

# Announcements

- Quiz 3
  - Monday, September 22
  - Covers: Lectures 5-6, Homework 5-6, Laboratory 3
- Assignment I
  - *Cellular Automata* with strings in C
  - To be posted this afternoon
  - Due 5:00 p.m., Friday, September 26
  - Write your code!

# C Reference Card (ANSI)

## Program Structure/Functions

```

type func(type1,...)
type name
main() {
    declarations
    statements
}
type func(arg1,...) {
    declarations
    statements
    return value;
}
/* */
main(int argc, char *argv[])
exit(arg)

```

## C Preprocessor

```

include library file
include user file
replacement text
replacement macro
    Example. #define max(A,B) ((A)>(B) ? (A) : (B))
undefine
quoted string in replace
concatenate args and rescan
conditional execution
    #if, #else, #elif, #endif
is name defined, not defined?
name defined?
line continuation char

```

## Data Types/Declarations

character (1 byte)	<i>char</i>
integer	<i>int</i>
float (single precision)	<i>float</i>
float (double precision)	<i>double</i>
short (16 bit integer)	<i>short</i>
long (32 bit integer)	<i>long</i>
positive and negative	<i>signed</i>
only positive	<i>unsigned</i>
pointer to <i>int</i> , <i>float</i> ,...	* <i>int</i> , * <i>float</i> ,...
enumeration constant	<i>enum</i>
constant (unchanging) value	<i>const</i>
declare external variable	<i>extern</i>
register variable	<i>register</i>
local to source file	<i>static</i>
no value	<i>void</i>
structure	<i>struct</i>
create name by data type	<i>typedef</i> <i>typename</i>
size of an object (type is <i>size_t</i> )	<i>sizeof</i> <i>object</i>
size of a data type (type is <i>size_t</i> )	<i>sizeof</i> ( <i>type</i> <i>name</i> )

## Initialization

```

initialize variable
initialize array
initialize char string

```

```

type name=value
type name[]={value1,...}
char name[]="string"

```

## Constants

<i>long</i> (suffix)	<i>L</i> or <i>l</i>
<i>float</i> (suffix)	<i>F</i> or <i>f</i>
exponential form	<i>e</i>
octal (prefix zero)	<i>0</i>
hexadecimal (prefix zero-ex)	<i>0x</i> or <i>0X</i>
character constant (char, octal, hex)	' <i>a</i> ', '\ooo', '\xhh'
newline, cr, tab, backspace	\n, \r, \t, \b
special characters	\\", ?, \\, "
string constant (ends with '\0')	"abc...de"

## Pointers, Arrays & Structures

declare pointer to <i>type</i>	<i>type</i> * <i>name</i>
declare function returning pointer to <i>type</i>	<i>type</i> * <i>f</i> ()
declare pointer to function returning <i>type</i>	<i>type</i> (* <i>pf</i> )()
generic pointer type	<i>void</i> *
null pointer	<i>NULL</i>
object pointed to by <i>pointer</i>	* <i>pointer</i>
address of object <i>name</i>	& <i>name</i>
array	<i>name</i> [ <i>dim</i> ]
multi-dim array	<i>name</i> [ <i>dim</i> <sub>1</sub> ][ <i>dim</i> <sub>2</sub> ]...
<b>Structures</b>	
<b>struct</b> <i>tag</i> {	structure template
<i>declarations</i>	declaration of members
};	
create structure	<b>struct</b> <i>tag</i> <i>name</i>
member of structure from template	<i>name</i> . <i>member</i>
member of pointed to structure	<i>pointer</i> -> <i>member</i>
Example. (* <i>p</i> ). <i>x</i> and <i>p</i> -> <i>x</i> are the same	
single value, multiple type structure	<b>union</b>
bit field with <i>b</i> bits	<i>member</i> : <i>b</i>

