

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

<i>type fnc(type₁,...)</i>	function declarations
<i>type name</i>	external variable declarations
<i>main()</i> { <i>declarations</i> <i>statements</i> }	main routine local variable declarations
<i>type fnc(arg₁,...)</i> { <i>declarations</i> <i>statements</i> return <i>value</i> ; }	function definition local variable declarations
<i>/* */</i>	comments
<i>main(int argc, char *argv[])</i>	main with args
<i>exit(arg)</i>	terminate execution

C Preprocessor

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

Data Types/Declarations

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

Initialization

initialize variable	type name = value
initialize array	type name[] = {value ₁ ,...}
initialize char string	char name[] = "string"

Constants

long (suffix)	L or l
float (suffix)	F or f
exponential form	e
octal (prefix zero)	0
hexadecimal (prefix zero-ex)	0x or 0X
character constant (char, octal, hex)	'a', '\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 type	type *name
declare function returning pointer to type	type *f()
declare pointer to function returning type	type (*pf)()
generic pointer type	void *
null pointer	NULL
object pointed to by pointer	*pointer
address of object name	&name
array	name[dim]
multi-dim array	name[dim ₁][dim ₂]...

Structures

struct tag { declarations };	structure template declaration of members
create structure	struct tag name
member of structure from template	name.member
member of pointed to structure	pointer -> member
Example. (*p).x and p->x are the same	
single value, multiple type structure	union
bit field with b bits	member : b

Operators (grouped by precedence)

structure member operator	name.member
structure pointer	pointer->member
increment, decrement	++, --
plus, minus, logical not, bitwise not	+, -, !, ~
indirection via pointer, address of object	*pointer, &name
cast expression to type	(type) expr
size of an object	sizeof
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	expr ₁ ? expr ₂ : expr ₃
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 label
label	label:
return value from function	return expr

Flow Constructions

if statement	if (expr) statement else if (expr) statement else statement
while statement	while (expr) statement
for statement	for (expr ₁ ; expr ₂ ; expr ₃) statement
do statement	do statement while(expr);
switch statement	switch (expr) { case const ₁ : statement ₁ break; case const ₂ : statement ₂ break; default: statement }

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

String Operations <string.h>

s,t are strings, cs,ct are constant strings

length of s	strlen(s)
copy ct to s	strcpy(s,ct)
up to n chars	strncpy(s,ct,n)
concatenate ct after s	strcat(s,ct)
up to n chars	strncat(s,ct,n)
compare cs to ct	strcmp(cs,ct)
only first n chars	strncmp(cs,ct,n)
pointer to first c in cs	strchr(cs,c)
pointer to last c in cs	strrchr(cs,c)
copy n chars from ct to s	memcpy(s,ct,n)
copy n chars from ct to s (may overlap)	memmove(s,ct,n)
compare n chars of cs with ct	memcmp(cs,ct,n)
pointer to first c in first n chars of cs	memchr(cs,c,n)
put c into first n chars of cs	memset(s,c,n)

C Reference Card (ANSI)

Input/Output <stdio.h>

Standard I/O

standard input stream	<code>stdin</code>
standard output stream	<code>stdout</code>
standard error stream	<code>stderr</code>
end of file	<code>EOF</code>
get a character	<code>getchar()</code>
print a character	<code>putchar(chr)</code>
print formatted data	<code>printf("format", arg1, ...)</code>
print to string s	<code>sprintf(s, "format", arg1, ...)</code>
read formatted data	<code>scanf("format", &name1, ...)</code>
read from string s	<code>sscanf(s, "format", &name1, ...)</code>
read line to string s (< max chars)	<code>gets(s, max)</code>
print string s	<code>puts(s)</code>

File I/O

declare file pointer	<code>FILE *fp</code>
pointer to named file	<code>fopen("name", "mode")</code>
modes: r (read), w (write), a (append)	
get a character	<code>getc(fp)</code>
write a character	<code>putc(chr, fp)</code>
write to file	<code>fprintf(fp, "format", arg1, ...)</code>
read from file	<code>fscanf(fp, "format", arg1, ...)</code>
close file	<code>fclose(fp)</code>
non-zero if error	<code>ferror(fp)</code>
non-zero if EOF	<code>feof(fp)</code>
read line to string s (< max chars)	<code>fgets(s, max, fp)</code>
write string s	<code>fputs(s, fp)</code>

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	<code>va_list name;</code>
initialization of argument pointer	<code>va_start(name, lastarg)</code>
lastarg is last named parameter of the function	
access next unnamed arg, update pointer	<code>va_arg(name, type)</code>
call before exiting function	<code>va_end(name)</code>

Standard Utility Functions <stdlib.h>

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

Conversions

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

Storage Allocation

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

Array Functions

search array for key	<code>bsearch(key, array, n, size, cmp())</code>
sort array ascending order	<code>qsort(array, n, size, cmp())</code>

