Example Since the Hack language is self-explanatory, we start with an example. The only non-obvious command in the language is *@value*, where *value* is either a number or a symbol representing a number. This command simply stores the specified value in the A register. For example, if `sum` refers to memory location 17, then both `@17` and `@sum` will have the same effect: `A←17`.

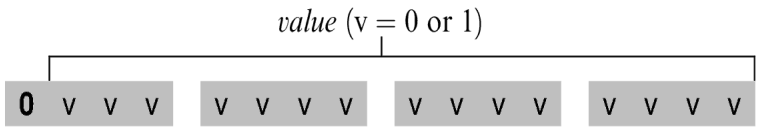
And now to the example: Suppose we want to add the integers 1 to 100, using repetitive addition. Figure 4.2 gives a C language solution and a possible compilation into the Hack language.

Although the Hack syntax is more accessible than that of most machine languages, it may still look obscure to readers who are not familiar with low-level programming. In particular, note that every operation involving a memory location requires two Hack commands: One for selecting the address on which we want to operate, and one for specifying the desired operation. Indeed, the Hack language consists of two generic instructions: an *address instruction*, also called *A*-instruction, and a *compute instruction*, also called *C*-instruction. Each instruction has a binary representation, a symbolic representation, and an effect on the computer, as we now specify.

4.2.2 The *A*-Instruction

The *A*-instruction is used to set the A register to a 15-bit value:

A-instruction: *@value* // Where *value* is either a non-negative decimal number
// or a symbol referring to such number.



This instruction causes the computer to store the specified value in the A register. For example, the instruction `@5`, which is equivalent to `0000000000000101`, causes the computer to store the binary representation of 5 in the A register.

The *A*-instruction is used for three different purposes. First, it provides the only way to enter a constant into the computer under program control. Second, it sets the stage for a subsequent *C*-instruction designed to manipulate a certain data memory location, by first setting A to the address of that location. Third, it sets the stage for a subsequent *C*-instruction that specifies a jump, by first loading the address of the jump destination to the A register. These uses are demonstrated in figure 4.2.