## Operators (grouped by precedence)

structure member operator	<i>name</i> . <i>member</i>
structure pointer	<i>pointer</i> -> <i>member</i>
increment, decrement	++, --
plus, minus, logical not, bitwise not	+, -, !, ~
indirection via pointer, address of object	* <i>pointer</i> , & <i>name</i>
cast expression to type	( <i>type</i> ) <i>expr</i>
size of an object	<b>sizeof</b>
multiply, divide, modulus (remainder)	*, /, %
add, subtract	+, -
left, right shift [bit ops]	<<, >>
comparisons	>, >=, <, <=
comparisons	==, !=
bitwise and	&
bitwise exclusive or	^
bitwise or (incl)	
logical and	&&
logical or	
conditional expression	<i>expr</i> <sub>1</sub> ? <i>expr</i> <sub>2</sub> : <i>expr</i> <sub>3</sub>
assignment operators	+=, -=, *=, ...
expression evaluation separator	,

Unary operators, conditional expression and assignment operators group right to left; all others group left to right.

## Flow of Control

statement terminator	;
block delimiters	{ }
exit from switch, while, do, for	break
next iteration of while, do, for	continue
go to	goto <i>label</i>
label	<i>label</i> :
return value from function	return <i>expr</i>

## Flow Constructions

<b>if</b> statement	<b>if</b> ( <i>expr</i> ) <i>statement</i>
	<b>else if</b> ( <i>expr</i> ) <i>statement</i>
	<b>else</b> <i>statement</i>
<b>while</b> statement	<b>while</b> ( <i>expr</i> )
	<i>statement</i>
<b>for</b> statement	<b>for</b> ( <i>expr</i> <sub>1</sub> ; <i>expr</i> <sub>2</sub> ; <i>expr</i> <sub>3</sub> )
	<i>statement</i>
<b>do</b> statement	<b>do</b> <i>statement</i>
	<b>while</b> ( <i>expr</i> );
<b>switch</b> statement	<b>switch</b> ( <i>expr</i> ) {
	<b>case</b> <i>const</i> <sub>1</sub> : <i>statement</i> <sub>1</sub> <b>break</b> ;
	<b>case</b> <i>const</i> <sub>2</sub> : <i>statement</i> <sub>2</sub> <b>break</b> ;
	<b>default</b> : <i>statement</i>

## ANSI Standard Libraries

```

<assert.h>   <ctype.h>   <errno.h>   <float.h>   <limits.h>
<locale.h>   <math.h>     <setjmp.h>  <signal.h>  <stdarg.h>
<stddef.h>   <stdio.h>   <stdlib.h>  <string.h>  <time.h>

```

## Character Class Tests <ctype.h>

alphanumeric?	isalnum( <i>c</i> )
alphabetic?	isalpha( <i>c</i> )
control character?	iscntrl( <i>c</i> )
decimal digit?	isdigit( <i>c</i> )
printing character (not incl space)?	isgraph( <i>c</i> )
lower case letter?	islower( <i>c</i> )
printing character (incl space)?	isprint( <i>c</i> )
printing char except space, letter, digit?	ispunct( <i>c</i> )
space, formfeed, newline, cr, tab, vtab?	isspace( <i>c</i> )
upper case letter?	isupper( <i>c</i> )
hexadecimal digit?	isxdigit( <i>c</i> )
convert to lower case?	tolower( <i>c</i> )
convert to upper case?	toupper( <i>c</i> )

