```
    first_sum = d + i2 + i4 + j1 + j3 + j5;
    second_sum = i1 + i3 + i5 + j2 + j4;
    total = 3 * first_sum + second_sum;

    printf("Check digit: %d\n", 9 - ((total - 1) % 10));

    return 0;
}
```

Note that the expression `9 - ((total - 1) % 10)` could have been written as `9 - (total - 1) % 10`, but the extra set of parentheses makes it easier to understand.

4.2 Assignment Operators

Once the value of an expression has been computed, we'll often need to store it in a variable for later use. C's = (*simple assignment*) operator is used for that purpose. For updating a value already stored in a variable, C provides an assortment of compound assignment operators.

Simple Assignment

The effect of the assignment *v* = *e* is to evaluate the expression *e* and copy its value into *v*. As the following examples show, *e* can be a constant, a variable, or a more complicated expression:

```
i = 5;              /* i is now 5   */
j = i;              /* j is now 5   */
k = 10 * i + j;     /* k is now 55  */
```

If *v* and *e* don't have the same type, then the value of *e* is converted to the type of *v* as the assignment takes place:

```
int i;
float f;

i = 72.99f;     /* i is now 72 */
f = 136;        /* f is now 136.0 */
```

conversion during assignment ➤ 7.4

We'll return to the topic of type conversion later.

In many programming languages, assignment is a *statement*; in C, however, assignment is an *operator*, just like +. In other words, the act of assignment produces a result, just as adding two numbers produces a result. The value of an assignment *v* = *e* is the value of *v after* the assignment. Thus, the value of `i = 72.99f` is 72 (not 72.99).