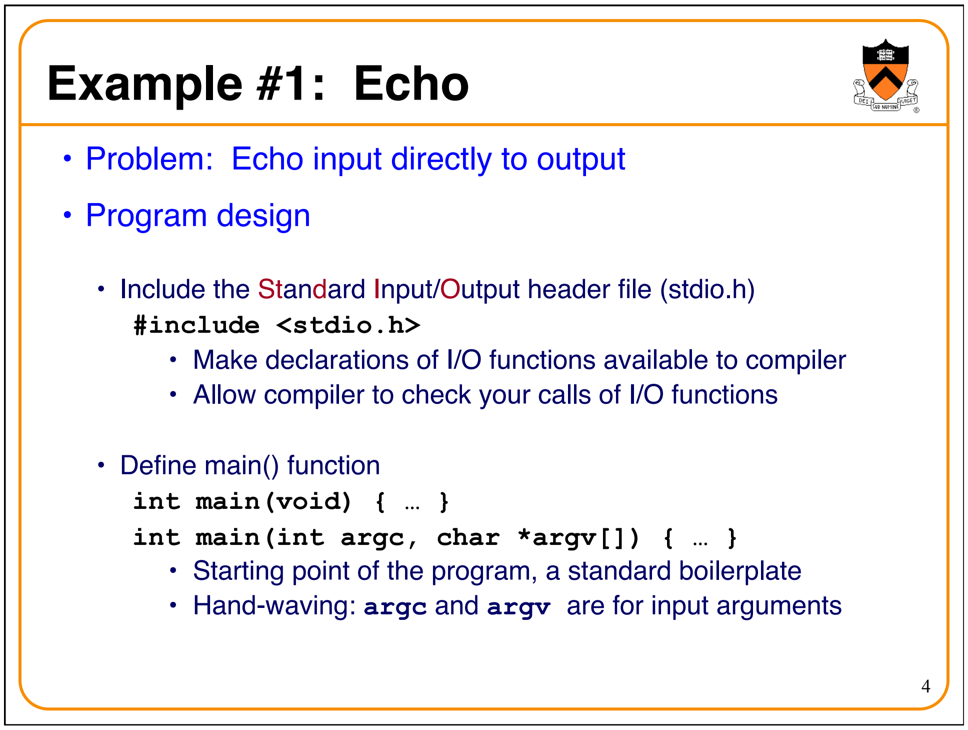
Goals of this Lecture

- Help you learn about:
  - The fundamentals of C
    - Program structure, control statements, character I/O
  - Deterministic finite state automata (DFA)
  - Expectations for programming assignments
- Why?
  - The fundamentals of C provide a foundation for the systematic coverage of C that will follow
  - A power programmer knows the fundamentals of C well
  - DFA are useful in many contexts
    - A very important context: Assignment 1
- How?
  - Through some examples



Overview of this Lecture

- C programming examples
  - Echo input to output
  - Convert all lowercase letters to uppercase
  - Convert first letter of each word to uppercase
- Glossing over some details related to “pointers”
  - … which will be covered subsequently in the course



Example #1: Echo

- Problem: Echo input directly to output
- Program design
  - Include the Standard Input/Output header file (stdio.h)

    ```
    #include <stdio.h>
    ```
    - Make declarations of I/O functions available to compiler
    - Allow compiler to check your calls of I/O functions
  - Define main() function

    ```
    int main(void) { … }
    int main(int argc, char *argv[]) { … }
    ```
    - Starting point of the program, a standard boilerplate
    - Hand-waving: **`argc`** and **`argv`** are for input arguments

Example #1: Echo (cont.)

- Within the main program
  - Read a single character

    ```
    c = getchar();
    ```
    - Read a single character from the "standard input stream" (stdin) and return it
  - Write a single character

    ```
    putchar(c);
    ```
    - Write a single character to the "standard output stream" (stdout)

5

Putting it All Together

```
#include <stdio.h>

int main(void) {
   int c;

   c = getchar();
   putchar(c);

   return 0;
}
```

Why **int** instead of **char**?

Why return a value?

6

Read and Write Ten Characters

- Loop to repeat a set of lines (e.g., `for` loop)
  - Three expressions: initialization, condition, and increment
  - E.g., start at 0, test for less than 10, and increment per iteration

```
#include <stdio.h>
int main(void) {
  int c, i;

  for (i=0; i<10; i++) {
    c = getchar();
    putchar(c);
  }

  return 0;
}
```

Why not this instead:
for (i = 1; i <= 10; i++)

7

Read and Write Forever

- Infinite `for` loop
  - Simply leave the expressions blank
  - E.g., `for ( ; ; )`

```
#include <stdio.h>
int main(void) {
  int c;

  for ( ; ; ) {
    c = getchar();
    putchar(c);
  }

  return 0;
}
```

When will this be executed?

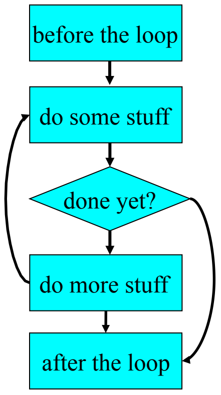
How would you terminate this program?