## String Operations <string.h>

<i>s</i> , <i>t</i> are strings, <i>cs</i> , <i>ct</i> are constant strings	
length of <i>s</i>	<b>strlen</b> ( <i>s</i> )
copy <i>ct</i> to <i>s</i>	<b>strcpy</b> ( <i>s</i> , <i>ct</i> )
	<b>strncpy</b> ( <i>s</i> , <i>ct</i> , <i>n</i> )
concatenate <i>ct</i> after <i>s</i>	<b>strcat</b> ( <i>s</i> , <i>ct</i> )
	<b>strncat</b> ( <i>s</i> , <i>ct</i> , <i>n</i> )
compare <i>cs</i> to <i>ct</i>	<b>strcmp</b> ( <i>cs</i> , <i>ct</i> )
	<b>strncmp</b> ( <i>cs</i> , <i>ct</i> , <i>n</i> )
only first <i>n</i> chars	<b>strchr</b> ( <i>cs</i> , <i>c</i> )
pointer to first <i>c</i> in <i>cs</i>	<b>strrchr</b> ( <i>cs</i> , <i>c</i> )
pointer to last <i>c</i> in <i>cs</i>	<b>memcp</b> ( <i>s</i> , <i>ct</i> , <i>n</i> )
copy <i>n</i> chars from <i>ct</i> to <i>s</i>	<b>memmove</b> ( <i>s</i> , <i>ct</i> , <i>n</i> )
copy <i>n</i> chars from <i>ct</i> to <i>s</i> (may overlap)	<b>memcmp</b> ( <i>cs</i> , <i>ct</i> , <i>n</i> )
compare <i>n</i> chars of <i>cs</i> with <i>ct</i>	<b>memchr</b> ( <i>cs</i> , <i>c</i> , <i>n</i> )
pointer to first <i>c</i> in first <i>n</i> chars of <i>cs</i>	<b>memset</b> ( <i>s</i> , <i>c</i> , <i>n</i> )

# C Reference Card (ANSI)

## Input/Output <stdio.h>

### Standard I/O

standard input stream	stdin
standard output stream	stdout
standard error stream	stderr
end of file	EOF
get a character	getchar()
print a character	putchar(chr)
print formatted data	printf("format", arg1,...)
print to string s	sprintf(s,"format",arg1,...)
read formatted data	scanf("format",&name1,...)
read from string s	sscanf(s,"format",&name1,...)
read line to string s (< max chars)	gets(s,max)
print string s	puts(s)
File I/O	
declare file pointer	FILE *fp
pointer to named file	fopen("name","mode")
modes: r (read), w (write), a (append)	
get a character	getc(fp)
write a character	putc(chr,fp)
write to file	fprintf(fp,"format",arg1,...)
read from file	fscanf(fp,"format",arg1,...)
close file	fclose(fp)
non-zero if error	ferror(fp)
non-zero if EOF	feof(fp)
read line to string s (< max chars)	fgets(s,max,fp)
write string s	fputs(s,fp)

### Codes for Formatted I/O: "%-+ 0w.pmc"

-	left justify
+	print with sign
space	print space if no sign
0	pad with leading zeros
w	min field width
p	precision
m	conversion character: h short, l long, L long double
c	conversion character: d,i integer    u unsigned c single char   s char string f double       e,E exponential o octal       x,X hexadecimal p pointer      n number of chars written g,G same as f or e,E depending on exponent

## Variable Argument Lists <stdarg.h>

declaration of pointer to arguments    va\_list name;  
initialization of argument pointer    va\_start(name, lastarg)  
    lastarg is last named parameter of the function  
access next unnamed arg, update pointer    va\_arg(name,type)  
call before exiting function            va\_end(name)

## Standard Utility Functions <stdlib.h>

absolute value of int n	abs(n)
absolute value of long n	labs(n)
quotient and remainder of ints n,d	div(n,d)
return structure with div_t quot and div_t rem	
quotient and remainder of longs n,d	ldiv(n,d)
returns structure with ldiv_t quot and ldiv_t rem	
pseudo-random integer [0,RAND_MAX]	rand()
set random seed to n	srand(n)
terminate program execution	exit(status)
pass string s to system for execution	system(s)

### Conversions

convert string s to double	atof(s)
convert string s to integer	atoi(s)
convert string s to long	atol(s)
convert prefix of s to double	strtod(s,endp)
convert prefix of s (base b) to long	strtol(s,endp,b)
same, but unsigned long	strtoul(s,endp,b)

### Storage Allocation

allocate storage	malloc(size), calloc(nobj,size)
change size of object	realloc(pts,size)
deallocate space	free(ptr)

