

COS 217: Introduction to Programming Systems

Program Design Decisions
&
C Language Design (Logical Data)



PRINCETON UNIVERSITY



Agenda

Simple C Programs

- charcount
 - character I/O
- upper (ctype library)
 - portability concerns
 - char details
- upper1 (switch statements, enums, functions)
 - internal documentation (i.e., comments)

Two big differences from Java

- Variable declarations
- Logical operators



Recall: The charcount Program

The program:

`charcount.c`

```
#include <stdio.h>
/* Write to stdout the number of
   chars in stdin. Return 0. */
int main(void) {
    int c;
    int charCount = 0;
    c = getchar();
    while (c != EOF) {
        charCount++;
        c = getchar();
    }
    printf("%d\n", charCount);
    return 0;
}
```



stdio.h Features (excerpts)

```
$ man stdio.h
NAME
    stdio.h -- standard buffered input/output

SYNOPSIS
    #include <stdio.h>

DESCRIPTION
    The <stdio.h> header shall define the following data types through typedef:
        FILE          A structure containing information about a file.
        size_t        As described in <stddef.h>.

    The <stdio.h> header shall define the following macro which shall expand to an
    integer constant expression with type int and a negative value:
        EOF           End-of-file return value.

    The <stdio.h> header shall define the following macros which shall expand to
    expressions of type ``pointer to FILE'' that point to the FILE objects associated,
    respectively, with the standard error, input, and output streams:
        stderr        Standard error output stream.
        stdin         Standard input stream.
        stdout        Standard output stream.
```



stdio.h Features (excerpts, continued)

```
$ man stdio.h
```

```
...
```

The following shall be declared as functions and may also be defined as macros.
Function prototypes shall be provided.

```
int    fclose(FILE *);
int    feof(FILE *);
int    fflush(FILE *);
int    fgetc(FILE *);
FILE   *fopen(const char *restrict, const char *restrict);
int    fprintf(FILE *restrict, const char *restrict, ...);
int    fscanf(FILE *restrict, const char *restrict, ...);
int    getc(FILE *);
int    getchar(void);
int    printf(const char *restrict, ...);
int    putc(int, FILE *);
int    putchar(int);
int    scanf(const char *restrict, ...);
```



Character Input/Output (I/O) in C

Design of C:

- Does not provide I/O facilities in the language
- Instead provides I/O facilities in standard library, declared in `stdio.h`
 - Constant: EOF
 - Data type: FILE (described later in course)
 - Variables: `stdin`, `stdout`, and `stderr`
 - Functions: ...

Reading characters

- `getchar()` function with return type wider than char (specifically, int)
- Returns EOF (a special non-character int) to indicate failure
- **Reminder: there is no such thing as "the EOF character"**

Writing characters

- `putchar()` function accepting one parameter
- For symmetry with `getchar()`, parameter is an int



iClicker Question



Q: There are other ways to `charcount` – which is best?

A.

```
for (c = getchar(); c != EOF; c = getchar())  
    charCount++;
```

B.

```
while ((c = getchar()) != EOF)  
    charCount++;
```

C.

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

D.

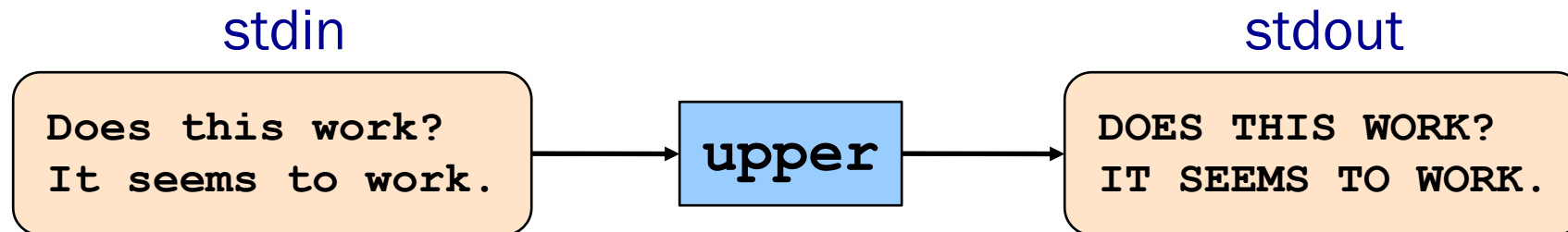
```
c = getchar();  
while (c != EOF)  
{   charCount++;  
    c = getchar();  
}
```



