5. If `Ch` is a variable of type `Char` or subrange of `Char`, then `Read(F,Ch)` assigns the character at the current position of file `F` or the value of `F↑` to `Ch`, followed by a `Get(F)`, the choice being implementation-dependent.

6. If a parameter `V` is of type `Integer` or a subrange of `Integer` then `Read` accepts a sequence of characters forming a signed integer with possible leading blanks. The integer value denoted by this sequence is then assigned to `V`.

7. If a parameter `V` is of type `Real`, `Read` accepts a sequence of characters forming a signed number with possible leading blanks. The real value denoted by this sequence is then assigned to `V`.

In scanning `F` (skipping blanks) to read numbers, `Read` may also skip end–of–line markers. `F` is left positioned to the non–digit character following the last digit constituting a number. To correctly read consecutive numbers, separate them by blanks or put them on separate lines. `Read` accepts the longest sequence of digits, and if two numbers are not separated, `Read` cannot distinguish them as two numbers (and neither can people!)

Examples:

Read and process a sequence of numbers where the last value is immediately followed by an asterisk. Assume `F` to be a textfile, `X` and `Ch` to be variables of types `Integer` (or `Real`) and `Char` respectively.

```
Reset(F);
repeat
  Read(F,X,Ch);
  P(X)
until Ch = '*'
```

Perhaps a more common situation is when there is no way of knowing how many data items are to be read, and there is no special symbol that terminates the list. Two convenient schemata are shown below. They make use of procedure `SkipBlanks`:

```
procedure SkipBlanks(var F: Text);
  var Done: Boolean;
begin
  Done := False;
```