8

Read and Write Until End-Of-File

- Test for end-of-file
  - **`EOF`** is a global constant, defined in stdio.h
  - The **`break`** statement jumps out of the innermost enclosing loop

```
#include <stdio.h>
int main(void) {
  int c;
  for ( ; ; ) {
     c = getchar();
     if (c == EOF)
        break;
     putchar(c);
  }
  return 0;
}
```



9

Many Ways to Do the Same Job

```
for (c=getchar(); c!=EOF; c=getchar())
  putchar(c);
```

Which approach is best?

```
while ((c=getchar())!=EOF)
   putchar(c);
```

← Typical idiom in C, but messy side-effect in loop test

```
for (;;)  {
 c = getchar();
 if (c == EOF)
    break;
 putchar(c);
}
```

```
c = getchar();
while (c!= EOF){
  putchar(c);
   c = getchar();
}
```

10

Review of Example #1

- Character I/O
  - Including **stdio.h**
  - Functions **getchar()** and **putchar()**
  - Representation of a character as an integer
  - Predefined constant **EOF**
- Program control flow
  - The **for** and **while** statements
  - The **break** statement
  - The **return** statement
- Operators
  - Assignment operator: **=**
  - Increment operator: **++**
  - Relational operator to compare for equality: **==**
  - Relational operator to compare for inequality: **!=**

11

Example #2: Convert Uppercase

- Problem: Write a program to convert a file to all uppercase
  - Leave non-alphabetic characters alone
- Program design:

```
repeat
    Read a character
    If unsuccessful, break out of loop
    If the character is lower-case, convert to upper-
case
    Write the character
```

12

ASCII

American Standard Code for Information Interchange

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
16	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
32	SP	!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
48	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
64	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
80	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
96	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
112	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Lower case: 97-122 and upper case: 65-90
E.g., 'a' is 97 and 'A' is 65 (i.e., 32 apart)

13

Implementation in C

```
#include <stdio.h>
int main(void) {
    int c;
    for ( ; ; ) {
        c = getchar();
        if (c == EOF) break;
        if ((c >= 97) && (c < 123))
            c -= 32;
        putchar(c);
    }
    return 0;
}
```

14

It works!

- Submit
- Receive your grade with quiet confidence

15

It’s a …

B-

16

What? But it works …

- A good program is:
  - Clean
  - Readable
  - Maintainable
- It’s not enough that your program works!
  - We take this seriously in COS 217

Avoid Hard-coded Numbers

```
#include <stdio.h>
int main(void) {
   int c;
   for ( ; ; ) {
      c = getchar();
      if (c == EOF) break;
      if ((c >= 97) && (c < 123))
         c -= 32;
      putchar(c);
   }
   return 0;
}
```

Ugly.
Works for
ASCII only

Improvement: Character Constants

```
#include <stdio.h>
int main(void) {
   int c;
   for ( ; ; ) {
      c = getchar();
      if (c == EOF) break;
      if ((c >= 'a') && (c <= 'z'))
         c += 'A' - 'a';
      putchar(c);
   }
   return 0;
}
```

Better.
But still assumes that alphabetic character codes are contiguous

19

Improvement: Existing Functions

Standard C Library Functions ctype(3C)

Section 3C is for C library functions

NAME

ctype, isdigit, isxdigit, islower, isupper, isalpha, isalnum, isspace, iscntrl, ispunct, isprint, isgraph, isascii - character handling

SYNOPSIS

```
#include <ctype.h>
int isalpha(int c);
int isupper(int c);
int islower(int c);
int isdigit(int c);
int isalnum(int c);
int isspace(int c);
int ispunct(int c);
int isprint(int c);
int isgraph(int c);
int iscntrl(int c);
int toupper(int c);
int tolower(int c);
```

DESCRIPTION

These macros classify character-coded integer values. Each is a predicate returning non-zero for true, 0 for false...

The toupper() function has as a domain a type int, the value of which is representable as an unsigned char or the value of EOF.... If the argument of toupper() represents a lower-case letter ... the result is the corresponding upper-case letter. All other arguments in the domain are returned unchanged.

20

Using the ctype Functions

```
#include <stdio.h>
#include <ctype.h>
int main(void) {
   int c;
   for ( ; ; ) {
      c = getchar();
      if (c == EOF) break;
      if (islower(c))
         c = toupper(c);
      putchar(c);
   }
   return 0;
}
```

Returns non-zero (true) iff c is a lowercase character

Building and Running

```
% ls
upper.c
% gcc217 upper.c -o upper
% ls
upper upper.c
% upper
We'll be on time today!
WE'LL BE ON TIME TODAY!
^D
%
```

Run the Code on Itself

```
% upper < upper.c
#INCLUDE <STDIO.H>
#INCLUDE <CTYPE.H>
INT MAIN(VOID) {
   INT C;
   FOR ( ; ; ) {
      C = GETCHAR();
      IF (C == EOF) BREAK;
      IF (ISLOWER(C))
         C = TOUPPER(C);
      PUTCHAR(C);
   }
   RETURN 0;
}
```

23

Output Redirection

```
% upper < upper.c > junk.c
% gcc217 junk.c -o junk
test.c:1:2: invalid preprocessing directive #INCLUDE
test.c:2:2: invalid preprocessing directive #INCLUDE
test.c:3: syntax error before "MAIN"
etc...
```

