EE209: C Examples

Reminder

- Sign up for ee209 mailing list
  - If you haven't received any email from ee209 yet
  - Follow the link from our class homepage
- Sign up for our class Moodle
- Precept
  - First: 7:00-8:15pm, 9/7 Wed
  - Creative Learning building 411

Goals of this Lecture

- Help you learn about:
  - The fundamentals of C
    - Overall program structure, control statements, character I/O functions
  - 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 (e.g. 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.)

- Program design (cont.)
  - 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)

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?

Read and Write Ten Characters

```
#include <stdio.h>

int main(void)
{
  int c, i;

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

  return 0;
}
```

- 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

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

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;
}
```

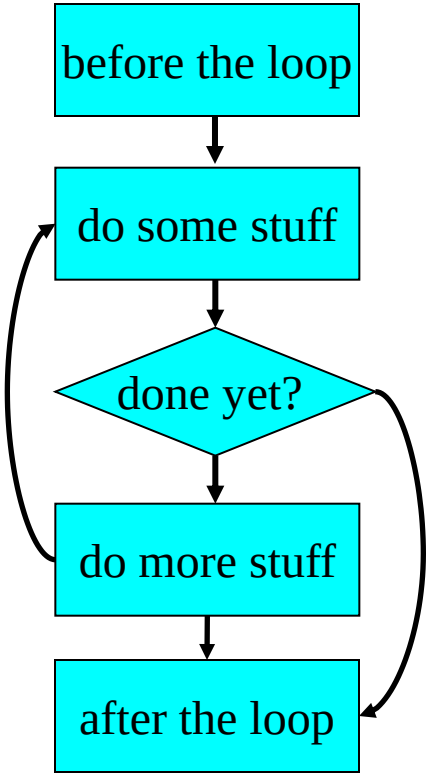
When will this be executed?

How would you terminate this program?

Read and Write Until End-Of-File

- Test for end-of-file
  - **EOF** is a global constant, defined in stdio.h (#define EOF (-1))
  - 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;
}
```



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();
}
```

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: **`!=`**

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
```

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)

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;
}
```

That's a B-minus

- A good program is:
  - Clean
  - Readable
  - Maintainable
- It's not enough that your program works!
  - We take this seriously in EE 209

Avoid Mysterious 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;
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 assumes that alphabetic character codes are contiguous

Improvement: Existing Functions

Standard C Library Functions ctype(3C)

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.

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
% gcc209 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;
}
```

Output Redirection

```
% upper < upper.c > junk.c
% gcc209 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...
```

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

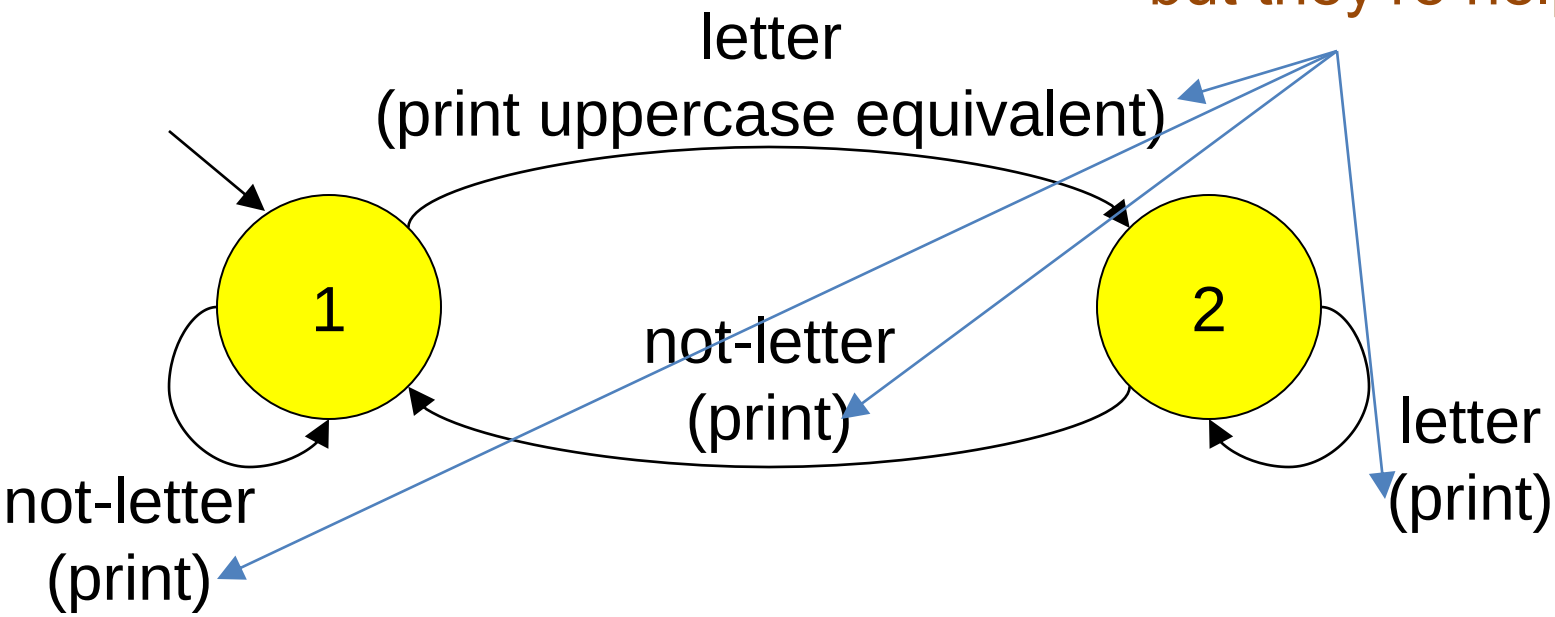
Example #3: Capitalize First Letter

- Capitalize the first letter of each word
  - "ee 209 is fun" → "Ee 209 Is Fun"
- 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 "e" in "ee", but not "o" or "c" in "rocks"
- Solution: keep some extra information around
  - Whether you've encountered the first letter in the word

Deterministic Finite Automaton

Deterministic Finite Automaton (DFA)

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

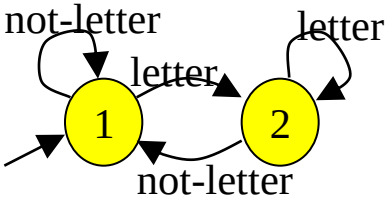


- States
- Transitions labeled by characters (or categories)
- Optionally, transitions labeled by actions

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;
}
```

Implementation




```
<process one character> =
switch (state)
{
    case 1:
        <state 1 action>
        break;

    case 2:
        <state 2 action>
        break;

    default:
        <this should never happen>
}
```

```
if (isalpha(c)) {
    putchar(toupper(c));
    state = 2;
}
else putchar(c);
```

```
if (!isalpha(c))
    state = 1;
putchar(c);
```

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;
}
```

Running Code on Itself

```
% gcc209 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;
}
```

OK, That's a B

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

Improvement: Names for States

- Define your own named constants

```
enum Statetype {NORMAL,INWORD};
```

  – Define an enumeration type

```
enum Statetype state;
```

  – Define a variable of that type

Improvement: Names for States

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

OK, That's a B+

- Works correctly, but
  - No modularity
- What now?
  - Should handle each state in a separate function

Improvement: Modularity

```
#include <stdio.h>
#include <ctype.h>
enum Statetype {NORMAL,INWORD};
enum Statetype handleNormalState(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 = handleNormalState(c);
            break;
         case INWORD:
            state = handleInwordState(c);
            break;
      }
   }
   return 0;
}
```

Improvement: Modularity

```
enum Statetype handleNormalState(int c)
{
   enum Statetype state;
   if (isalpha(c)) {
      putchar(toupper(c));
      state = INWORD;
   }
   else {
      putchar(c);
      state = NORMAL;
   }
   return state;
}
```

