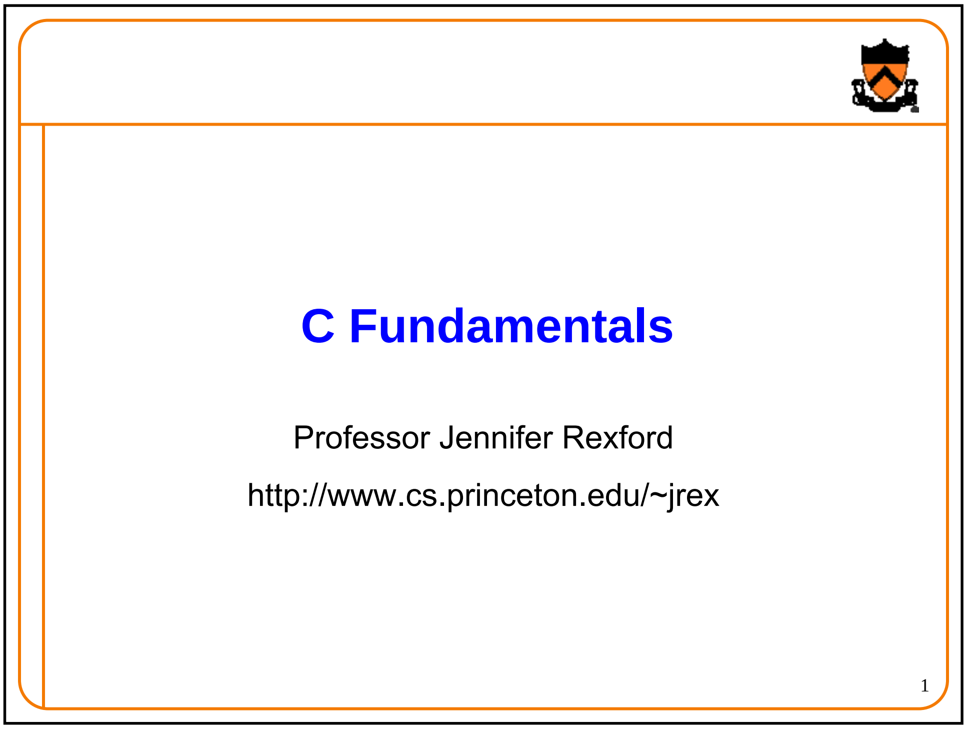
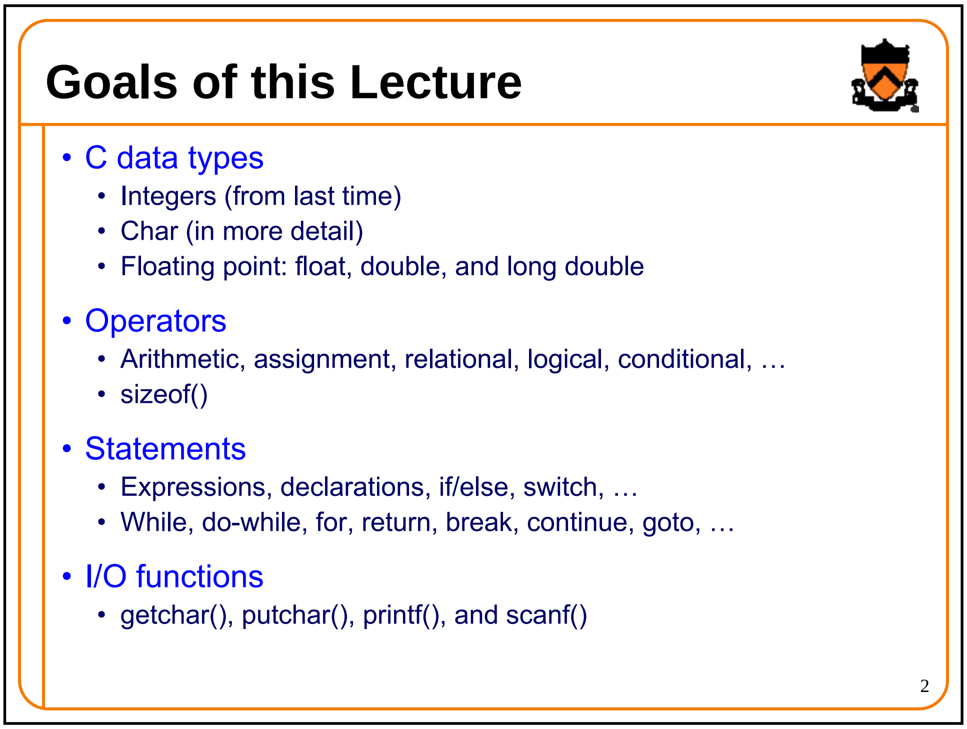


C Fundamentals

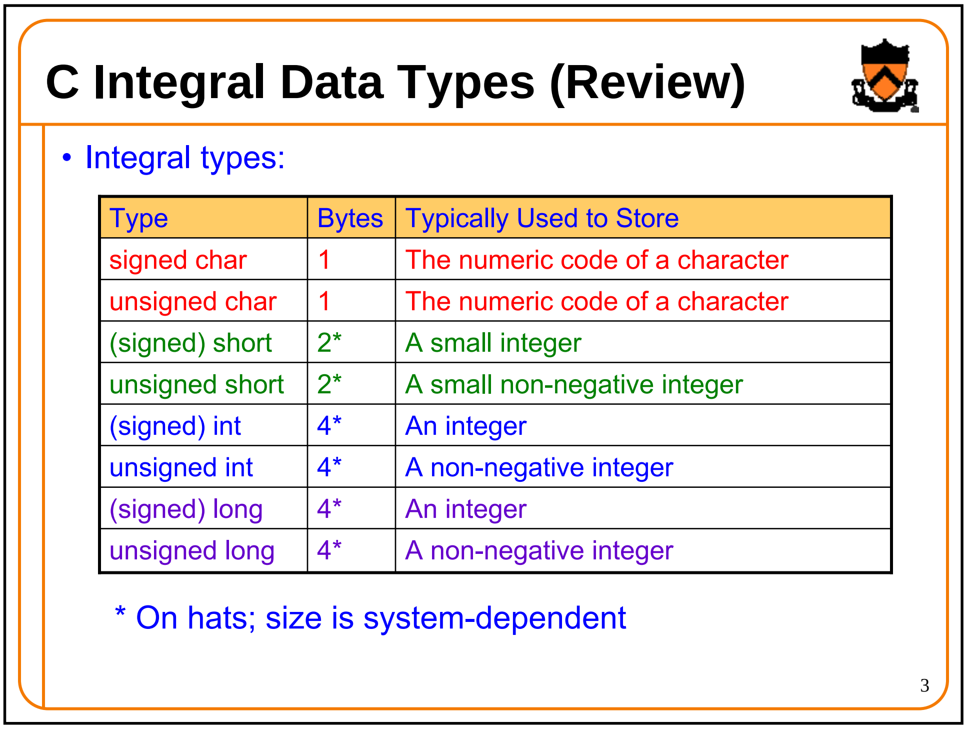
Professor Jennifer Rexford

http://www.cs.princeton.edu/~jrex



Goals of this Lecture

- C data types
  - Integers (from last time)
  - Char (in more detail)
  - Floating point: float, double, and long double
- Operators
  - Arithmetic, assignment, relational, logical, conditional, …
  - sizeof()
- Statements
  - Expressions, declarations, if/else, switch, …
  - While, do-while, for, return, break, continue, goto, …
- I/O functions
  - getchar(), putchar(), printf(), and scanf()

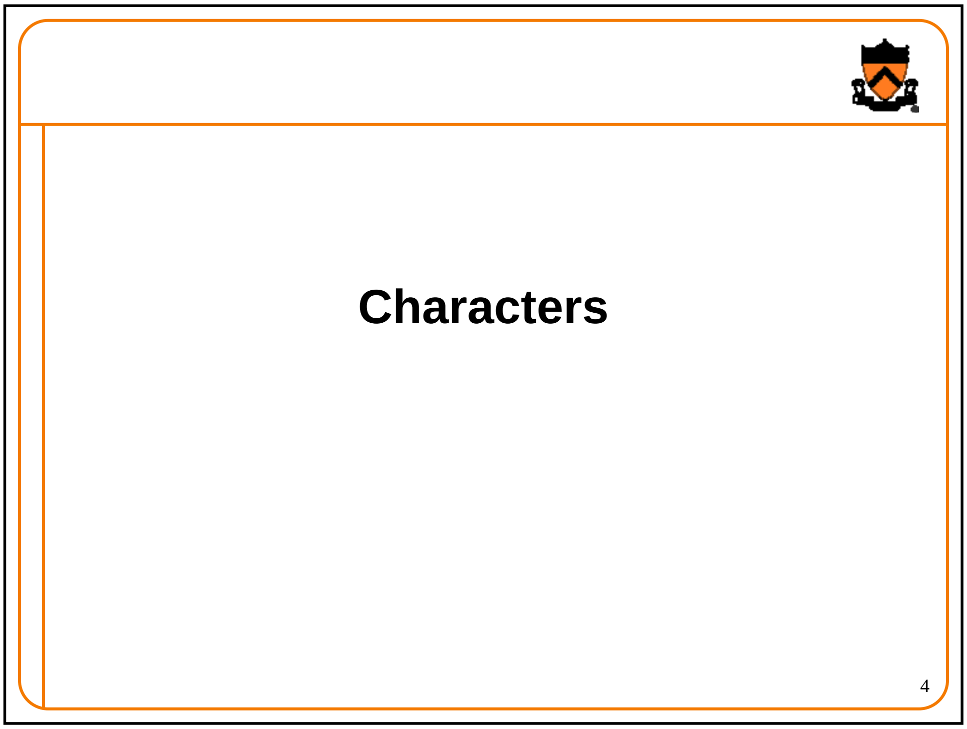


C Integral Data Types (Review)

- Integral types:

Type	Bytes	Typically Used to Store
signed char	1	The numeric code of a character
unsigned char	1	The numeric code of a character
(signed) short	2*	A small integer
unsigned short	2*	A small non-negative integer
(signed) int	4*	An integer
unsigned int	4*	A non-negative integer
(signed) long	4*	An integer
unsigned long	4*	A non-negative integer

* On hats; size is system-dependent



Characters

Using char for Characters

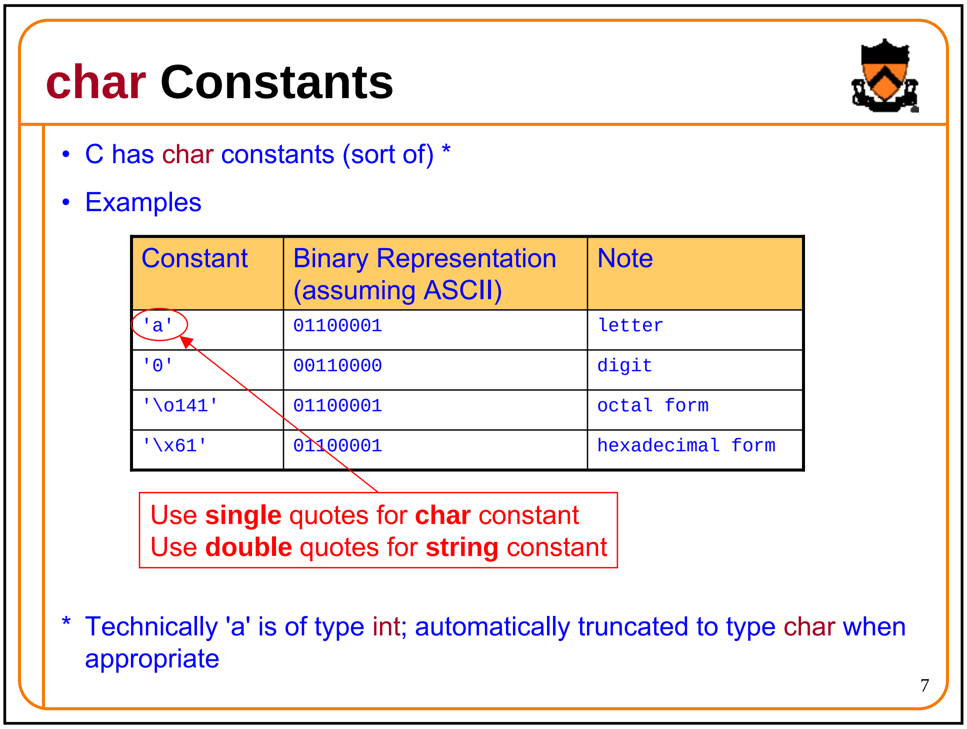
- Type char can be used for (limited range) arithmetic, but…
- Usually used to store characters – thus the name!
  - Must use a code to map 1-byte numbers to characters
  - Common code: ASCII
- Other ways to represent characters
  - Less common: EBCDIC
  - What about Unicode? “wide” characters (2 bytes)

The ASCII Code

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)



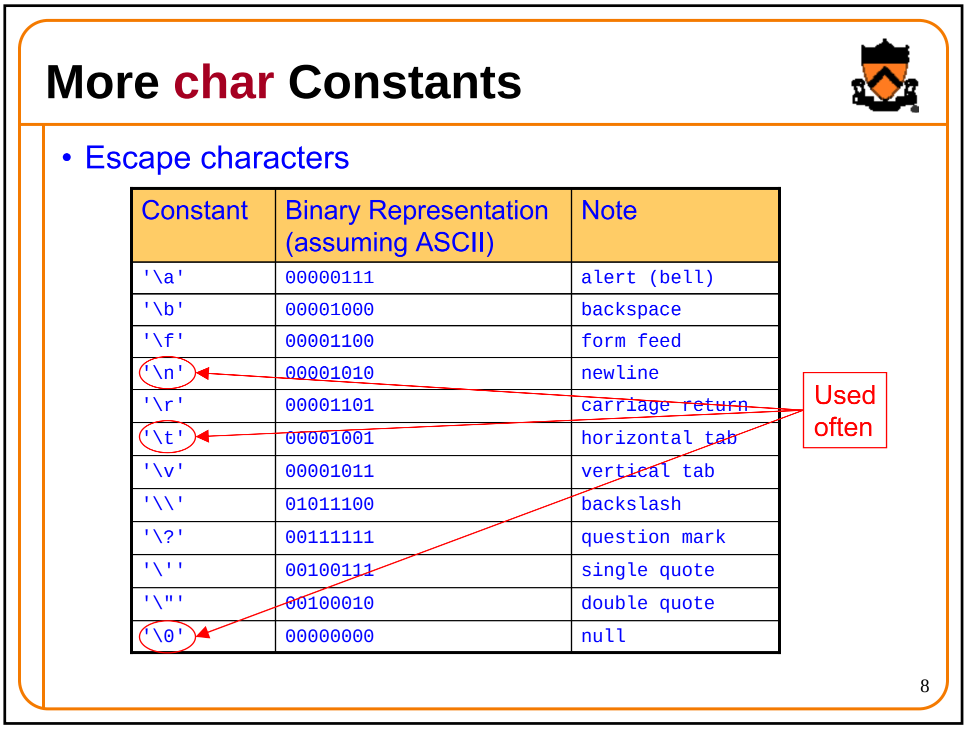
char Constants

- C has char constants (sort of) *
- Examples

Constant	Binary Representation (assuming ASCII)	Note
`'a'`	`01100001`	`letter`
`'0'`	`00110000`	`digit`
`'\o141'`	`01100001`	`octal form`
`'\x61'`	`01100001`	`hexadecimal form`

Use **single** quotes for **char** constant
Use **double** quotes for **string** constant

* Technically 'a' is of type int; automatically truncated to type char when appropriate

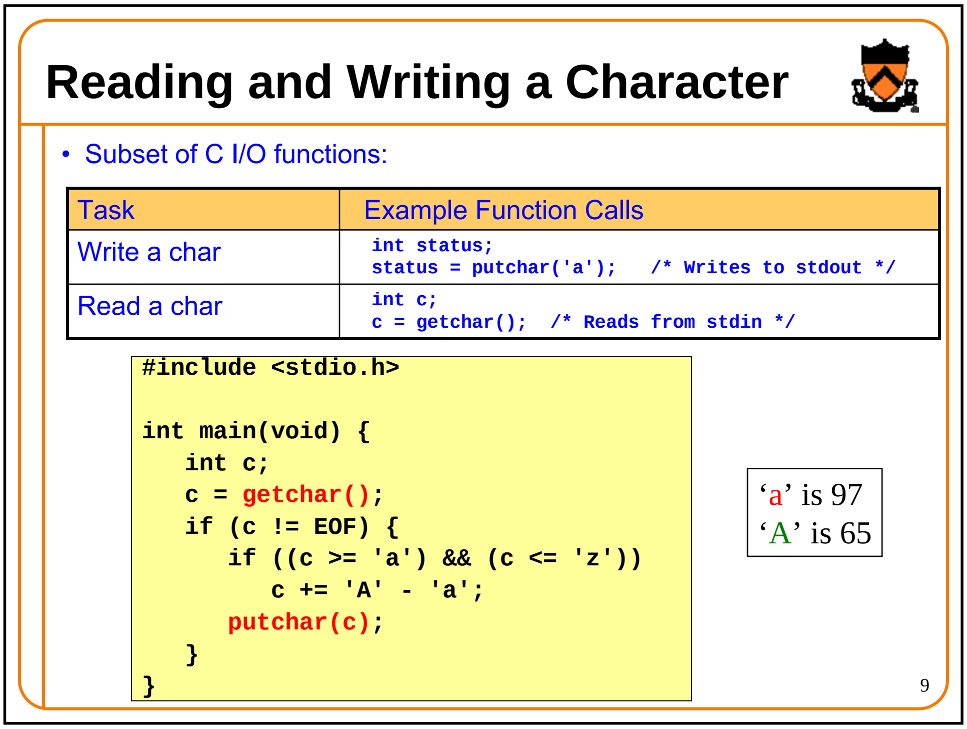


More char Constants

- Escape characters

Constant	Binary Representation (assuming ASCII)	Note
`'\a'`	`00000111`	`alert (bell)`
`'\b'`	`00001000`	`backspace`
`'\f'`	`00001100`	`form feed`
`'\n'`	`00001010`	`newline`
`'\r'`	`00001101`	`carriage return`
`'\t'`	`00001001`	`horizontal tab`
`'\v'`	`00001011`	`vertical tab`
`'\\'`	`01011100`	`backslash`
`'\?'`	`00111111`	`question mark`
`'\''`	`00100111`	`single quote`
`'\"'`	`00100010`	`double quote`
`'\0'`	`00000000`	`null`

Used often



Reading and Writing a Character

- Subset of C I/O functions:

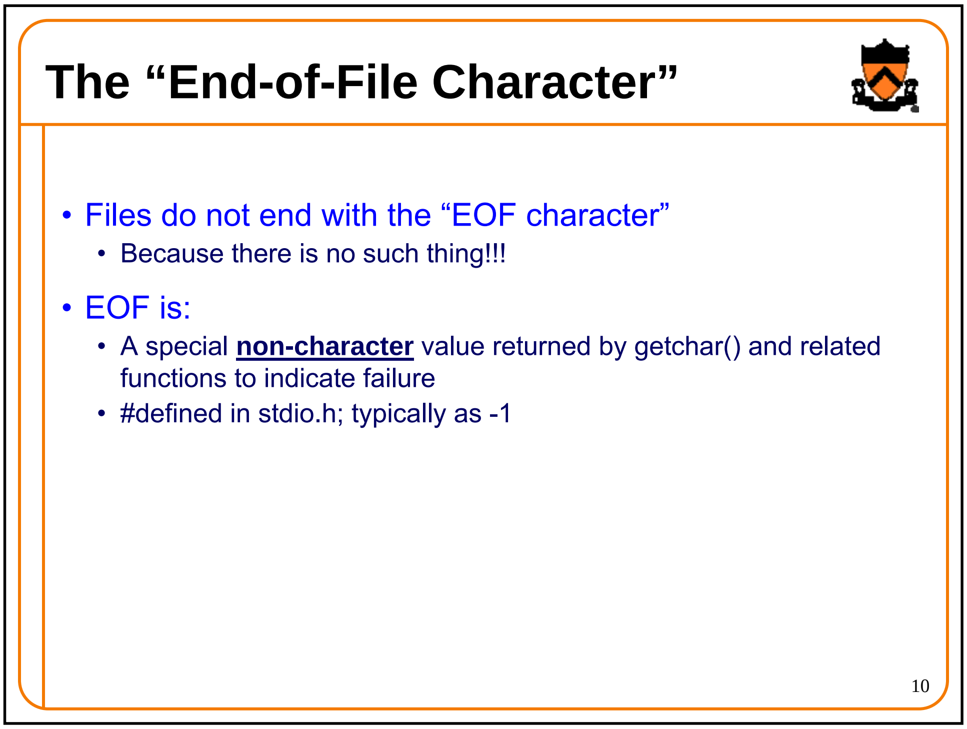
Task	Example Function Calls

Write a char	`int status;`
	`status = putchar('a');   /* Writes to stdout */`

Read a char	`int c;`
	`c = getchar();  /* Reads from stdin */`

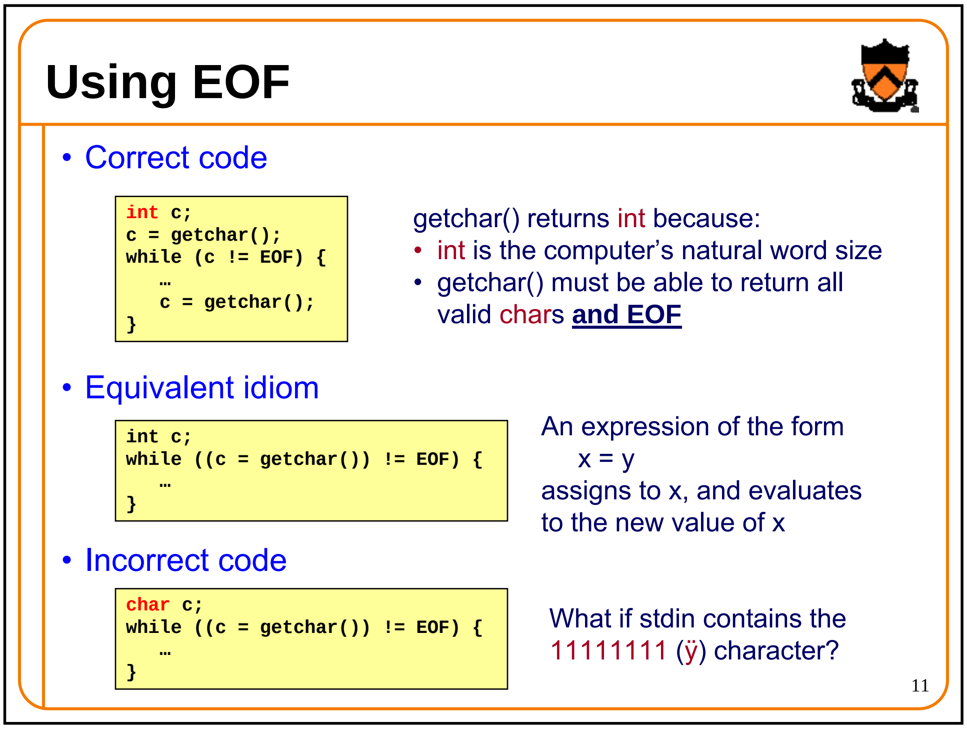
```
#include <stdio.h>

int main(void) {
   int c;
   c = getchar();
   if (c != EOF) {
      if ((c >= 'a') && (c <= 'z'))
         c += 'A' - 'a';
      putchar(c);
   }
}
```

‘a’ is 97
‘A’ is 65



The “End-of-File Character”

- Files do not end with the “EOF character”
  - Because there is no such thing!!!
- EOF is:
  - A special **non-character** value returned by getchar() and related functions to indicate failure
  - #defined in stdio.h; typically as -1



Using EOF

- Correct code

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

getchar() returns int because:

- int is the computer’s natural word size
- getchar() must be able to return all valid chars **and EOF**

- Equivalent idiom

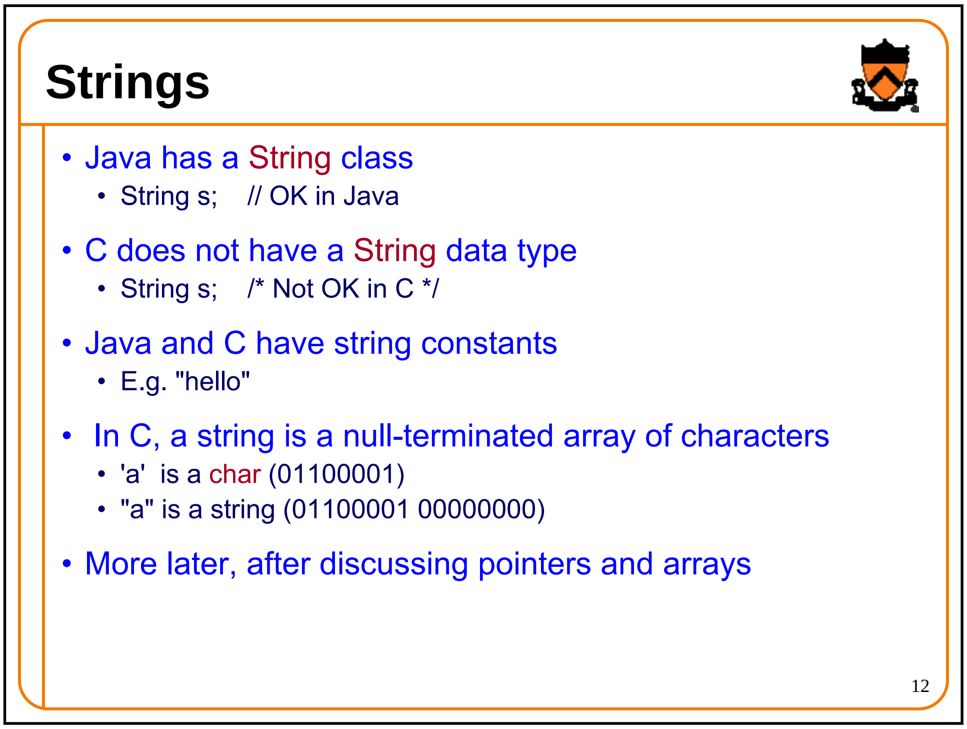
```
int c;
while ((c = getchar()) != EOF) {
   …
}
```

An expression of the form
    x = y
assigns to x, and evaluates to the new value of x

- Incorrect code

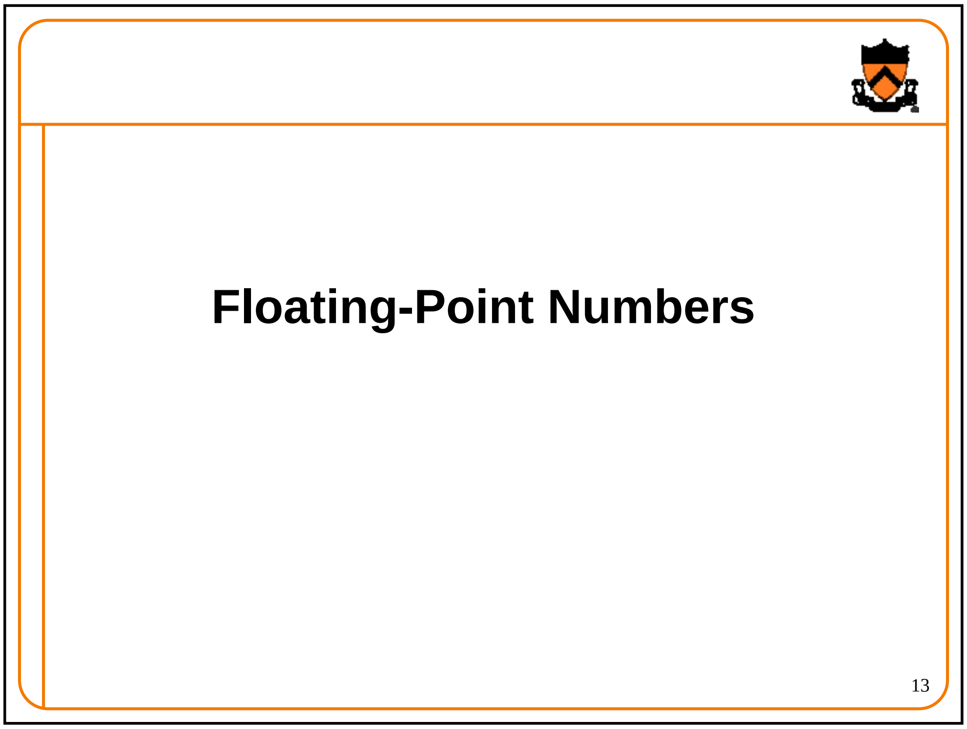
```
char c;
while ((c = getchar()) != EOF) {
   …
}
```

What if stdin contains the 11111111 (ÿ) character?

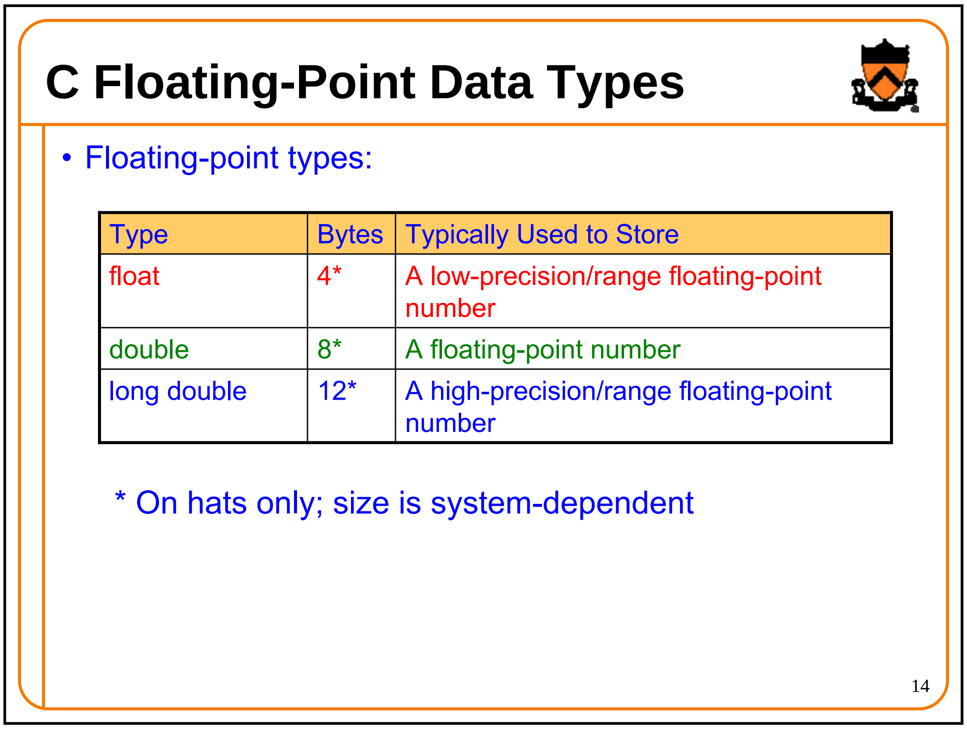


Strings

- Java has a String class
  - String s;   // OK in Java
- C does not have a String data type
  - String s;   /* Not OK in C */
- Java and C have string constants
  - E.g. "hello"
- In C, a string is a null-terminated array of characters
  - 'a' is a char (01100001)
  - "a" is a string (01100001 00000000)
- More later, after discussing pointers and arrays



Floating-Point Numbers

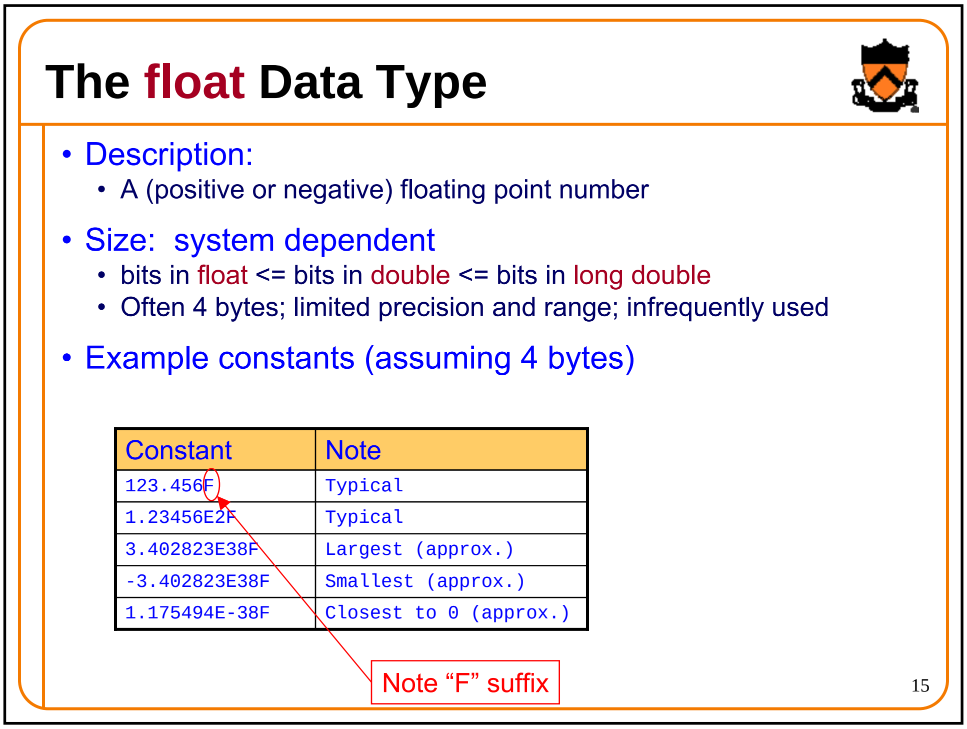


C Floating-Point Data Types

- Floating-point types:

Type	Bytes	Typically Used to Store
float	4*	A low-precision/range floating-point number
double	8*	A floating-point number
long double	12*	A high-precision/range floating-point number

* On hats only; size is system-dependent

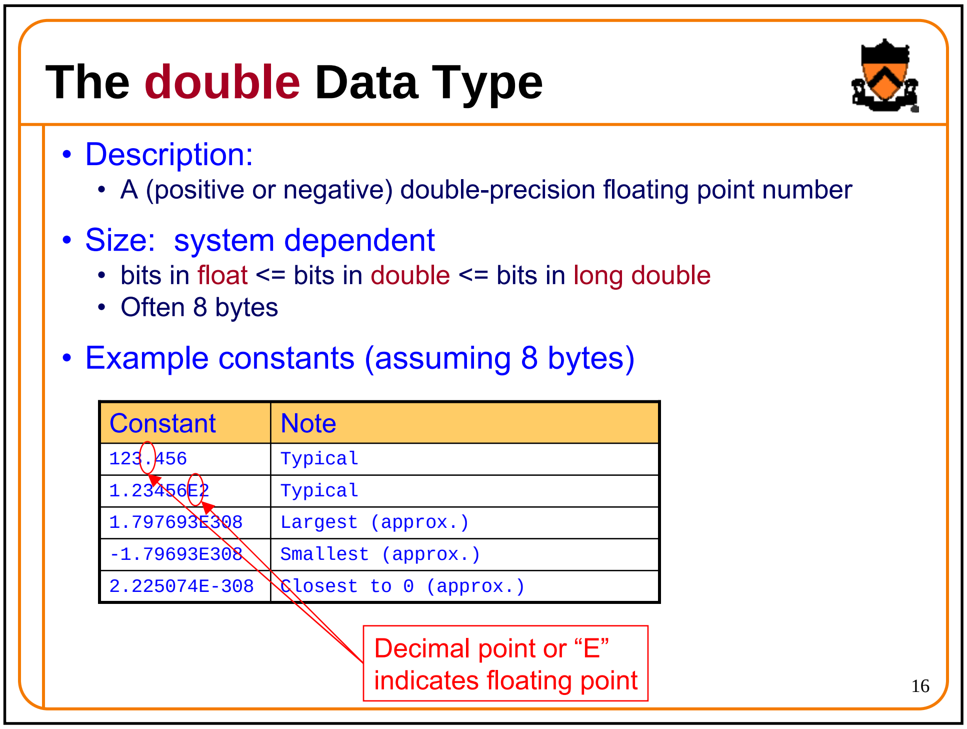


The float Data Type

- Description:
  - A (positive or negative) floating point number
- Size: system dependent
  - bits in float <= bits in double <= bits in long double
  - Often 4 bytes; limited precision and range; infrequently used
- Example constants (assuming 4 bytes)

Constant	Note
`123.456F`	`Typical`
`1.23456E2F`	`Typical`
`3.402823E38F`	`Largest (approx.)`
`-3.402823E38F`	`Smallest (approx.)`
`1.175494E-38F`	`Closest to 0 (approx.)`

Note “F” suffix



The double Data Type

- Description:
  - A (positive or negative) double-precision floating point number
- Size: system dependent
  - bits in float <= bits in double <= bits in long double
  - Often 8 bytes
- Example constants (assuming 8 bytes)

Constant	Note
`123.456`	`Typical`
`1.23456E2`	`Typical`
`1.797693E308`	`Largest (approx.)`
`-1.79693E308`	`Smallest (approx.)`
`2.225074E-308`	`Closest to 0 (approx.)`

Decimal point or “E” indicates floating point

The long double Data Type

- Description:
  - A (positive or negative) floating point number
- Size: system dependent
  - bits in float <= bits in double <= bits in long double
  - Often 10 or 12 bytes
- Example constants (assuming 12 bytes)

Constant	Note
`123.456L`	`Typical`
`1.23456E2L`	`Typical`
`1.189731E4932L`	`Largest (approx.)`
`-1.189731E4932L`	`Smallest (approx.)`
`3.362103E-4932L`	`Closest to 0 (approx.)`

Note “L” suffix

17

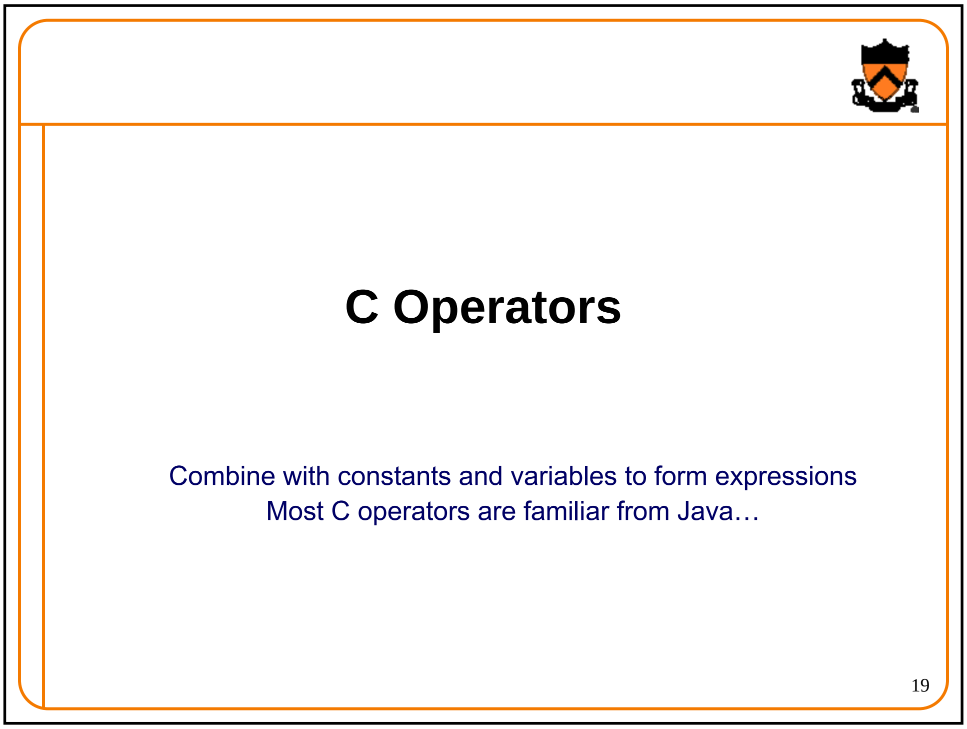
Data Types: C vs. Java

Java	C
boolean	(no equivalent)
byte	(no equivalent)
(no equivalent)	long double
(no equivalent)	unsigned types
char comprises 2 bytes (Unicode)	char comprises 1 byte (often ASCII)
Sizes of all types specified	char is one byte Sizes of all other types unspecified