### Array Functions

search array for key	bsearch(key,array,n,size,cmp())
sort array ascending order	qsort(array,n,size,cmp())

## Time and Date Functions <time.h>

processor time used by program	clock()
Example, clock() /CLOCKS_PER_SEC is time in seconds	
current calendar time	time()
time <sub>2</sub> -time <sub>1</sub> in seconds (double)	difftime(time <sub>2</sub> ,time <sub>1</sub> )
arithmetic types representing times	clock_t, time_t
structure type for calendar time comps	tm
tm_sec	seconds after minute
tm_min	minutes after hour
tm_hour	hours since midnight
tm_mday	day of month
tm_mon	months since January
tm_year	years since 1900
tm_wday	days since Sunday
tm_yday	days since January 1
tm_isdst	Daylight Savings Time flag
convert local time to calendar time	mktime(tp)
convert time in tp to string	asctime(tp)
convert calendar time in tp to local time	ctime(tp)
convert calendar time to GMT	gmtime(tp)
convert calendar time to local time	localtime(tp)
format date and time info	strftime(s,max,"format",tp)
tp is a pointer to a structure of type tm	

## Mathematical Functions <math.h>

Arguments and returned values are double

trig functions	sin(x), cos(x), tan(x)
inverse trig functions	asin(x), acos(x), atan(x)
arctan(y/x)	atan2(y,x)
hyperbolic trig functions	sinh(x), cosh(x), tanh(x)
exponentials & logs	exp(x), log(x), log10(x)
exponentials & logs (2 power)	ldexp(x,n), frexp(x,*e)
division & remainder	modf(x,*ip), fmod(x,y)
powers	pow(x,y), sqrt(x)
rounding	ceil(x), floor(x), fabs(x)

## Integer Type Limits <limits.h>

The numbers given in parentheses are typical values for the constants on a 32-bit Unix system.

CHAR_BIT	bits in char	(8)
CHAR_MAX	max value of char	(127 or 255)
CHAR_MIN	min value of char	(-128 or 0)
INT_MAX	max value of int	(+32,767)
INT_MIN	min value of int	(-32,768)
LONG_MAX	max value of long	(+2,147,483,647)
LONG_MIN	min value of long	(-2,147,483,648)
SCHAR_MAX	max value of signed char	(+127)
SCHAR_MIN	min value of signed char	(-128)
SHRT_MAX	max value of short	(+32,767)
SHRT_MIN	min value of short	(-32,768)
UCHAR_MAX	max value of unsigned char	(255)
UINT_MAX	max value of unsigned int	(65,535)
ULONG_MAX	max value of unsigned long	(4,294,967,295)
USHRT_MAX	max value of unsigned short	(65,536)

## Floating Type Limits <float.h>

FLT_RADIX	radix of exponent rep	(2)
FLT_ROUNDS	floating point rounding mode	
FLT_DIG	decimal digits of precision	(6)
FLT_EPSILON	smallest x so 1.0 + x ≠ 1.0	(10 <sup>-5</sup> )
FLT_MANT_DIG	number of digits in mantissa	
FLT_MAX	maximum floating point number	(10 <sup>37</sup> )
FLT_MAX_EXP	maximum exponent	
FLT_MIN	minimum floating point number	(10 <sup>-37</sup> )
FLT_MIN_EXP	minimum exponent	
DBL_DIG	decimal digits of precision	(10)
DBL_EPSILON	smallest x so 1.0 + x ≠ 1.0	(10 <sup>-9</sup> )
DBL_MANT_DIG	number of digits in mantissa	
DBL_MAX	max double floating point number	(10 <sup>37</sup> )
DBL_MAX_EXP	maximum exponent	
DBL_MIN	min double floating point number	(10 <sup>-37</sup> )
DBL_MIN_EXP	minimum exponent	

May 1999 v1.3. Copyright © 1999 Joseph H. Silverman

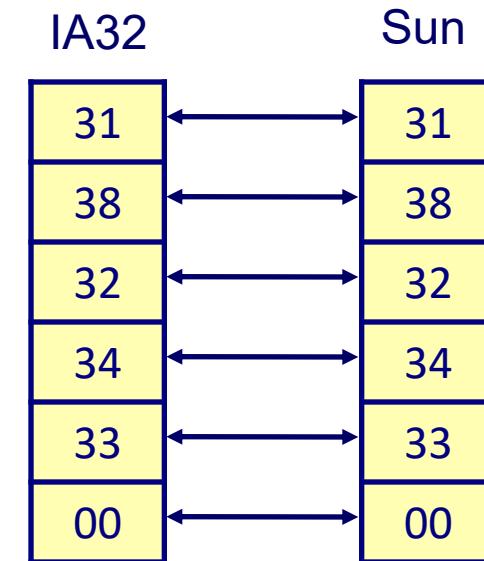
Permission is granted to make and distribute copies of this card provided the copyright notice and this permission notice are preserved on all copies.

Send comments and corrections to J.H. Silverman, Math. Dept., Brown Univ., Providence, RI 02912 USA. (jhs@math.brown.edu)

# Representing Strings

- **Strings in C**
  - Represented by an array of characters
  - Each character encoded in *ASCII* format
    - Standard 7-bit encoding of character set
    - Character “0” has code 0x30
      - Digit  $i$  has code  $0x30+i$
  - String should be *null-terminated*
    - Final character = 0
- **Compatibility**
  - Byte ordering not an issue

```
char S[6] = "18243";
```