Recall: The upper Program

Functionality

- Read all chars from stdin
- Convert each lower-case alphabetic char to upper case
 - Leave other kinds of chars alone
- Write result to stdout





The C char Data Type

char is 1 byte – designed to hold a single character, but used for more

Mapping from char values to characters on pretty much all machines:

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									HT	LF					
16																
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	{		}	~	

Notes: Many non-printing characters left blank in table above

Lower-case and upper-case letters are 32 apart



upper Version 1

```
#include <stdio.h>
int main(void)
{
    int c;
    while ((c = getchar()) != EOF) {
        if ((c >= 97) && (c <= 122))
            c -= 32;
        putchar(c);
    }
    return 0;
}
```

What's wrong?

EBCDIC



Extended Binary Coded Decimal Interchange Code

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	NUL					HT										
16																
32						LF										
48																
64	SP											.	<	(+	
80	&										!	\$	*)	;	
96	-	/										,	%	-	>	?
112									`		:	#	@	'	=	"
128		a	b	c	d	e	f	g	h	i		{				
144		j	k	l	m	n	o	p	q	r		}				
160		~	s	t	u	v	w	x	y	z						
176																
192		A	B	C	D	E	F	G	H	I						
208		J	K	L	M	N	O	P	Q	R						
224	\		S	T	U	V	W	X	Y	Z						
240	0	1	2	3	4	5	6	7	8	9						

Partial map



Character Literals

Single quote syntax: 'a' is a value of type char with the value 97

Use backslash to write special characters

- Examples (with numeric equivalents in ASCII, EBCDIC):

'a'	the a character	(97, 129)
'A'	the A character	(65, 193)
'0'	the zero character	(48, 240)
'\0'	the NUL (nullbyte) character	(0, 0)
'\n'	the newline character	(10, 37)
'\t'	the horizontal tab character	(9, 5)
'\\'	the backslash character	(92, 224)
'\''	the single quote character	(39, 125)
'\"'	the double quote character	(34, 127)

upper Version 2



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

Arithmetic
on chars?

What's wrong now?

EBCDIC



Extended Binary Coded Decimal Interchange Code

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	NUL					HT										
16																
32						LF										
48																
64	SP											.	<	(+	
80	&										!	\$	*)	;	
96	-	/										,	%	_	>	?
112									`		:	#	@	'	=	"
128		a	b	c	d	e	f	g	h	i		{				
144		j	k	l	m	n	o	p	q	r		}				
160		~	s	t	u	v	w	x	y	z						
176																
192		A	B	C	D	E	F	G	H	I						
208		J	K	L	M	N	O	P	Q	R						
224	\		S	T	U	V	W	X	Y	Z						
240	0	1	2	3	4	5	6	7	8	9						

Partial map

Note: Lower case not contiguous; same for upper case

upper Version 3



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



iClicker Question



Q: Is the if statement really necessary?

A. Gee, I don't know.
Let me check
the man page
(again)!

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


ctype.h Functions



```
$ man toupper
```

NAME

```
toupper, tolower - convert letter to upper or lower case
```

SYNOPSIS

```
#include <ctype.h>
int toupper(int c);
int tolower(int c);
```

DESCRIPTION

```
toupper() converts the letter c to upper case, if possible.
tolower() converts the letter c to lower case, if possible.
```

```
If c is not an unsigned char value, or EOF, the behavior of
these functions is undefined.
```

RETURN VALUE

```
The value returned is that of the converted letter,
or c if the conversion was not possible.
```



iClicker Question



Q: Is the if statement really necessary?

- A. Yes, necessary for correctness.
- B. Not necessary, but I'd leave it in.
- C. Not necessary, and I'd get rid of it.