Time and Date Functions <time.h>

processor time used by program	<code>clock()</code>
Example. <code>clock()/CLOCKS_PER_SEC</code> is time in seconds	
current calendar time	<code>time()</code>
<code>time2-time1</code> in seconds (double)	<code>difftime(time2, time1)</code>
arithmetic types representing times	<code>clock_t, time_t</code>
structure type for calendar time comps	<code>tm</code>
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	<code>mktime(tp)</code>
convert time in tp to string	<code>asctime(tp)</code>
convert calendar time in tp to local time	<code>ctime(tp)</code>
convert calendar time to GMT	<code>gmtime(tp)</code>
convert calendar time to local time	<code>localtime(tp)</code>
format date and time info	<code>strftime(s, smax, "format", tp)</code>
tp is a pointer to a structure of type <code>tm</code>	

Mathematical Functions <math.h>

Arguments and returned values are double

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

Integer Type Limits <limits.h>

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

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

Float Type Limits <float.h>

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

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

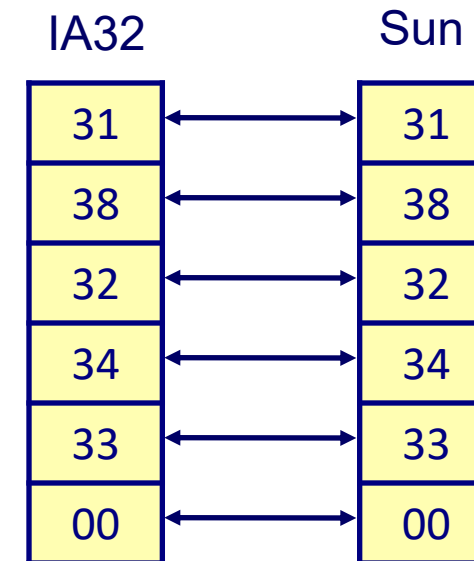
Permission is granted to make and distribute copies of this card provided the copyright notice and this permission notice are preserved on all copies.

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;
```

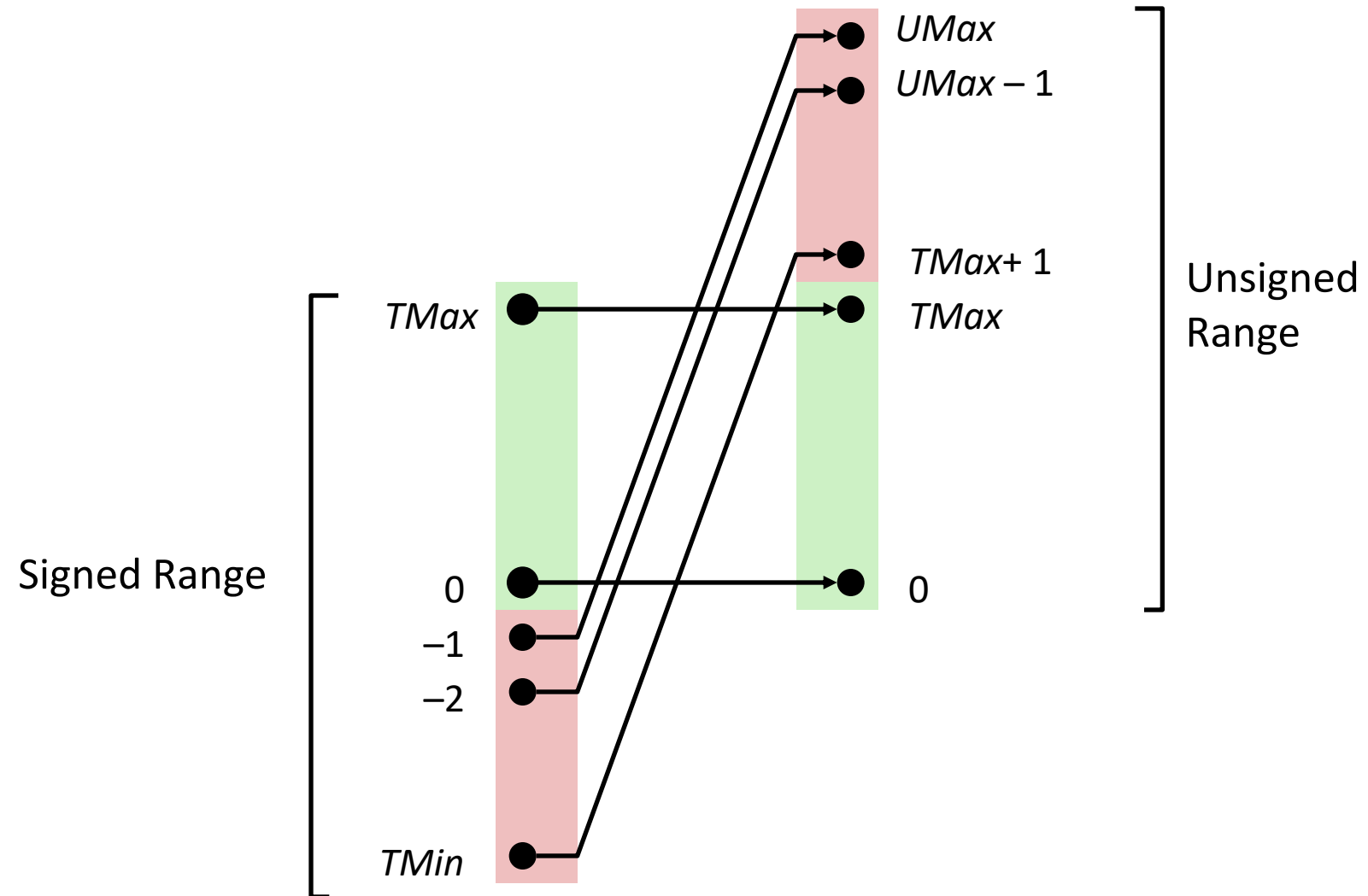
```
uy = (unsigned) ty;
```

- *Implicit* casting also occurs via assignments and procedure calls

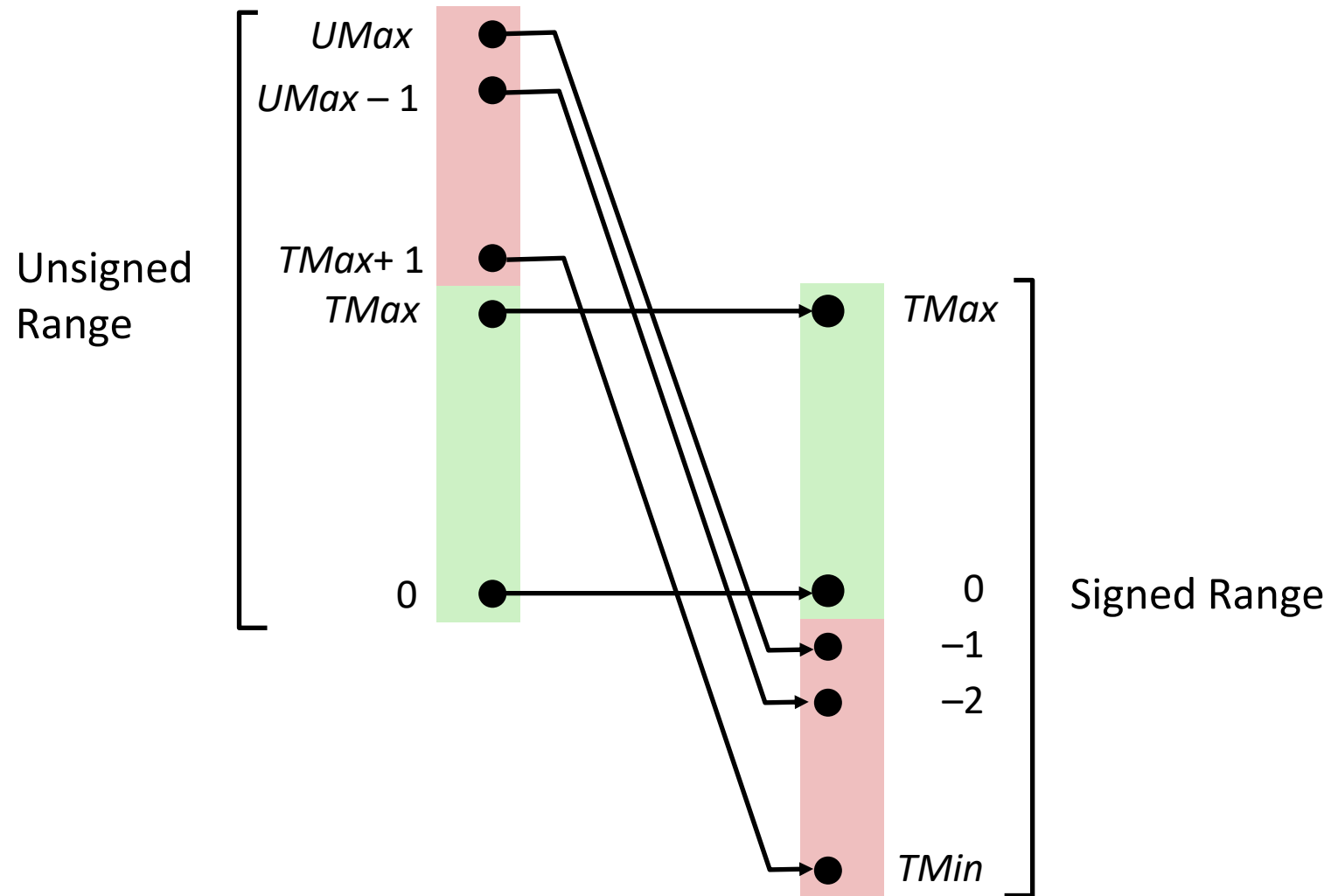
```
tx = ux;
```

```
uy = ty;
```

Signed \rightarrow Unsigned Conversion



Unsigned Conversion \rightarrow 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 ₁	Constant ₂	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?