Improvement: Modularity

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

OK, That's an A-

- Works correctly, but
  - No comments
- What now?
  - Should add (at least) function-level comments

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
  - **Not** describe **how the function works**

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

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;
}
```

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;
}
```

An “A” Effort

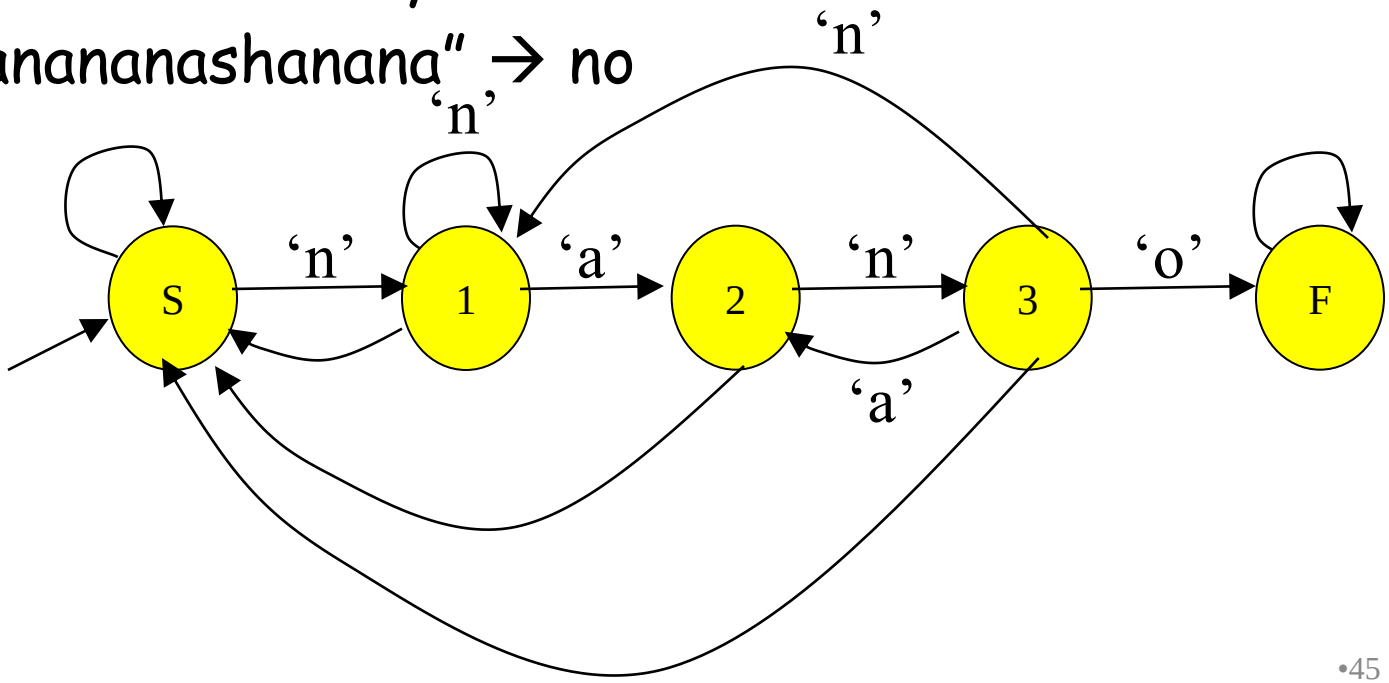
```
/*-----------------------------------------------------------*/
/* main: Read text from stdin. Convert the first character    */
/* of each "word" to uppercase, where a word is a sequence of */
/* letters. Write the result to stdout. Return 0.             */
/*-----------------------------------------------------------*/
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;
}
```

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 character
  - Actions can occur during transitions
- Expectations for EE 209 assignments
  - Readable
    - Meaningful names for variables and values
  - 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

Another DFA Example

- Does the string have "nano" in it?
  - "banano" → yes
  - "nnnnnnnanofff" → yes
  - "bananonano" → yes
  - "bananananashanana" → no



Yet Another DFA Example

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"

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