24

Review of Example #2

- Representing characters
  - ASCII character set
  - Character constants (e.g., ‘A’ or ‘a’)
- Manipulating characters
  - Arithmetic on characters
  - Functions like `islower()` and `toupper()`
- Compiling and running C code
  - Compile to generate executable file
  - Invoke executable to run program
  - Can redirect stdin and/or stdout

25

Example #3: Capitalize First Letter

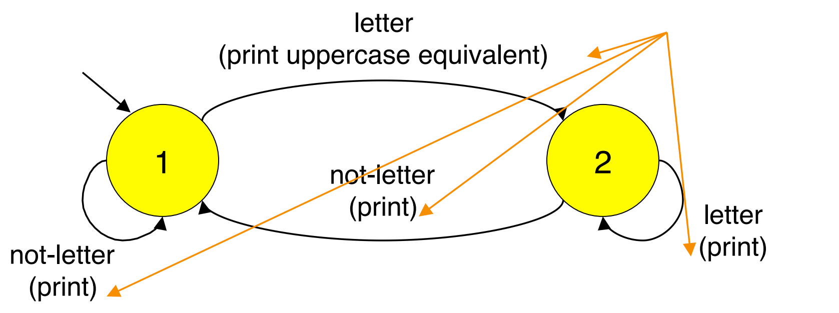
- Capitalize the first letter of each word
  - “cos 217 rocks” → “Cos 217 Rocks”
- Sequence through the string, one letter at a time
  - Print either the character, or the uppercase version
- Challenge: need to remember where you are
  - Capitalize “c” in “cos”, but not “o” in “cos” or “c” in “rocks”
- Solution: keep some extra information around
  - Whether you’ve encountered the first letter in the word
  - Same input letter can lead to different processing depending on this

26

Deterministic Finite Automaton

Deterministic Finite Automaton (DFA)

Actions are not part of DFA formalism; but they're helpful



- States
  - State 1: I've not encountered first letter in current word, i.e. I'm inside a word
  - State 2: I've encountered first letter in current word, i.e. I'm not inside a word
- Inputs
- Actions

27

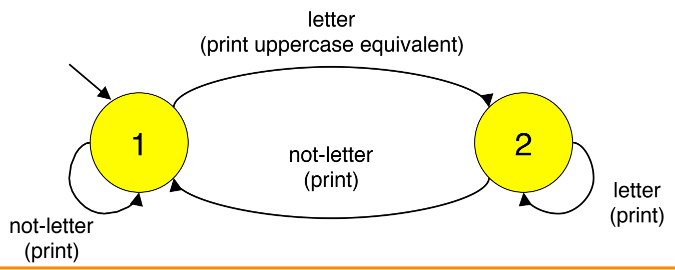
Implementation Skeleton

```
#include <stdio.h>
#include <ctype.h>
int main (void) {
   int c;
   for ( ; ; ) {
      c = getchar();
      if (c == EOF) break;
      <process one character>
   }
   return 0;
}
```

28

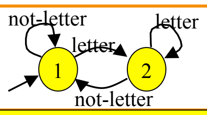
Implementation Skeleton

- Process one character:
  - Check current state
  - Check input character
  - Based on state and character, check DFA and execute:
    - transition to new state
    - the indicated action
    - Note: same input can lead to different actions



29

Implementation



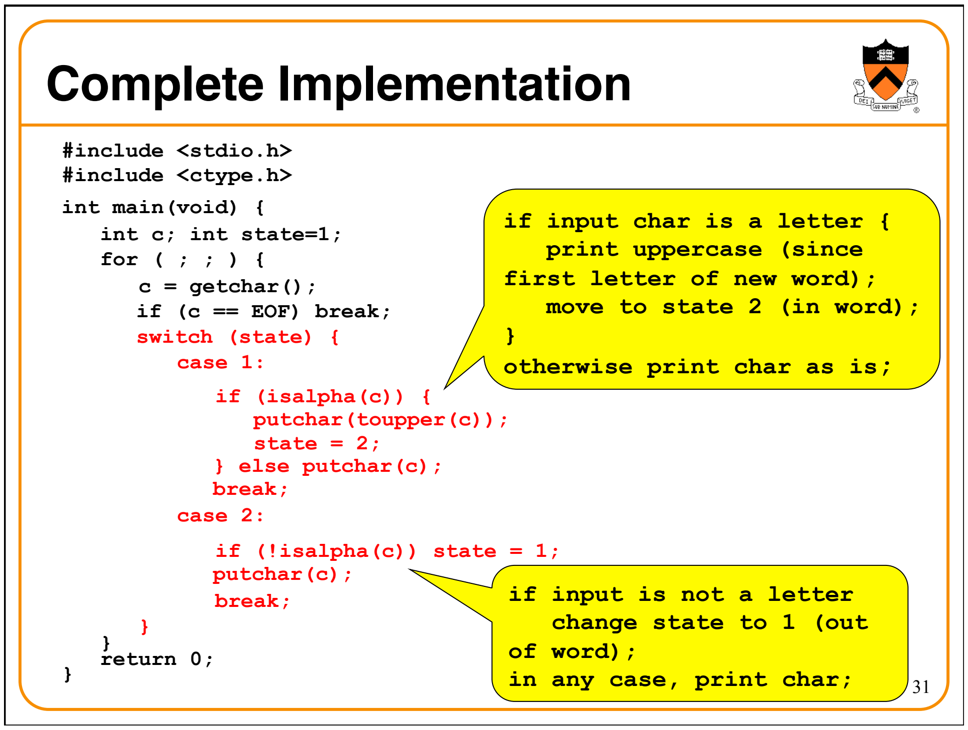
```
Process one character:

switch (state) {
    case 1:
        <state 1 input check and action>
        break;
    case 2:
        <state 2 input check and action>
        break;
    default:
        <this should never happen>
}
```

```
if input char is a letter {
   print uppercase (since
first letter of new word);
   move to state 2 (in word);
}
otherwise print char as is;
```

```
if input not a letter
   change state to 1 (out of
word);
in any case, print char as is;
```

30



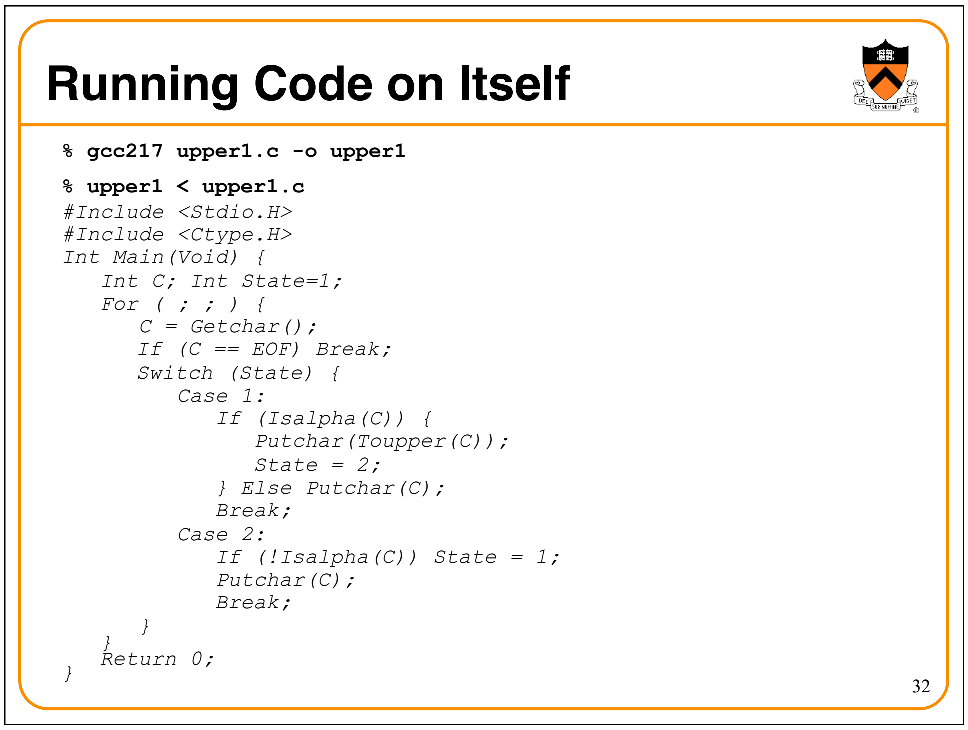
Complete Implementation

```
#include <stdio.h>
#include <ctype.h>
int main(void) {
   int c; int state=1;
   for ( ; ; ) {
      c = getchar();
      if (c == EOF) break;
      switch (state) {
         case 1:
            if (isalpha(c)) {
               putchar(toupper(c));
               state = 2;
            } else putchar(c);
            break;
         case 2:
            if (!isalpha(c)) state = 1;
            putchar(c);
            break;
      }
   }
   return 0;
}
```

```
if input char is a letter {
   print uppercase (since
first letter of new word);
   move to state 2 (in word);
}
otherwise print char as is;
```

```
if input is not a letter
   change state to 1 (out
of word);
in any case, print char;
```

31



Running Code on Itself

```
% gcc217 upper1.c -o upper1
% upper1 < upper1.c
#Include <Stdio.H>
#Include <Ctype.H>
Int Main(Void) {
   Int C; Int State=1;
   For ( ; ; ) {
      C = Getchar();
      If (C == EOF) Break;
      Switch (State) {
         Case 1:
            If (Isalpha(C)) {
               Putchar(Toupper(C));
               State = 2;
            } Else Putchar(C);
            Break;
         Case 2:
            If (!Isalpha(C)) State = 1;
            Putchar(C);
            Break;
      }
   }
   Return 0;
}
```

32

It works!

- Submit
- What did I get? What did I get?

