a converted value of 4,294,967,286. The comparison `i < u` will therefore produce 0. Some compilers produce a warning message such as "*comparison between signed and unsigned*" when a program attempts to compare a signed number with an unsigned number.

Because of traps like this one, it's best to use unsigned integers as little as possible and, especially, never mix them with signed integers.

The following example shows the usual arithmetic conversions in action:

```
char c;
short int s;
int i;
unsigned int u;
long int l;
unsigned long int ul;
float f;
double d;
long double ld;

i = i + c;      /* c is converted to int               */
i = i + s;      /* s is converted to int               */
u = u + i;      /* i is converted to unsigned int      */
l = l + u;      /* u is converted to long int          */
ul = ul + l;    /* l is converted to unsigned long int */
f = f + ul;     /* ul is converted to float            */
d = d + f;      /* f is converted to double            */
ld = ld + d;    /* d is converted to long double       */
```

Conversion During Assignment

The usual arithmetic conversions don't apply to assignment. Instead, C follows the simple rule that the expression on the right side of the assignment is converted to the type of the variable on the left side. If the variable's type is at least as "wide" as the expression's, this will work without a snag. For example:

```
char c;
int i;
float f;
double d;

i = c;    /* c is converted to int    */
f = i;    /* i is converted to float */
d = f;    /* f is converted to double */
```

Other cases are problematic. Assigning a floating-point number to an integer variable drops the fractional part of the number:

```
int i;

i = 842.97;     /* i is now 842 */
i = -842.97;    /* i is now -842 */
```