The `inspect` Construct

November 13, 2015

An `inspect` loop is an iterator that automatically avoids errors. The syntax is as follows.

```
inspect (input_file, delimiter) {
  // Execute these statements on each input unit that allows successful execution.
  // These statements may contain function calls.
}
```

To illustrate the usage, we first present a small example. Say that we would like to collect the leading letters from all lines in a document. We would like to process the lines that start with alphabetic letters and to ignore the lines that start with other characters. Figure 1 presents sample input and output. Figure 2 compares two solutions, where Figure 2a uses traditional constructs and Figure 2b uses the `inspect` construct.

The program in Figure 2b works as follows. It first associates `f` to an input file named "data". The file consists of smaller input units, specifically, lines. The program starts execution from the `main` function, where an `inspect` loop iterates over the lines in file `f` until reaching the end of the file. For each line, the `inspect` loop automatically determines whether or not to execute the loop body based on whether or not the execution would succeed. Specifically,

- If a line starts with an alphabetic letter, the `inspect` loop executes the body which prints this leading letter.
- If a line does not start with an alphabetic letter, the `inspect` loop does not execute the body, so that there is no assertion failure.

In general, `inspect` loops iterate over delimited input units, skipping the units that would trigger errors. Consequently, each input unit is either successfully processed or completely ignored. The semantics use the following two criteria.

- **Delimiters:** A specified delimiter decomposes the input into smaller input units. An input unit is the contents between two delimiters. Each time the `inspect` body starts, a fresh input unit is available to be read.
- **Errors:** The `inspect` loop automatically skips the input units that would trigger errors if processed. Errors include assertion violations, arithmetic errors, array-access errors, file-access errors, attempts to read beyond input units, and resource exhaustion.

For completeness, please refer to other language constructs in a separate manual.

```
Hello,
world
!
```

(a) Sample input

```
Hw
```

(b) Sample output

Figure 1: Sample input and output

```
f = open("data");
main {
  while (!end(f)) {
    x = read(f);
    if ((x>='a' && x<='z') || (x>='A' && x<='Z')) {
      print(x);
    }
    while (!end(f) && x!='\n') {
      x = read(f);
    }
  }
  return 0;
}
```

(a) Traditional solution

```
f = open("data");
main {
  inspect (f, '\n') {
    x = read(f);
    assert((x>='a' && x<='z') || (x>='A' && x<='Z'));
    print(x);
  }
  return 0;
}
```

(b) `inspect` solution

Figure 2: Solutions