Your grade

B

OK, That's a B

- Works correctly, but
  - Mysterious integer constants ("magic numbers")
- What now?
  - States should have names, not just 1, 2

35

Improvement: Names for States

- Define your own named constants

```
enum Statetype {NOT_IN_WORD,IN_WORD};
```

  - Define an enumeration type

```
enum Statetype state;
```

  - Define a variable of that type

36

Improvement: Names for States

```
#include <stdio.h>
#include <ctype.h>
enum Statetype {NOT_IN_WORD,IN_WORD};
int main(void) {
   int c; enum Statetype state = NOT_IN_WORD;
   for ( ; ; ) {
      c = getchar();
      if (c == EOF) break;
      switch (state) {
         case NOT_IN_WORD:
            if (isalpha(c)) {
               putchar(toupper(c));
               state = IN_WORD;
            } else putchar(c);
            break;
         case IN_WORD:
            if (!isalpha(c)) state = NOT_IN_WORD;
            putchar(c);
            break;
      }
   }
   return 0;
}
```

37

It still works, no magic constants

- Submit
- Can I have my A+ please? I have a party to go to.

38

Ask and you shall not receive …

B+

39

Huh?

- Works correctly, but
  - No modularity
- What now?
  - Should handle each state in a separate function
  - Each state handling function does the work for a given state, including reading the input and taking the action
  - It returns the new state, which we will store in the "state" variable for the next iteration of our infinite loop

40

Improvement: Modularity

```
#include <stdio.h>
#include <ctype.h>
enum Statetype {NORMAL,INWORD};
enum Statetype handleNotInwordState(int c) {...}
enum Statetype handleInwordState(int c) {...}
int main(void) {
   int c;
   enum Statetype state = NORMAL;
   for ( ; ; ) {
      c = getchar();
      if (c == EOF) break;
      switch (state) {
         case NORMAL:
            state = handleNotInwordState(c);
            break;
         case INWORD:
            state = handleInwordState(c);
            break;
      }
   }
   return 0;
}
```

41

Improvement: Modularity

```
enum Statetype handleNotInwordState(int c) {
   enum Statetype state;
   if (isalpha(c)) {
      putchar(toupper(c));
      state = IN_WORD;
   }
   else {
      putchar(c);
      state = NOT_IN_WORD;
   }
   return state;
}
```

42

Improvement: Modularity

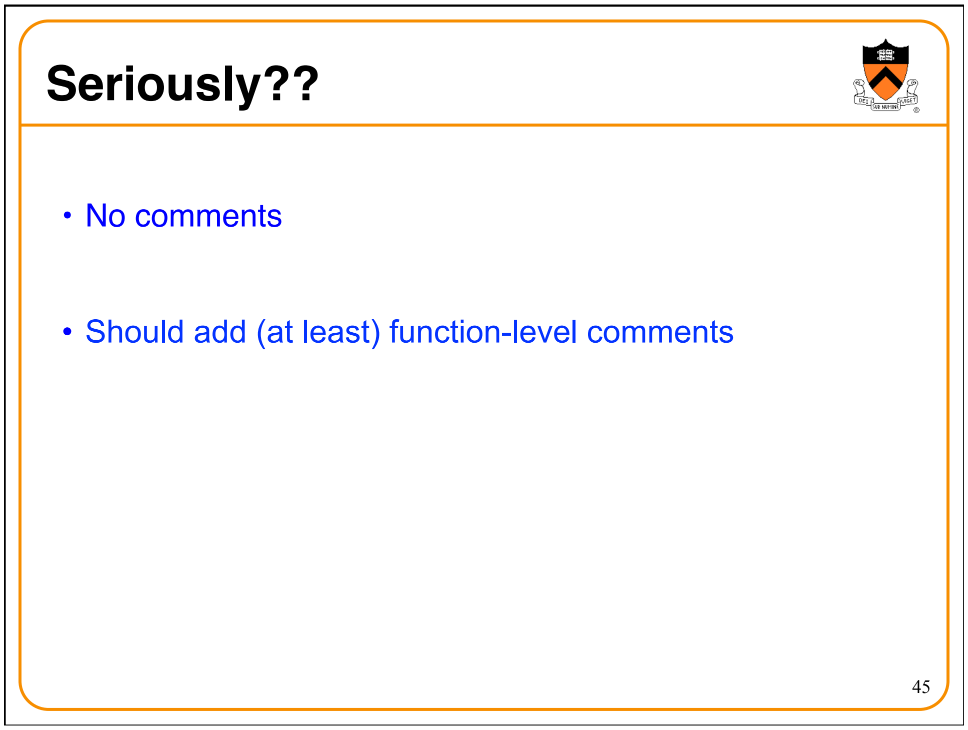
```
enum Statetype handleInwordState(int c) {
   enum Statetype state;
   putchar(c);
   if (!isalpha(c))
      state = NOT_IN_WORD;
   else
      state = IN_WORD;
   return state;
}
```

