Reading and Writing Characters using getchar and putchar

Q&A C provides other ways to read and write single characters. In particular, we can use the getchar and putchar functions instead of calling scanf and printf. putchar writes a single character:

```
putchar(ch);
```

Each time getchar is called, it reads one character, which it returns. In order to save this character, we must use assignment to store it in a variable:

```
ch = getchar();  /* reads a character and stores it in ch */
```

getchar actually returns an int value rather than a char value (the reason will be discussed in later chapters). As a result, it's not unusual for a variable to have type int rather than char if it will be used to store a character read by getchar. Like scanf, getchar doesn't skip white-space characters as it reads.

Using getchar and putchar (rather than scanf and printf) saves time when the program is executed. getchar and putchar are fast for two reasons. First, they're much simpler than scanf and printf, which are designed to read and write many kinds of data in a variety of formats. Second, getchar and putchar are usually implemented as macros for additional speed. macros ➤*14.3*

getchar has another advantage over scanf: because it returns the character that it reads, getchar lends itself to various C idioms, including loops that search for a character or skip over all occurrences of a character. Consider the scanf loop that we used to skip the rest of an input line:

```
do {
  scanf("%c", &ch);
} while (ch != '\n');
```

Rewriting this loop using getchar gives us the following:

```
do {
  ch = getchar();
} while (ch != '\n');
```

Moving the call of getchar into the controlling expression allows us to condense the loop:

```
while ((ch = getchar()) != '\n')
  ;
```

This loop reads a character, stores it into the variable ch, then tests if ch is not equal to the new-line character. If the test succeeds, the loop body (which is empty) is executed, then the loop test is performed once more, causing a new character to be read. Actually, we don't even need the ch variable; we can just compare the return value of getchar with the new-line character: