INTEGER PROMOTION:
If an int can represent all values of the original type, the value is converted to an int; otherwise, it is converted to an unsigned int. These are called the integer promotions. All other types are unchanged by the integer promotions.

IMPLICIT TYPE CONVERSION:

```
bool -> char -> short int -> int ->
       unsigned int -> long -> unsigned ->
       long long -> float -> double -> long double
```

When casting values between `int`, `float`, and `double` formats, the program changes the numeric values and the bit representations as follows (assuming data type `int` is 32 bits):

- From `int` to `float`, the number cannot overflow, but it may be rounded.
- From `int` or `float` to `double`, the exact numeric value can be preserved because `double` has both greater range (i.e., the range of representable values), as well as greater precision (i.e., the number of significant bits).
- From `double` to `float`, the value can overflow to $+\infty$ or $-\infty$, since the range is smaller. Otherwise, it may be rounded, because the precision is smaller.
- From `float` or `double` to `int`, the value will be rounded toward zero. For example, 1.999 will be converted to 1, while −1.999 will be converted to −1. Furthermore, the value may overflow. The C standards do not specify a fixed result for this case. Intel-compatible microprocessors designate the bit pattern $[10\cdots00]$ ($TMin_w$ for word size w) as an *integer indefinite* value. Any conversion from floating point to integer that cannot assign a reasonable integer approximation yields this value. Thus, the expression `(int) +1e10` yields -21483648, generating a negative value from a positive one.

2.2.4 Conversions between Signed and Unsigned

C allows casting between different numeric data types. For example, suppose variable x is declared as `int` and u as `unsigned`. The expression `(unsigned) x` converts the value of x to an unsigned value, and `(int) u` converts the value of u to a signed integer. What should be the effect of casting signed value to unsigned, or vice versa? From a mathematical perspective, one can imagine several different conventions. Clearly, we want to preserve any value that can be represented in both forms. On the other hand, converting a negative value to unsigned might yield zero. Converting an unsigned value that is too large to be represented in two's-complement form might yield *TMax*. For most implementations of C, however, the answer to this question is based on a bit-level perspective, rather than on a numeric one.

For example, consider the following code:

```
1       short int v = -12345;
2       unsigned short uv = (unsigned short) v;
3       printf("v = %d, uv = %u\n", v, uv);
```

When run on a two's-complement machine, it generates the following output:

```
v = -12345, uv = 53191
```

C data type	Minimum	Maximum
[signed] char	−128	127
unsigned char	0	255
short	−32,768	32,767
unsigned short	0	65,535
int	−2,147,483,648	2,147,483,647
unsigned	0	4,294,967,295
long	−2,147,483,648	2,147,483,647
unsigned long	0	4,294,967,295
int32_t	−2,147,483,648	2,147,483,647
uint32_t	0	4,294,967,295
int64_t	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint64_t	0	18,446,744,073,709,551,615

Figure 2.9 Typical ranges for C integral data types for 32-bit programs.

C data type	Minimum	Maximum
[signed] char	−128	127
unsigned char	0	255
short	−32,768	32,767
unsigned short	0	65,535
int	−2,147,483,648	2,147,483,647
unsigned	0	4,294,967,295
long	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
unsigned long	0	18,446,744,073,709,551,615
int32_t	−2,147,483,648	2,147,483,647
uint32_t	0	4,294,967,295
int64_t	−9,223,372,036,854,775,808	9,223,372,036,854,775,807
uint64_t	0	18,446,744,073,709,551,615

Figure 2.10 Typical ranges for C integral data types for 64-bit programs.

specifier	Output	Example
d *or* i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (uppercase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/exponent), lowercase	3.9265e+2
E	Scientific notation (mantissa/exponent), uppercase	3.9265E+2
g	Use the shortest representation: %e or %f	392.65
G	Use the shortest representation: %E or %F	392.65
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a pointer to a signed int. The number of characters written so far is stored in the pointed location.	
%	A % followed by another % character will write a single % to the stream.	%

	specifiers						
length	**d i**	**u o x X**	**f F e E g G a A**	**c**	**s**	**p**	**n**
(none)	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*
l	long int	unsigned long int		wint_t	wchar_t*		long int*
ll	long long int	unsigned long long int					long long int*
j	intmax_t	uintmax_t					intmax_t*
z	size_t	size_t					size_t*
t	ptrdiff_t	ptrdiff_t					ptrdiff_t*
L			long double				

C declaration		Bytes	
Signed	Unsigned	32-bit	64-bit
[signed] char	unsigned char	1	1
short	unsigned short	2	2
int	unsigned	4	4
long	unsigned long	4	8
int32_t	uint32_t	4	4
int64_t	uint64_t	8	8
char*		4	8
float		4	4
double		8	8