Recall Java goal:
Portability → specify sizes

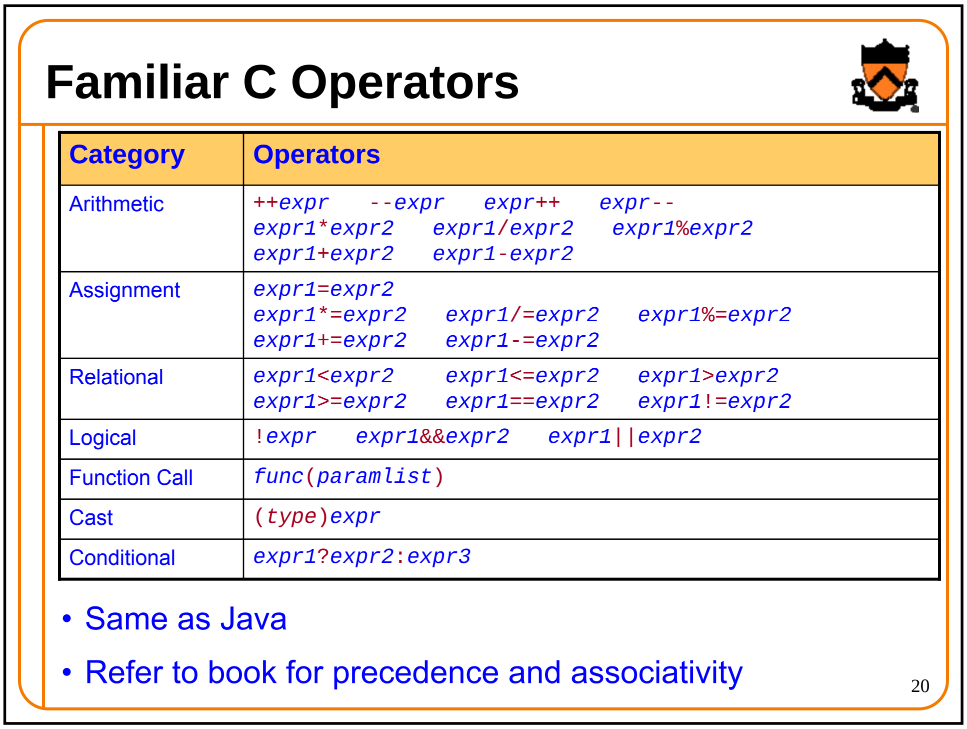
Recall C goal:
Create an OS → use natural word size

18



C Operators

Combine with constants and variables to form expressions
Most C operators are familiar from Java…



Familiar C Operators

Category	Operators

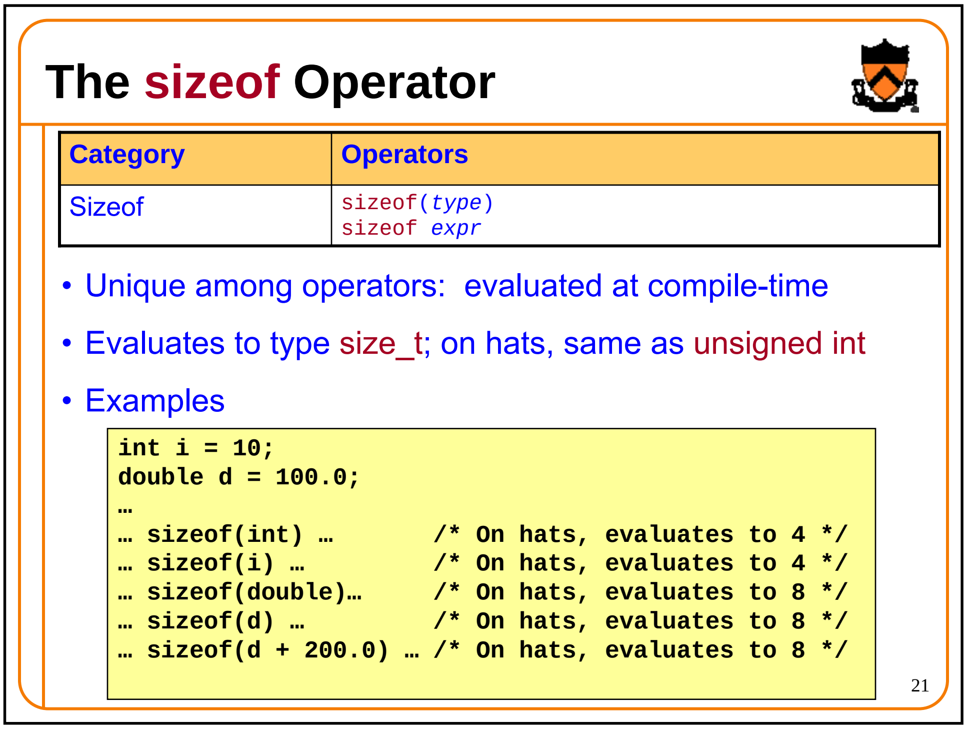
Arithmetic	`++expr   --expr   expr++   expr--`
	`expr1*expr2   expr1/expr2   expr1%expr2`
	`expr1+expr2   expr1-expr2`

Assignment	`expr1=expr2`
	`expr1*=expr2   expr1/=expr2   expr1%=expr2`
	`expr1+=expr2   expr1-=expr2`

Relational	`expr1<expr2   expr1<=expr2   expr1>expr2`
	`expr1>=expr2   expr1==expr2   expr1!=expr2`

Logical	`!expr   expr1&&expr2   expr1||expr2`
Function Call	`func(paramlist)`
Cast	`(type)expr`
Conditional	`expr1?expr2:expr3`

- Same as Java
- Refer to book for precedence and associativity



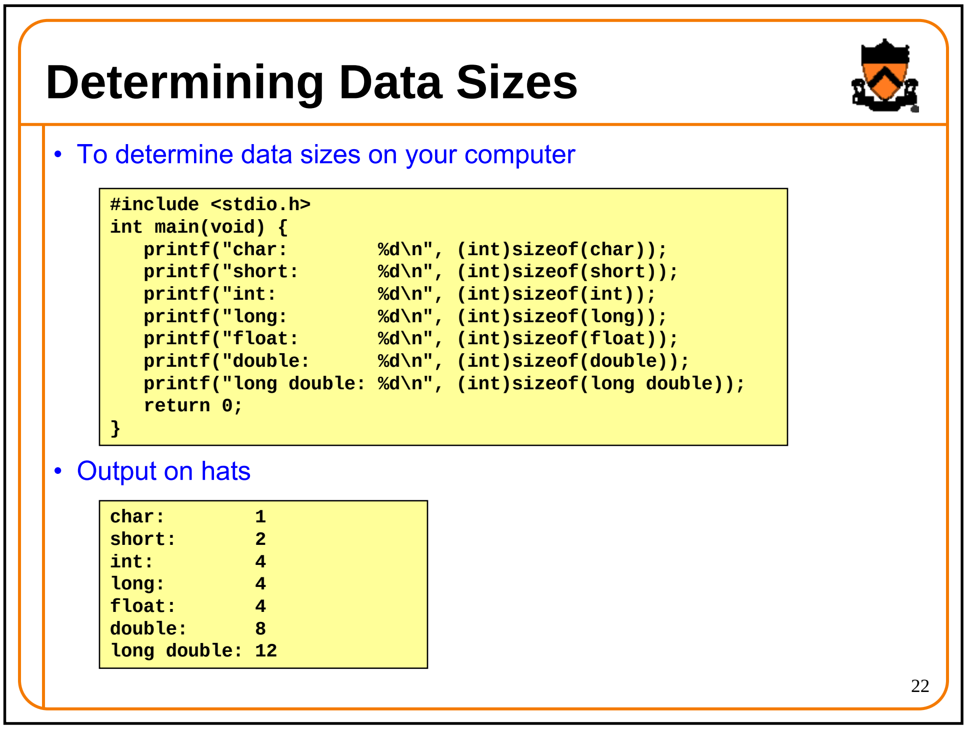
The sizeof Operator

Category	Operators

Sizeof	`sizeof(`*`type`*`)`
	`sizeof `*`expr`*

- Unique among operators: evaluated at compile-time
- Evaluates to type size_t; on hats, same as unsigned int
- Examples

```
int i = 10;
double d = 100.0;
…
… sizeof(int) …       /* On hats, evaluates to 4 */
… sizeof(i) …         /* On hats, evaluates to 4 */
… sizeof(double)…     /* On hats, evaluates to 8 */
… sizeof(d) …         /* On hats, evaluates to 8 */
… sizeof(d + 200.0) … /* On hats, evaluates to 8 */
```



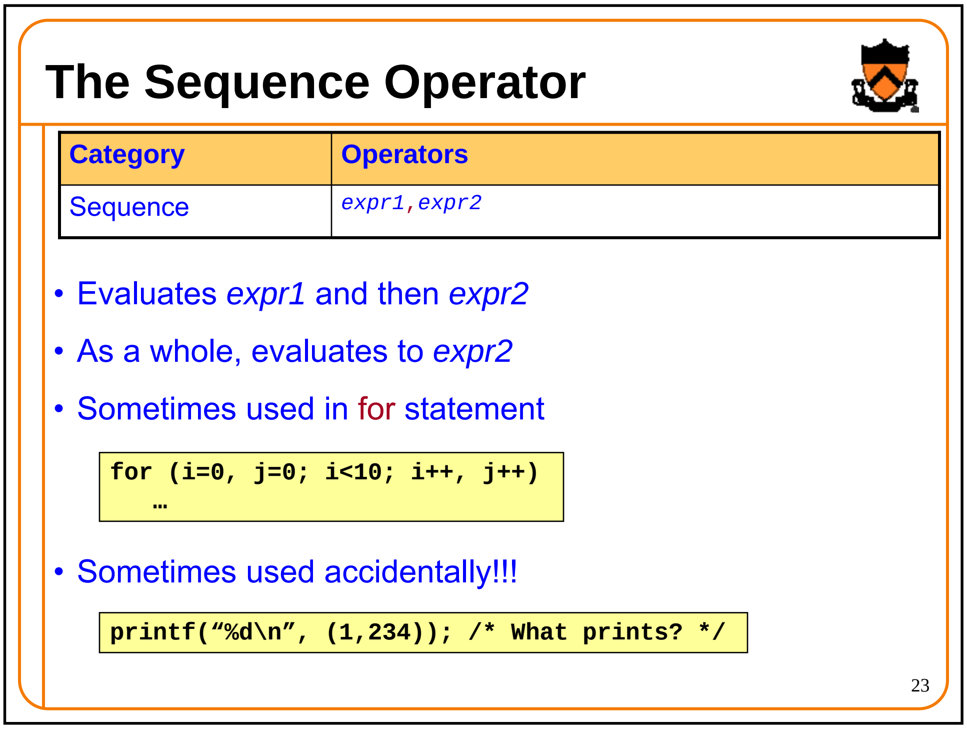
Determining Data Sizes

- To determine data sizes on your computer

```
#include <stdio.h>
int main(void) {
   printf("char:        %d\n", (int)sizeof(char));
   printf("short:       %d\n", (int)sizeof(short));
   printf("int:         %d\n", (int)sizeof(int));
   printf("long:        %d\n", (int)sizeof(long));
   printf("float:       %d\n", (int)sizeof(float));
   printf("double:      %d\n", (int)sizeof(double));
   printf("long double: %d\n", (int)sizeof(long double));
   return 0;
}
```