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

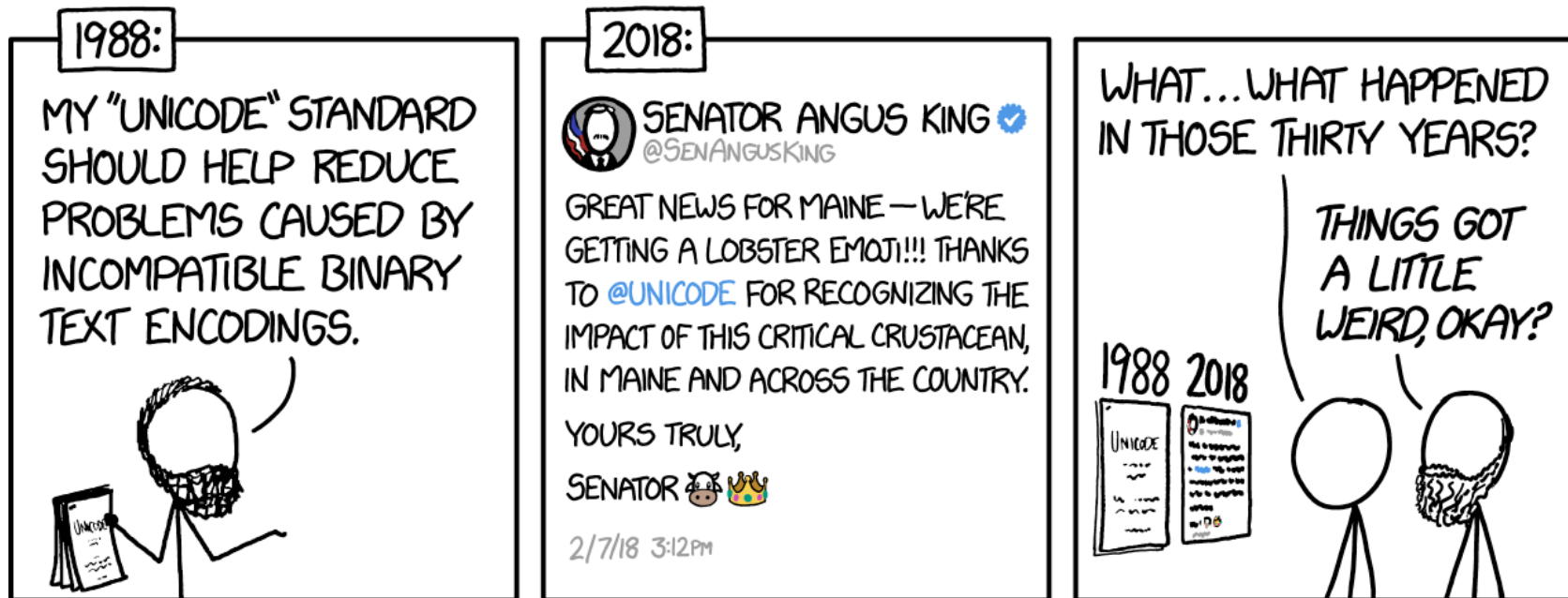



Modern Unicode

When C was designed, characters fit into 8 (really 7) bits, so C's chars are 8 bits long.

When Java was designed, Unicode fit into 16 bits, so Java's chars are 16 bits long.

Then this happened:



<https://xkcd.com/1953/>

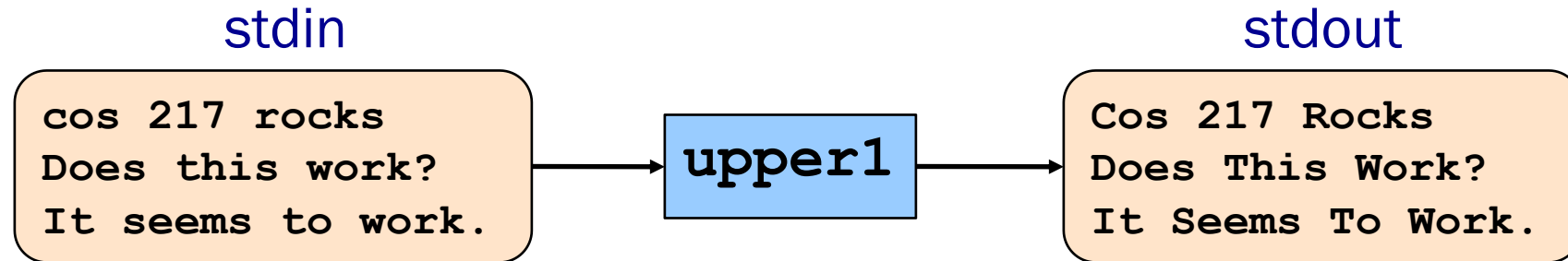
Result: modern systems use *variable length* (UTF-8/16/32) encoding for Unicode.



Recall: The upper1 Program

Functionality

- Read all chars from stdin
- Capitalize the first letter of each word
 - “cos 217 rocks” ⇒ “Cos 217 Rocks”
- Write result to stdout



What we need: maintain extra information, namely “in a word” vs “*not* in a word”

- Need systematic way of reasoning about what to do with that information

upper1 Version 3



```
#include <stdio.h>
#include <ctype.h>
enum Statetype {NORMAL, INWORD};

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

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

```
int main(void)
{
    int c;
    enum Statetype state = NORMAL;
    while ((c = getchar()) != EOF) {
        switch (state) {
            case NORMAL:
                state = handleNormalState(c);
                break;
            case INWORD:
                state = handleInwordState(c);
                break;
        }
    }
    return 0;
}
```

That's an A-.
What's wrong?



upper1 Toward Final Version

Problem:

- The program works, but...
- No comments

Solution:

- Add (at least) function-level comments



Function Comments

Function comment should describe

what the function does (from the caller's viewpoint)

- Input to the function
 - Parameters, input streams
- Output from the function
 - Return value, output streams, (call-by-reference parameters)

Function comment should **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 characters. Write the result to stdout. Return 0.
```

Describes what the function does
(from caller's viewpoint)



upper1 Final Comments

```
/* defines constants representing each state in the DFA */  
enum Statetype {NORMAL, INWORD};
```

```
/* Implement the NORMAL state of the DFA. c is the current  
DFA character. Write c or its uppercase equivalent to  
stdout, as specified by the DFA. Return the next state. */  
  
enum Statetype handleNormalState(int c) {
```

```
/* Implement the INWORD state of the DFA. c is the current  
DFA character. Write c to stdout, as specified by the DFA.  
Return the next state. */  
  
enum Statetype handleInwordState(int c) {
```

```
/* 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) {  
    /* Use a DFA approach. state indicates the DFA state. */  
    enum Statetype state = NORMAL;
```



Agenda

Simple C Programs

- charcount
 - character I/O
- upper (ctype library)
 - portability concerns
 - char details
- upper1 (switch statements, enums, functions)
 - internal documentation (i.e., comments)

Language Design: Two big differences from Java

- Variable declarations
- Logical operators



Declaring Variables

C requires variable declarations.

Motivation:

- Declaring variables allows compiler to check “spelling”
- Declaring variables allows compiler to allocate memory more efficiently
- Declaring variables’ types produces fewer surprises at runtime
- Declaring variables requires more from the programmer
 - Extra verbiage
 - Type foresight
 - “Do what I mean, not what I say”



Declaring Variables

C requires variable declarations.

- Declaration statement specifies type of variable (and other attributes too)

Examples:

```
int i;  
int i, j;  
int i = 5;  
const int i = 5; /* value of i cannot change */  
static int i; /* covered later in course */  
extern int i; /* covered later in course */
```



Declaring Variables

C requires variable declarations.

- Declaration statement specifies type of variable (and other attributes too)
- **Unlike Java**, declaration statements in C89 must appear **before** any other kind of statement in compound statement

```
{
    int i;
    /* Non-declaration
       stmts that use i. */
    ...
    int j;
    /* Non-declaration
       stmts that use j. */
    ...
}
```

Illegal in C89

```
{
    int i;
    int j;
    /* Non-declaration
       stmts that use i. */
    ...
    /* Non-declaration
       stmts that use j. */
    ...
}
```

Legal in C89



Agenda

Simple C Programs

- upper (character data and I/O, ctype library)
 - portability concerns
- upper1 (switch statements, enums, functions)
 - DFA program design

Two big differences from Java

- Variable declarations
- Logical operators



Logical Data Types

- No separate logical or Boolean data type
- Represent logical data using type char or int
 - Or any primitive type! :/
- Conventions:
 - Statements (if, while, etc.) use $0 \Rightarrow \text{FALSE}$, $\neq 0 \Rightarrow \text{TRUE}$
 - Relational operators ($<$, $>$, etc.) and logical operators ($!$, $\&\&$, $||$) produce the result 0 or 1



[@lunarts](#)



Logical Data Type Shortcuts

Using integers to represent logical data permits shortcuts

```
...  
int i;  
...  
if (i) /* same as (i != 0) */  
    statement1;  
else  
    statement2;  
...
```

It also permits some really bad code...

```
i = (1 != 2) + (3 > 4);
```



iClicker Brainteaser



Q: What is `i` set to in the following code?

```
i = (1 != 2) + (3 > 4);
```

- A. 0
- B. 1
- C. 2
- D. 3
- E. 4



Logical Data Type Dangers

Beware: the following code will cause loss of sleep

```
...  
int i;  
...  
i = 0;  
...  
if (i = 5)  
    statement1;  
...
```

What happens
in Java?

What happens
in C?



Next time ... numbers! (Bigger than 127.)