43

It's a thing of beauty …

A-

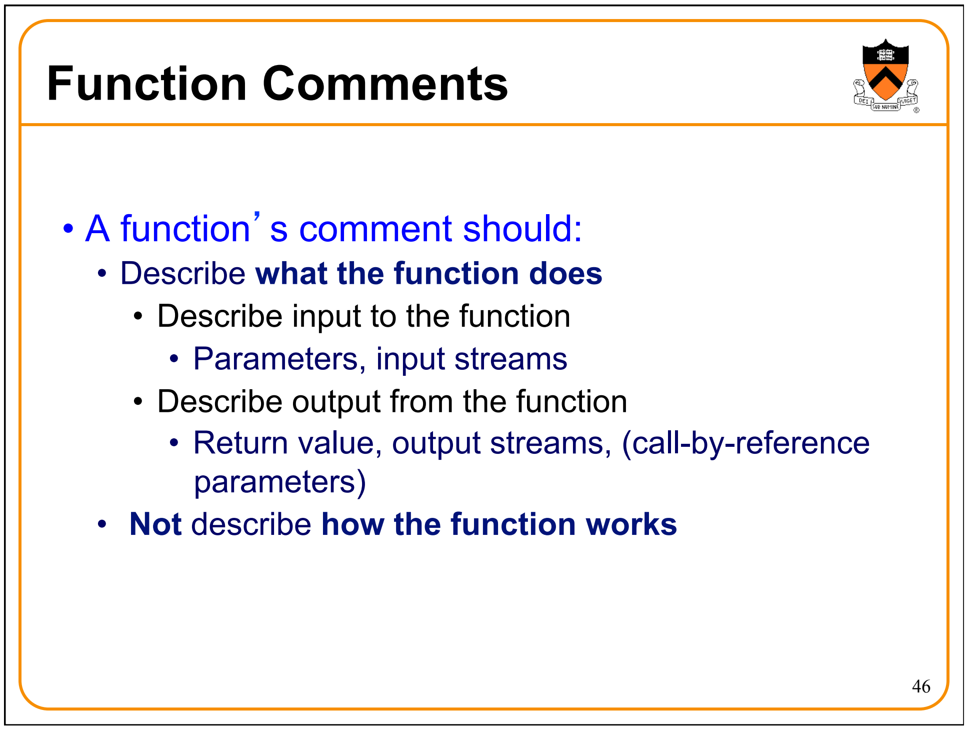
44



Seriously??

- No comments
- Should add (at least) function-level comments

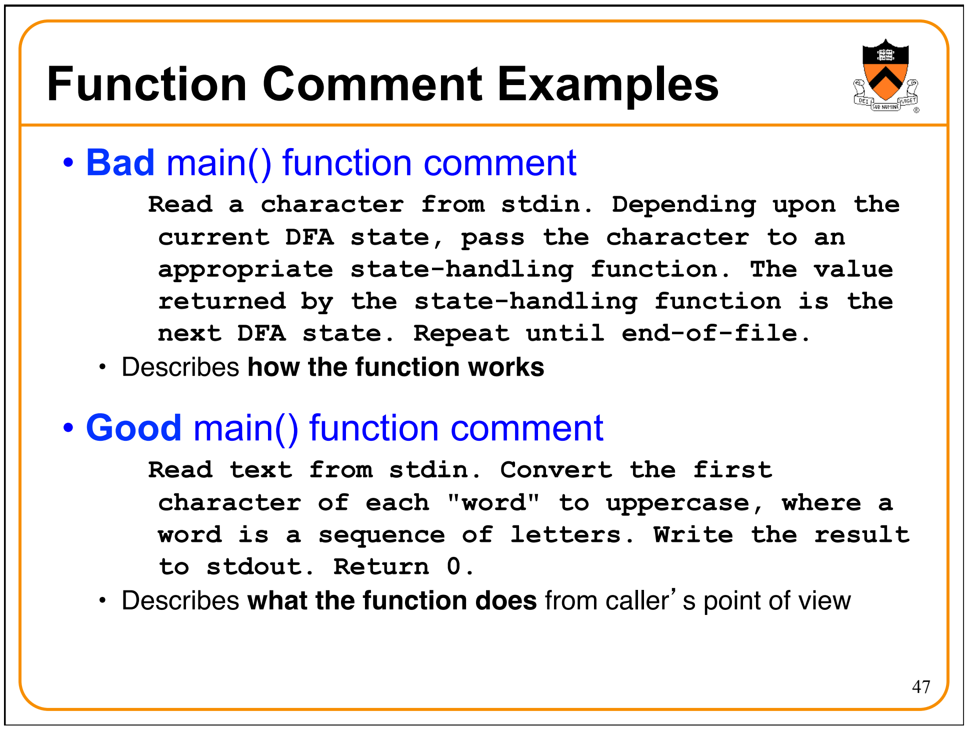
45



Function Comments

- A function's comment should:
  - Describe **what the function does**
    - Describe input to the function
      - Parameters, input streams
    - Describe output from the function
      - Return value, output streams, (call-by-reference parameters)
  - **Not** describe **how the function works**

46



Function Comment Examples

- **Bad** main() function comment

  ```
  Read a character from stdin. Depending upon the
   current DFA state, pass the character to an
   appropriate state-handling function. The value
   returned by the state-handling function is the
   next DFA state. Repeat until end-of-file.
  ```