- Output on hats

```
char:        1
short:       2
int:         4
long:        4
float:       4
double:      8
long double: 12
```



The Sequence Operator

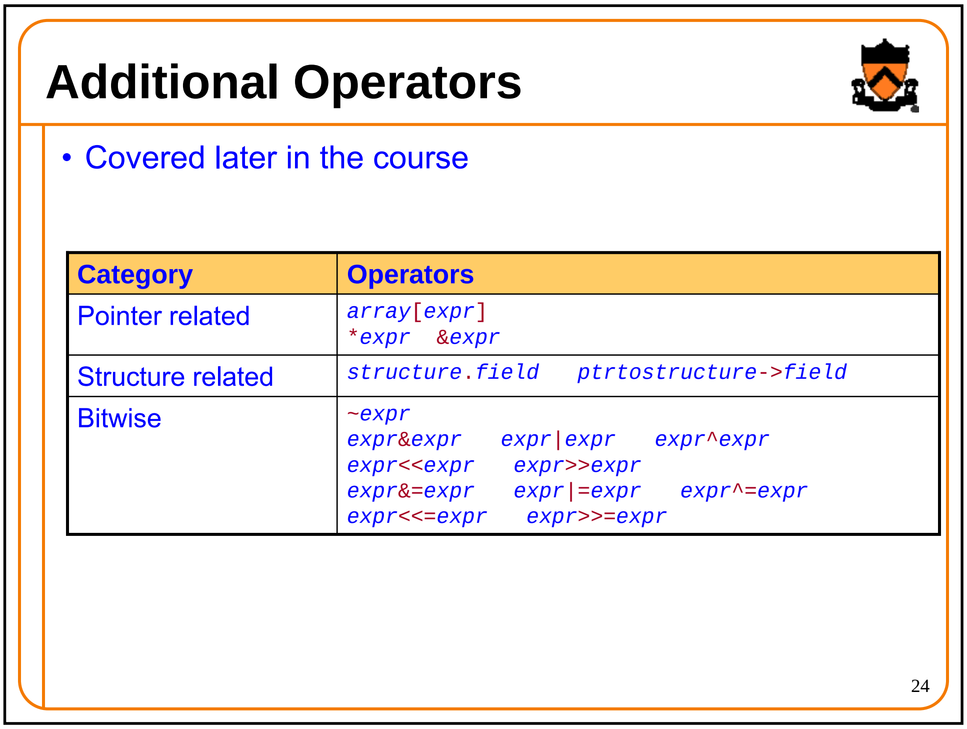
Category	Operators
Sequence	`expr1,expr2`

- Evaluates *expr1* and then *expr2*
- As a whole, evaluates to *expr2*
- Sometimes used in for statement

```
for (i=0, j=0; i<10; i++, j++)
   …
```

- Sometimes used accidentally!!!

```
printf(“%d\n”, (1,234)); /* What prints? */
```



Additional Operators

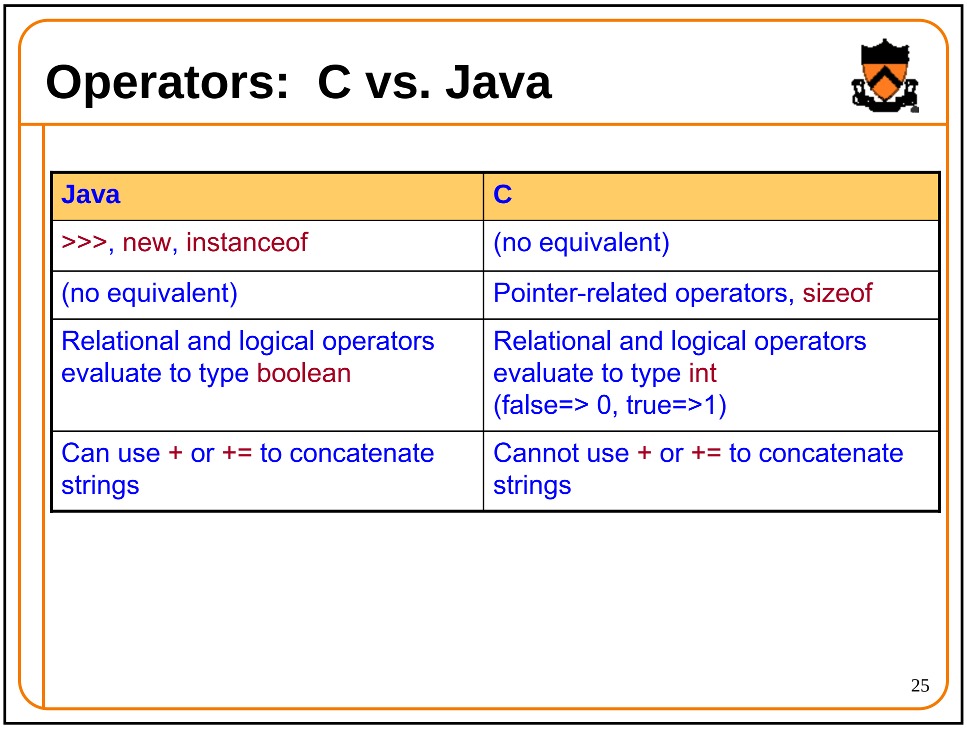
- Covered later in the course

Category	Operators

Pointer related	`array[expr]`
	`*expr   &expr`

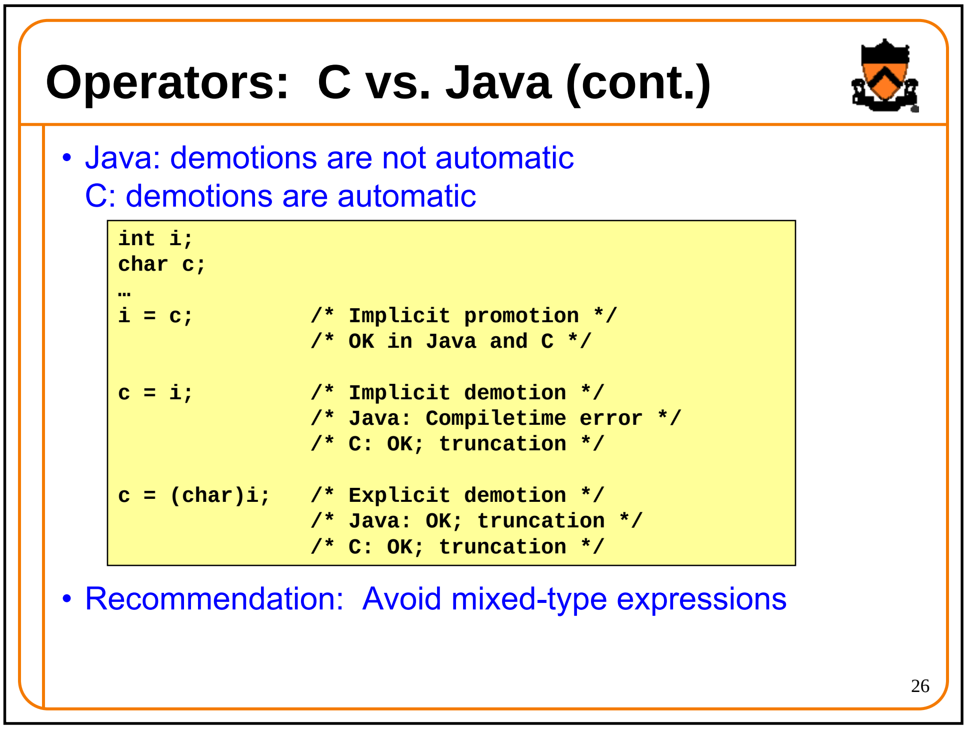
Structure related	`structure.field   ptrtostructure->field`

Bitwise	`~expr`
	`expr&expr   expr|expr   expr^expr`
	`expr<<expr   expr>>expr`
	`expr&=expr   expr|=expr   expr^=expr`
	`expr<<=expr   expr>>=expr`



Operators: C vs. Java

Java	C
>>>, new, instanceof	(no equivalent)
(no equivalent)	Pointer-related operators, sizeof
Relational and logical operators evaluate to type boolean	Relational and logical operators evaluate to type int (false=> 0, true=>1)
Can use + or += to concatenate strings	Cannot use + or += to concatenate strings

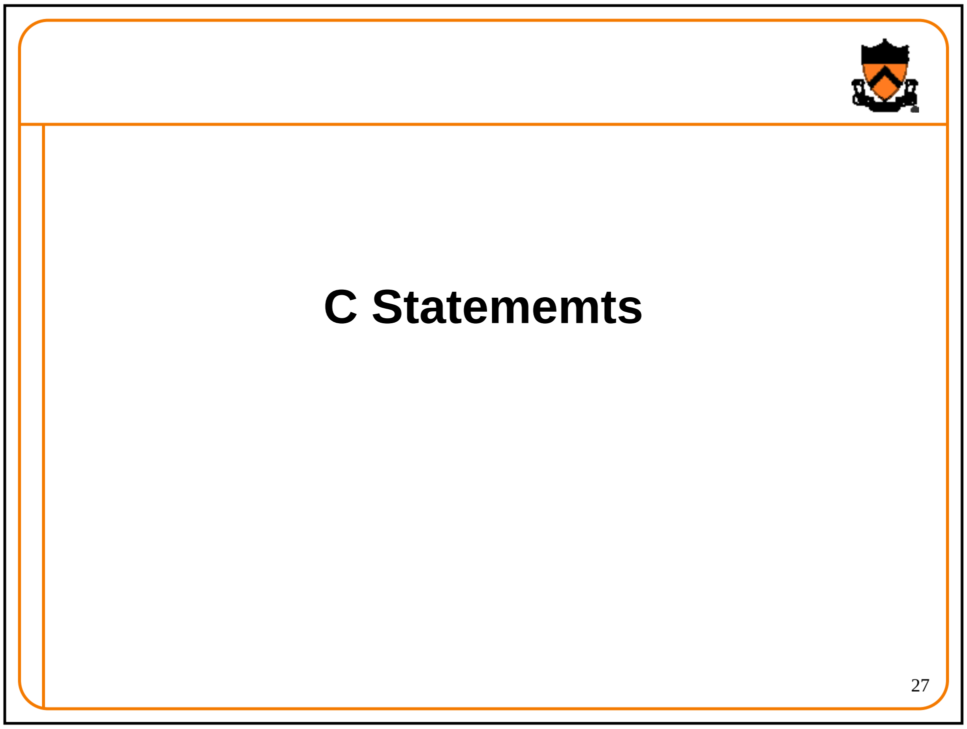


Operators: C vs. Java (cont.)

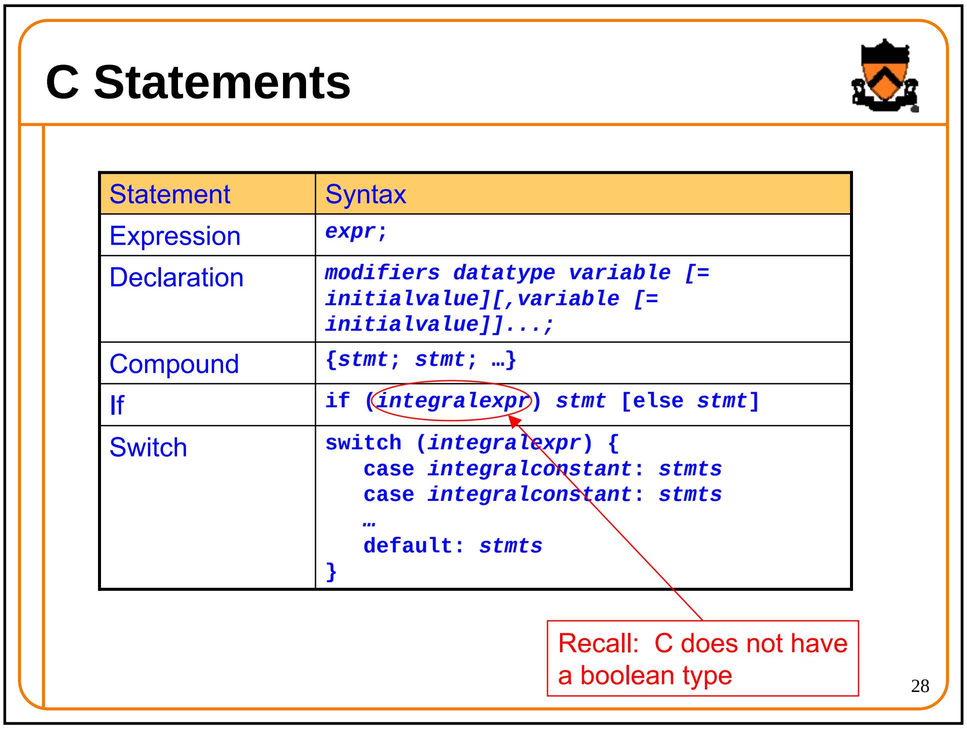
- Java: demotions are not automatic
  C: demotions are automatic

```
int i;
char c;
…
i = c;          /* Implicit promotion */
                /* OK in Java and C */

c = i;          /* Implicit demotion */
                /* Java: Compiletime error */
                /* C: OK; truncation */

c = (char)i;    /* Explicit demotion */
                /* Java: OK; truncation */
                /* C: OK; truncation */
```

- Recommendation: Avoid mixed-type expressions



C Statememts



C Statements

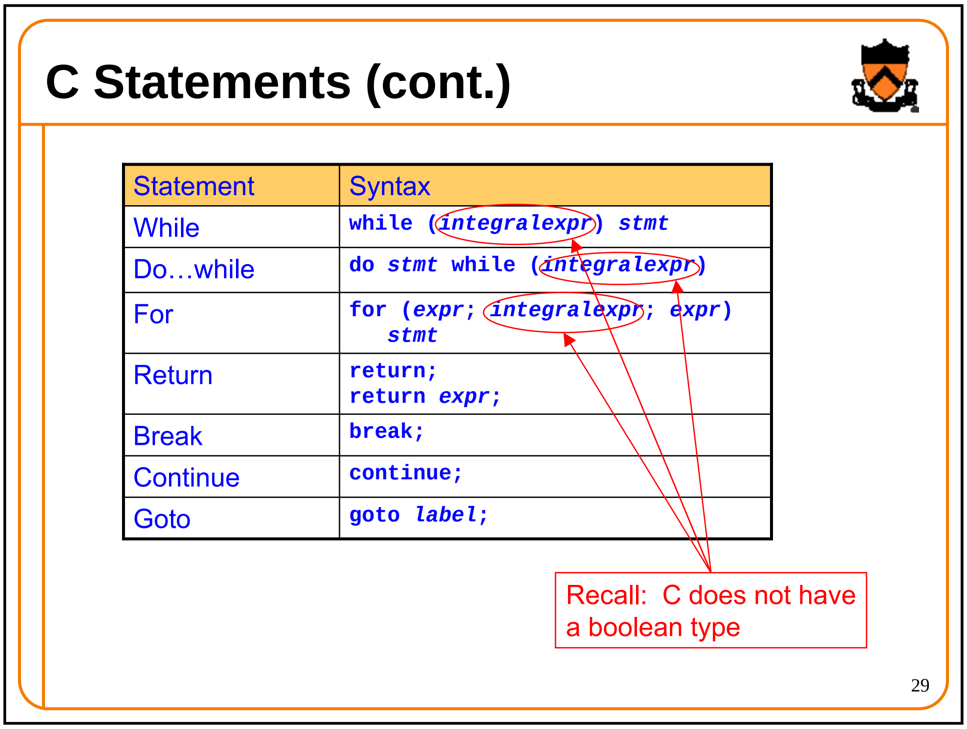
Statement	Syntax
Expression	`expr;`

Declaration	`modifiers datatype variable [=`
	`initialvalue][,variable [=`
	`initialvalue]]...;`

Compound	`{stmt; stmt; …}`
If	`if (integralexpr) stmt [else stmt]`

Switch	`switch (integralexpr) {`
	`   case integralconstant: stmts`
	`   case integralconstant: stmts`
	`   …`
	`   default: stmts`
	`}`

Recall: C does not have a boolean type



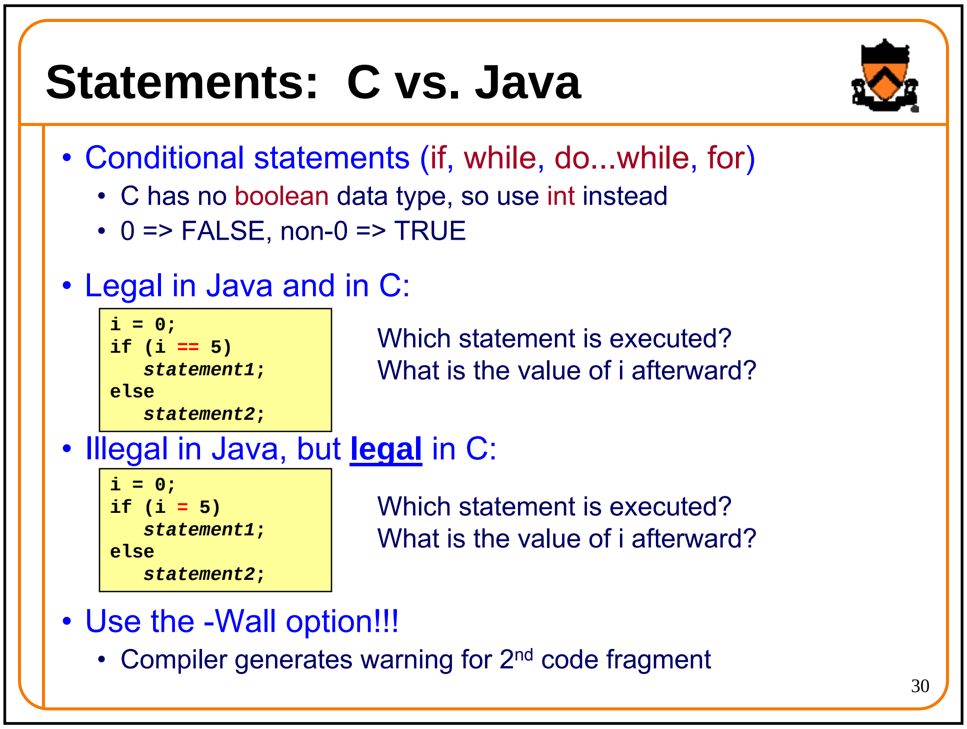
C Statements (cont.)

Statement	Syntax
While	while (integralexpr) stmt
Do…while	do stmt while (integralexpr)
For	for (expr; integralexpr; expr) stmt

Return	return;
	return expr;

Break	break;
Continue	continue;
Goto	goto label;

Recall: C does not have a boolean type



Statements: C vs. Java

- Conditional statements (if, while, do...while, for)
  - C has no boolean data type, so use int instead
  - 0 => FALSE, non-0 => TRUE
- Legal in Java and in C:

```
i = 0;
if (i == 5)
   statement1;
else
   statement2;
```

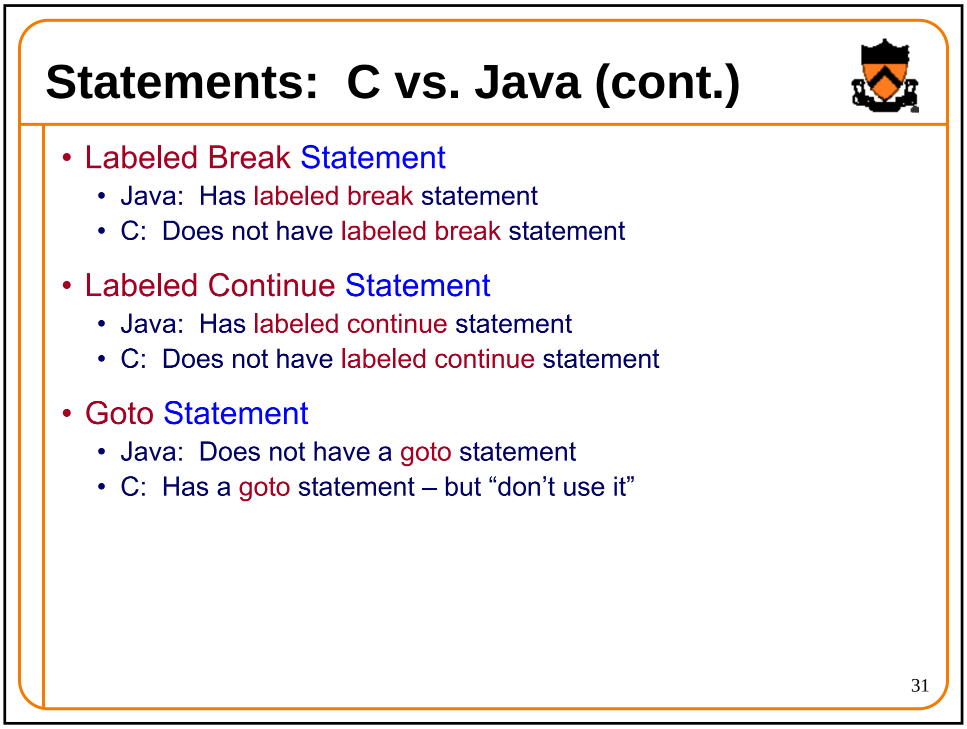
Which statement is executed?
What is the value of i afterward?