Lecture 7

# Integer Representations III

CPSC 275  
Introduction to Computer Systems

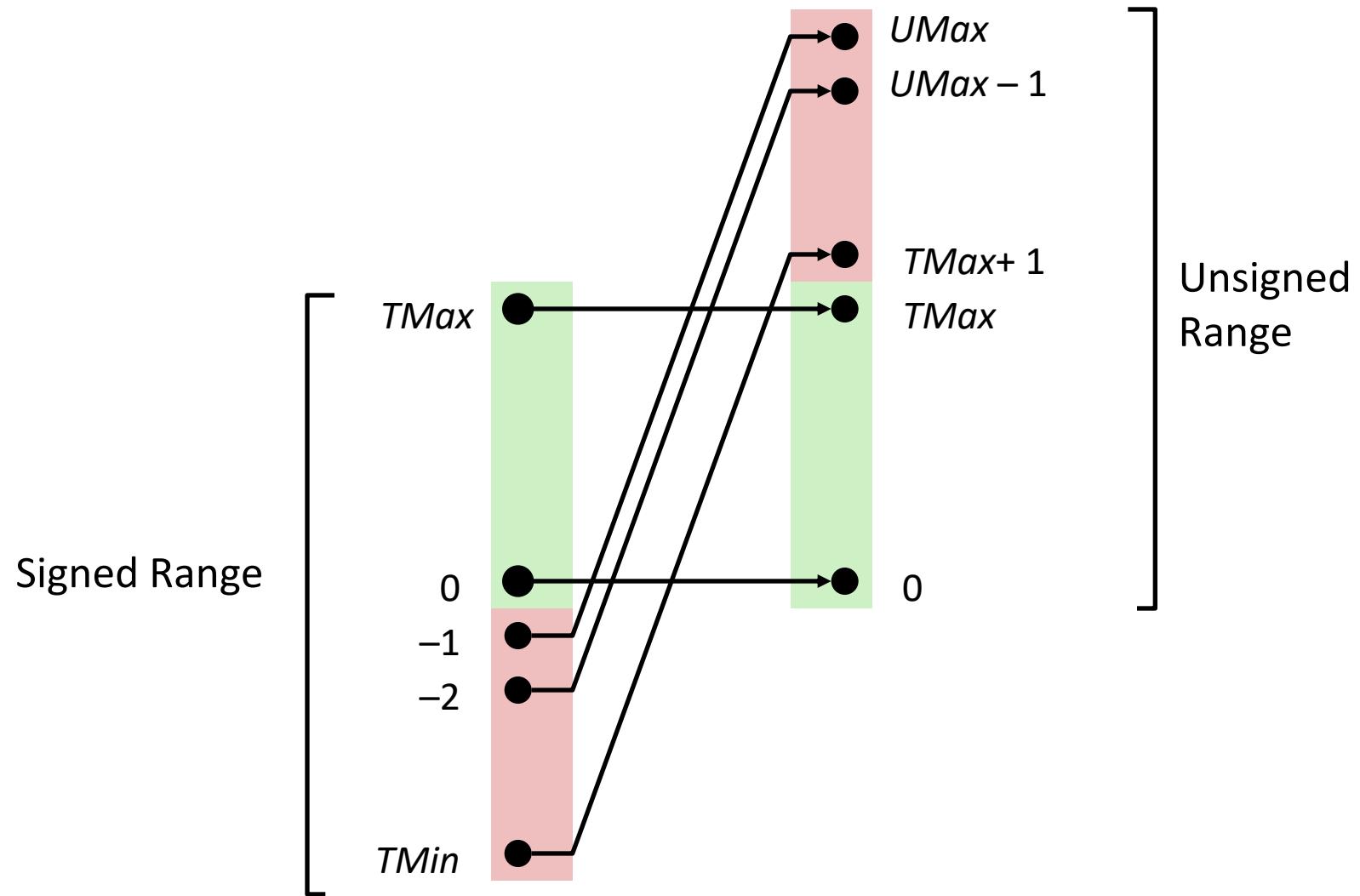
# Signed and Unsigned in C

- Constants
  - Default: signed integers
  - Unsigned if have “U” as suffix  
    0U, 4294967259U
- Casting
  - *Explicit* casting between signed & unsigned

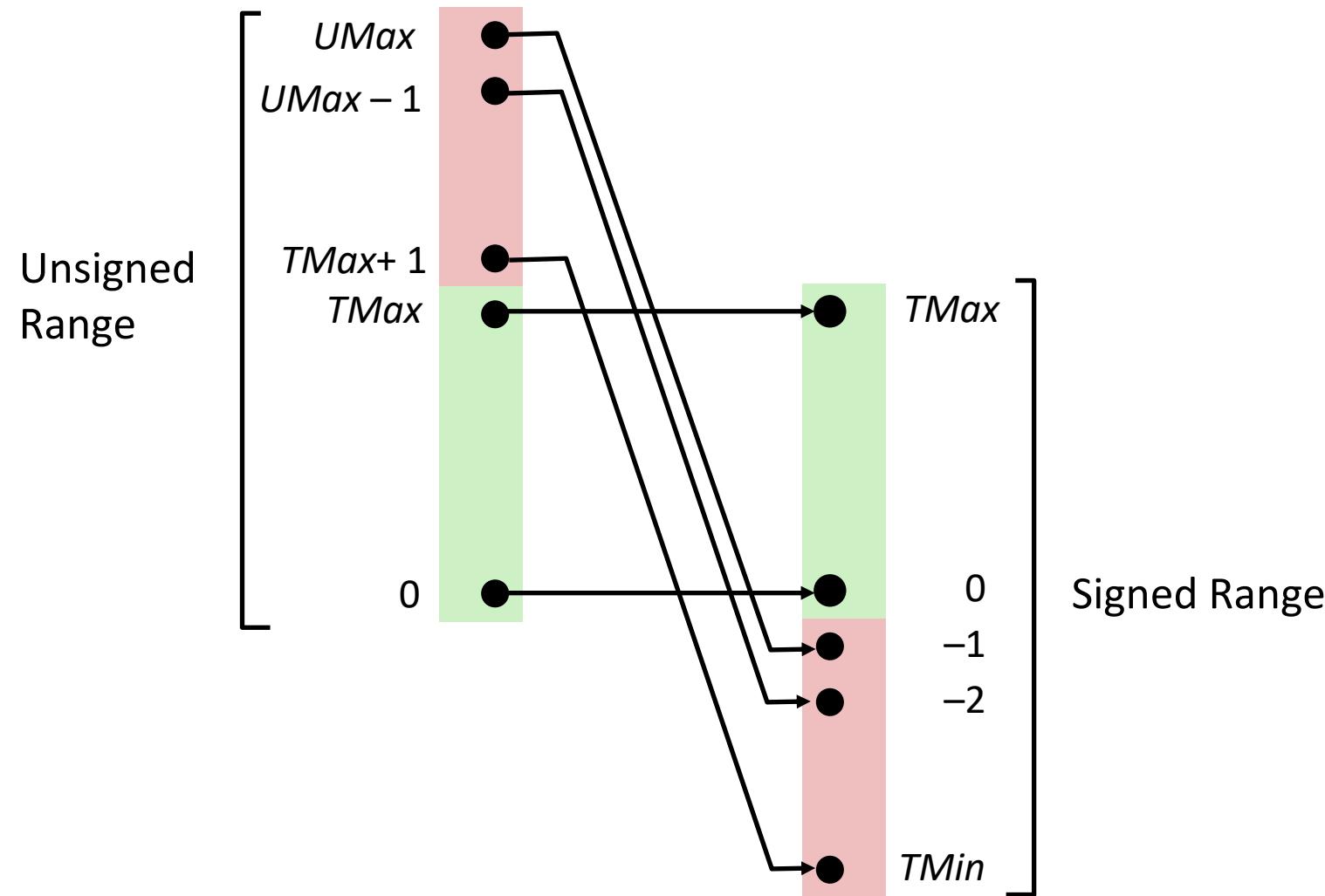
```
int tx, ty;  
  
unsigned ux, uy;  
  
tx = (int) ux;  
  
uy = (unsigned) ty;
```
  - *Implicit* casting also occurs via assignments and procedure calls

```
tx = ux;  
  
uy = ty;
```

