

2.6 Memory Concepts

Variable names such as `number1`, `number2` and `sum` actually correspond to *locations* in the computer's memory. Every variable has a **name**, a **type**, a **size** (in bytes) and a **value**.

In the addition program of [Fig. 2.7](#), when the following statement (line 12) executes:

```
int number1 = input.nextInt(); // read first number f
```



the number typed by the user is placed into a memory location corresponding to the name `number1`. Suppose that the user enters 45. The computer places that integer value into location `number1` ([Fig. 2.8](#)), replacing the previous value (if any) in that location. The previous value is lost, so this process is said to be *destructive*.



Fig. 2.8

Memory location showing the name and value of variable

number1.

When the statement (line 15)

```
int number2 = input.nextInt(); // read second number
```

executes, suppose that the user enters 72. The computer places that integer value into location `number2`. The memory now appears as shown in [Fig. 2.9](#).

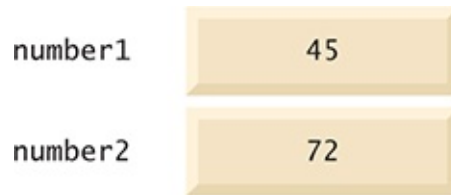


Fig. 2.9

Memory locations after storing values for `number1` and `number2`.

After the program of [Fig. 2.7](#) obtains values for `number1` and `number2`, it adds the values and places the total into variable `sum`. The statement (line 17)

```
int sum = number1 + number2; // add numbers, then sto
```

performs the addition, then replaces any previous value in

sum. After sum has been calculated, memory appears as shown in Fig. 2.10. The values of `number1` and `number2` appear exactly as they did before they were used in the calculation of `sum`. These values were used, but *not* destroyed, as the computer performed the calculation. When a value is read from a memory location, the process is *nondestructive*.

number1	45
number2	72
sum	117

Fig. 2.10

Memory locations after storing the sum of `number1` and `number2`.