- Illegal in Java, but **legal** in C:

```
i = 0;
if (i = 5)
   statement1;
else
   statement2;
```

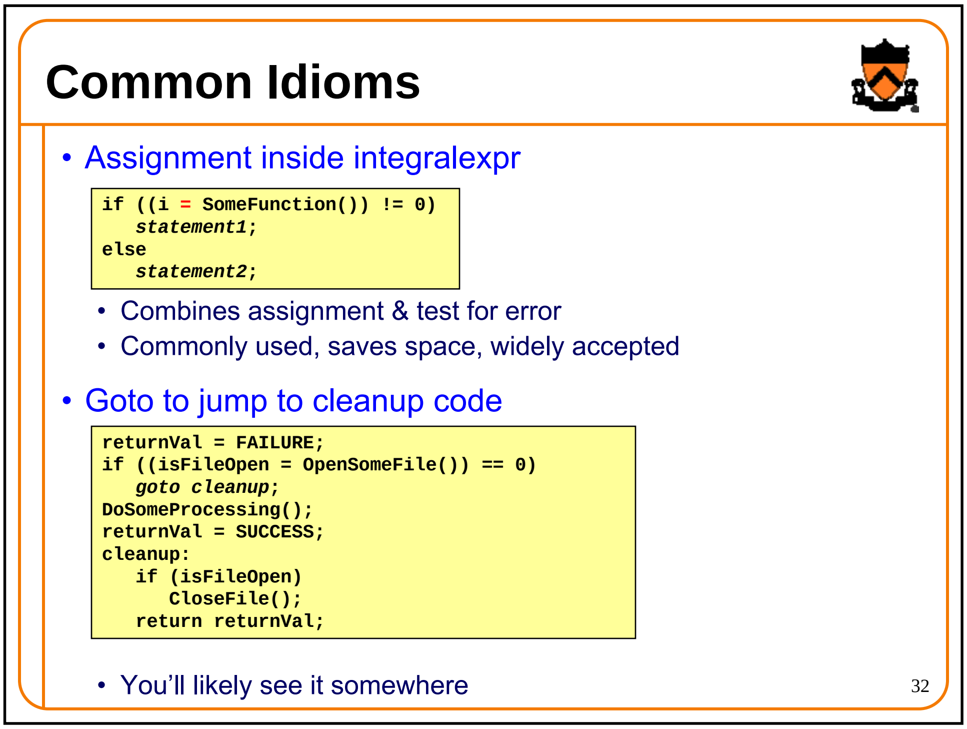
Which statement is executed?
What is the value of i afterward?

- Use the -Wall option!!!
  - Compiler generates warning for 2nd code fragment



Statements: C vs. Java (cont.)

- Labeled Break Statement
  - Java: Has labeled break statement
  - C: Does not have labeled break statement
- Labeled Continue Statement
  - Java: Has labeled continue statement
  - C: Does not have labeled continue statement
- Goto Statement
  - Java: Does not have a goto statement
  - C: Has a goto statement – but “don’t use it”



Common Idioms

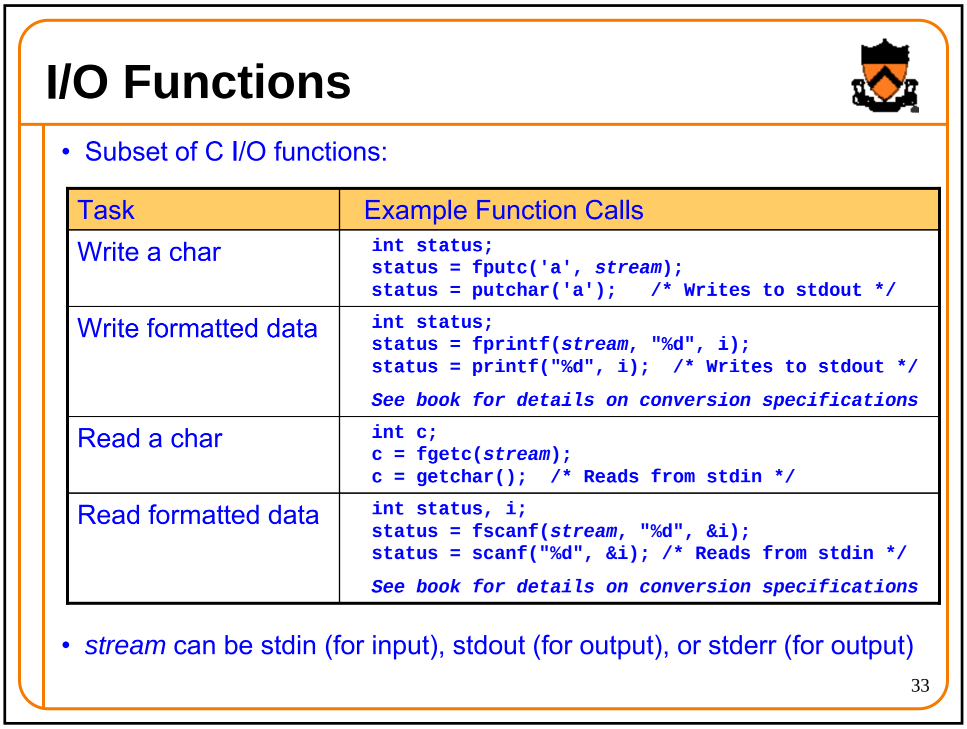
- Assignment inside integralexpr

```
if ((i = SomeFunction()) != 0)
   statement1;
else
   statement2;
```

  - Combines assignment & test for error
  - Commonly used, saves space, widely accepted
- Goto to jump to cleanup code

```
returnVal = FAILURE;
if ((isFileOpen = OpenSomeFile()) == 0)
   goto cleanup;
DoSomeProcessing();
returnVal = SUCCESS;
cleanup:
   if (isFileOpen)
      CloseFile();
   return returnVal;
```

  - You’ll likely see it somewhere



I/O Functions

- Subset of C I/O functions:

Task	Example Function Calls

Write a char	`int status;`
	`status = fputc('a', stream);`
	`status = putchar('a');   /* Writes to stdout */`

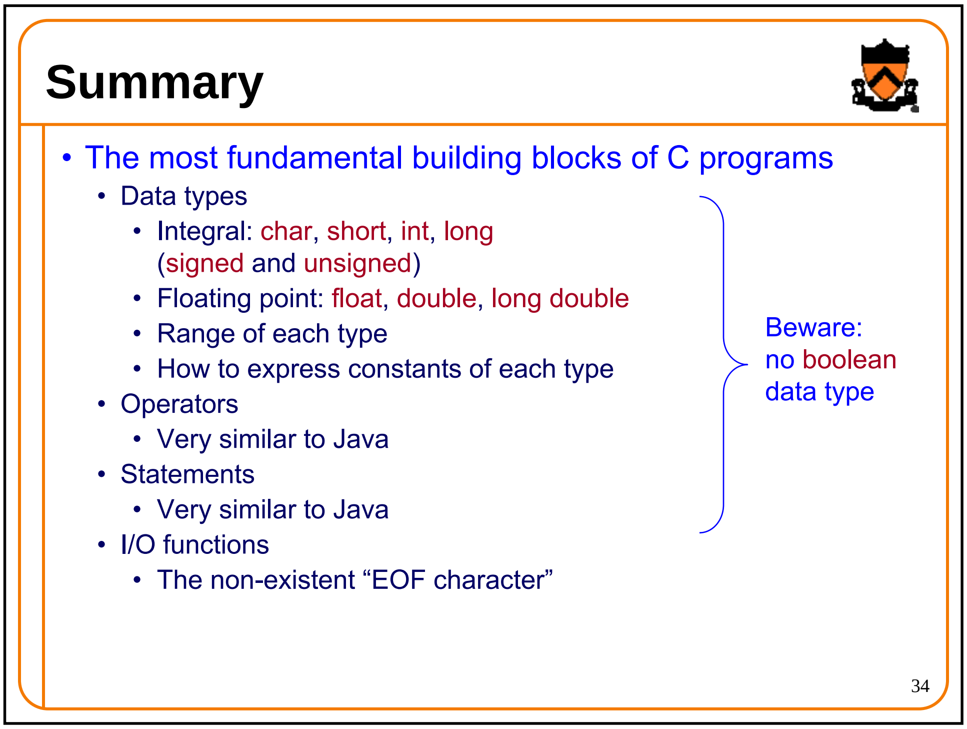
Write formatted data	`int status;`
	`status = fprintf(stream, "%d", i);`
	`status = printf("%d", i);  /* Writes to stdout */`
	See book for details on conversion specifications

Read a char	`int c;`
	`c = fgetc(stream);`
	`c = getchar();  /* Reads from stdin */`

Read formatted data	`int status, i;`
	`status = fscanf(stream, "%d", &i);`
	`status = scanf("%d", &i); /* Reads from stdin */`
	See book for details on conversion specifications

- *stream* can be stdin (for input), stdout (for output), or stderr (for output)

33



Summary

- The most fundamental building blocks of C programs
  - Data types
    - Integral: char, short, int, long (signed and unsigned)
    - Floating point: float, double, long double
    - Range of each type
    - How to express constants of each type
  - Operators
    - Very similar to Java
  - Statements
    - Very similar to Java
  - I/O functions
    - The non-existent “EOF character”

Beware:
no boolean
data type

34