# Signed → Unsigned Conversion



# Unsigned Conversion → Signed



# Casting Surprises

## ■ Expression Evaluation

- If there is a mix of unsigned and signed in single expression,  
*signed values implicitly cast to unsigned*
- Including comparison operations `<`, `>`, `==`, `<=`, `>=`
- Examples for  $w = 32$ : **TMIN = -2,147,483,648** , **TMAX = 2,147,483,647**

Constant <sub>1</sub>	Constant <sub>2</sub>	Relation	Evaluation
0	0U	==	unsigned
-1	0	<	signed
-1	0U	>	unsigned
2147483647	-2147483648	>	signed
2147483647U	-2147483648	<	unsigned
-1	-2	>	signed
(unsigned)-1	-2	>	unsigned



# Code Security Example

```
/* Declaration of library function memcpy */
void *memcpy(void *dest, void *src, size_t n);

/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel(OS)region to user buffer
 */
int copy_from_kernel(char user_dest[], int maxlen) {
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

# Typical usage

```
/* Declaration of library function memcpy */
void *memcpy(void *dest, void *src, size_t n);

/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel(OS)region to user buffer */
int copy_from_kernel(char user_dest[], int maxlen) {
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, MSIZE);
    printf("%s\n", mybuf);
}
```

# Malicious usage

```
/* Declaration of library function memcpy */
void *memcpy(void *dest, void *src, size_t n);

/* Kernel memory region holding user-accessible data */
#define KSIZE 1024
char kbuf[KSIZE];

/* Copy at most maxlen bytes from kernel(OS)region to user buffer
 */
int copy_from_kernel(char user_dest[], int maxlen) {
    int len = KSIZE < maxlen ? KSIZE : maxlen;
    memcpy(user_dest, kbuf, len);
    return len;
}
```

```
#define MSIZE 528

void getstuff() {
    char mybuf[MSIZE];
    copy_from_kernel(mybuf, -MSIZE);
    . .
}
```

What's happening?