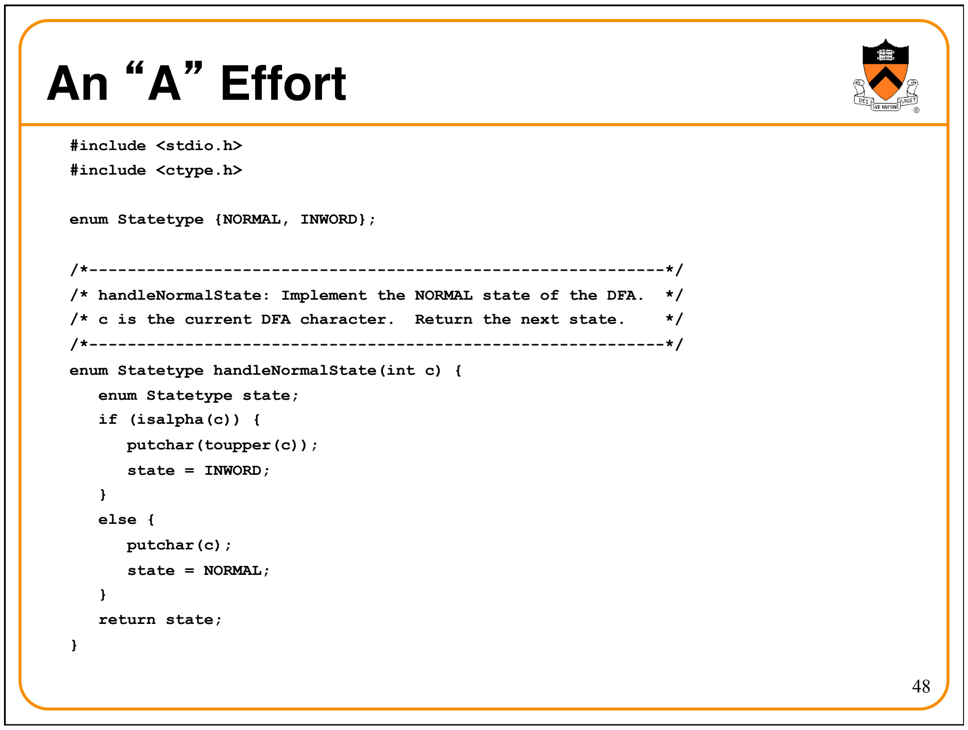
  - Describes **how the function works**

- **Good** main() function comment

  ```
  Read text from stdin. Convert the first
   character of each "word" to uppercase, where a
   word is a sequence of letters. Write the result
   to stdout. Return 0.
  ```

  - Describes **what the function does** from caller's point of view

47



An "A" Effort

```
#include <stdio.h>
#include <ctype.h>

enum Statetype {NORMAL, INWORD};

/*------------------------------------------------------------*/
/* handleNormalState: Implement the NORMAL state of the DFA.  */
/* c is the current DFA character.  Return the next state.    */
/*------------------------------------------------------------*/
enum Statetype handleNormalState(int c) {
   enum Statetype state;
   if (isalpha(c)) {
      putchar(toupper(c));
      state = INWORD;
   }
   else {
      putchar(c);
      state = NORMAL;
   }
   return state;
}
```

48

An "A" Effort

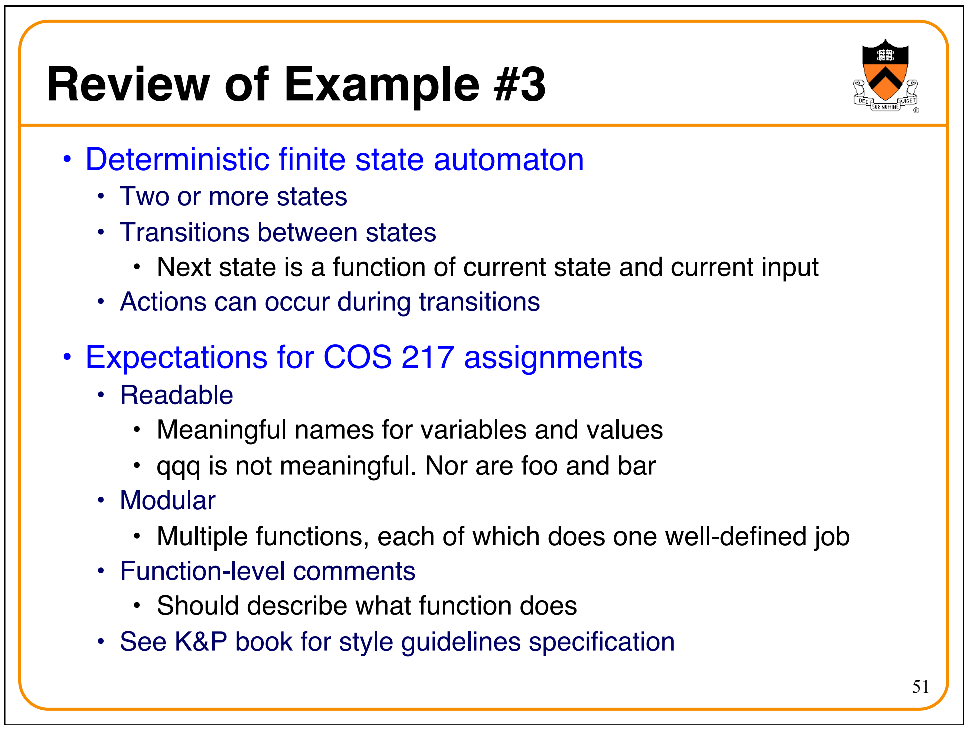
```
/*--------------------------------------------------------------*/
/* handleInwordState: Implement the INWORD state of the DFA.  */
/* c is the current DFA character.  Return the next state.    */
/*--------------------------------------------------------------*/
enum Statetype handleInwordState(int c) {
   enum Statetype state;
   putchar(c);
   if (!isalpha(c))
      state = NORMAL;
   else
      state = INWORD;
   return state;
}
```

49

An "A" Effort

```
/*------------------------------------------------------------*/
/* main: Read text from stdin. Convert the first character    */
/* of each "word" to uppercase, where a word is a sequence of */
/* letters. Write the result to stdout. Return 0.             */
/*------------------------------------------------------------*/
int main(void) {
   int c;
   enum Statetype state = NORMAL;
   /* Use a DFA approach.  state indicates the state of the DFA. */
   for ( ; ; ) {
      c = getchar();
      if (c == EOF) break;
      switch (state) {
         case NORMAL:
            state = handleNormalState(c);
            break;
         case INWORD:
            state = handleInwordState(c);
            break;
      }
   }
   return 0;
}
```

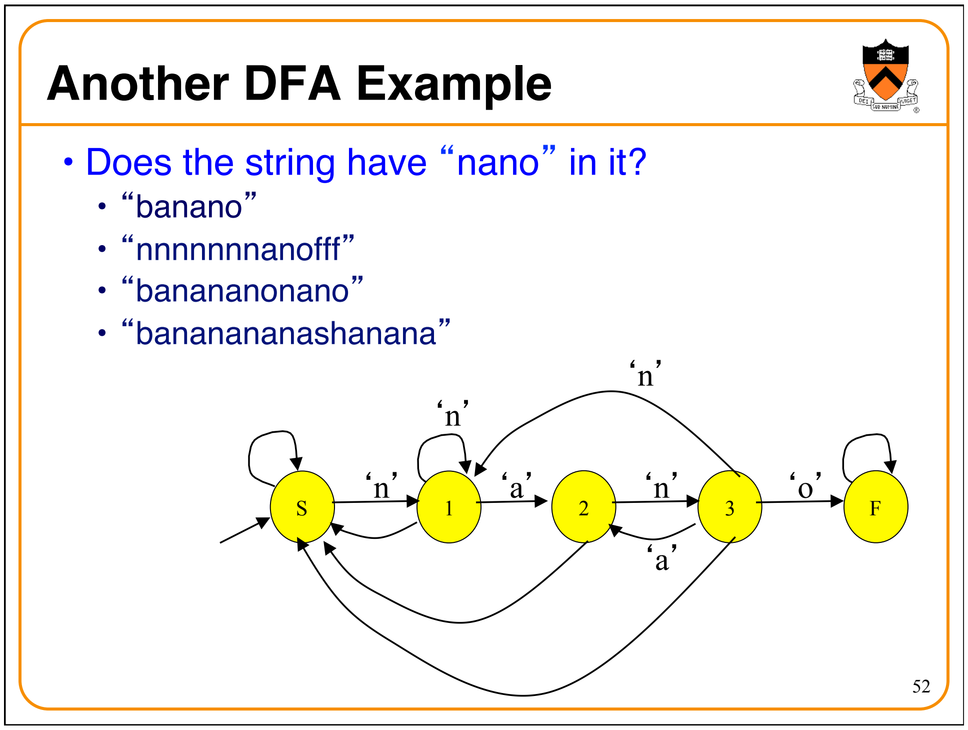
50



Review of Example #3

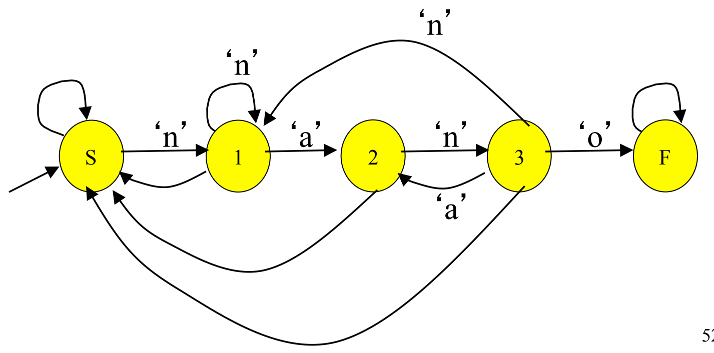
- Deterministic finite state automaton
  - Two or more states
  - Transitions between states
    - Next state is a function of current state and current input
  - Actions can occur during transitions
- Expectations for COS 217 assignments
  - Readable
    - Meaningful names for variables and values
    - qqq is not meaningful. Nor are foo and bar
  - Modular
    - Multiple functions, each of which does one well-defined job
  - Function-level comments
    - Should describe what function does
  - See K&P book for style guidelines specification

51



Another DFA Example

- Does the string have “nano” in it?
  - “banano”
  - “nnnnnnnanofff”
  - “banananonano”
  - “bananananashanana”



52

Yet Another DFA Example

Question #4 from fall 2005 midterm
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"

53

Summary

- Examples illustrating C
  - Overall program structure
  - Control statements (**if**, **while**, **for**, and **switch**)
  - Character input/output (**getchar()** and **putchar()**)
- Deterministic finite state automata (i.e., state machines)
- Expectations for programming assignments

54