String scanning—`upto(cs)`, continued

Consider a program to divide lines like this:

```
abc=1;xyz=2;pqr=xyz;
```

into pairs of names and values.

```
procedure main()
    while line := read() do {
        line ? while name := tab(upto('=')) do {
          move(1)
          value := tab(upto(';'))
          move(1)
          write("Name: ", name, ", Value: ",
            value)
          }
        write()
        }
end
```

Interaction:

```
abc=1;xyz=2;pqr=xyz;
Name: abc, Value: 1
Name: xyz, Value: 2
Name: pqr, Value: xyz

a=1;b=2
Name: a, Value: 1
Name: b, Value: 1
```

What's wrong?

How can it be fixed?

Pitfall: incomplete scope

A scanning expression with an incomplete scope can produce a baffling bug.

Consider a routine to cut a string into pieces of length `n`, and produce a list of the results:

```
procedure cut(s, n)
    L := []

    s ? while put(L, move(n)) # get next n chars
    put(L, tab(0))            # add leftover

    return L
end
```

Execution:

```
][ cut(&lcase, 10);
   r := L1:["abcdefghij","klmnopqrst",""]
```

Solution:

```
procedure cut(s, n)
    L := []
    s ? {
        while put(L, move(n))
        put(L, tab(0))
      }
    return L
end
```

Underlying mechanism: Scanning expressions can be nested. Exiting a scanning expression restores the previous values of `&pos` and `&subject`. (Initially `1` and `""`, respectively.)

Review

Review of string scanning thus far:

Scanning operator:

```
expr1 ? expr2
```

Sets `&subject` to the value of `expr1` and sets `&pos` to 1. When `expr2` terminates, the previous values of `&subject` and `&pos` are restored.

Functions for changing `&pos`:

`move(n)`	relative adjustment; string result
`tab(n)`	absolute adjustment; string result

Functions typically used in conjunction with `tab(n)`:

`many(cs)` produces position after run of characters in `cs`.

`upto(cs)` generates positions of characters in `cs`

Pitfalls:

`many(cs)` fails if the next character is not in `cs`.

Short scope on scanning expression causes unexpected restoration of prior `&subject` and `&pos` values.

String scanning examples

A procedure to compress a series of dots into a single dot:

```
procedure compress(s)
    r := ""
    s ? {
        while r ||:= tab(upto('.')+1) do
            tab(many('.'))
        r ||:= tab(0)
        }

    return r
end
```

A test program:

```
procedure main()
  while ln := (writes("String? ") & read()) do {
     write(compress(ln))
     write()
     }
end
```

Interaction:

```
String? a..test...right......here
a.test.right.here

String? ..testing.....
.testing.

String? .......
.
```

String scanning examples, continued

Problem: Write a procedure `rmchars(s, c)` that removes all characters in `c` from `s`. Example:

```
][ rmchars("a test here", 'aieou');
   r := " tst hr"  (string)

][ rmchars("a test here", &letters);
   r := "  "  (string)
```

Problem: Write a procedure `keepchars(s, c)` that returns a copy of `s` consisting of only the characters in `c`.

```
][ keepchars("(520) 577-6431", &digits);
   r := "5205776431"  (string)
```

String scanning examples, continued

Problem: Write a routine `expand(s)` that does simple run-length expansion:

```
][ expand("x3y4z");
   r := "xyyyzzzz"  (string)

][ expand("5ab0c");
   r := "aaaaab"  (string)

][ *expand("1000a1000bc");
   r := 2001  (integer)
```

Assume the input is well-formed.

String scanning examples, continued

Problem: Write a procedure `fname(path)` that accepts a UNIX path name such as `/x/y/z.c`, `../a/b/.init`, or `test_net`, and returns the file name component.

Problem: Make up a string scanning problem and solve it.

String scanning examples, continued

Problem: Write a program that reads the output of the `who` command and produces a list of users sorted by originating host.

Once upon a time, `who` output looked like this:

```
whm         pts/1        Feb 21 19:54    (mesquite.CS.Arizona.EDU)
cpilson     pts/228      Feb 21 20:30    (tuc-ts1-8.goodnet.com)
nicko       pts/62       Feb 20 07:44    (raleigh.CS.Arizona.EDU)
deepakl     pts/2        Feb 20 00:17    (italic.CS.Arizona.EDU)
ilwoo       pts/7        Feb 15 04:51    (folio.CS.Arizona.EDU)
siva        pts/135      Feb 21 21:37    (pug.CS.Arizona.EDU)
rajesh      pts/9        Feb 14 14:24    (astra.CS.Arizona.EDU)
muth        pts/8        Feb 19 09:18    (granjon.CS.Arizona.EDU)
butts       pts/111      Feb 21 20:41    (nomi)
ganote      pts/153      Feb 21 20:25    (lectura.CS.Arizona.EDU)
...
```

Desired output format:

```
rajesh          astra.CS.Arizona.EDU
ilwoo           folio.CS.Arizona.EDU
muth            granjon.CS.Arizona.EDU
deepakl         italic.CS.Arizona.EDU
ganote          lectura.CS.Arizona.EDU
whm             mesquite.CS.Arizona.EDU
butts           nomi
siva            pug.CS.Arizona.EDU
nicko           raleigh.CS.Arizona.EDU
cpilson         tuc-ts1-8.goodnet.com
```

Restriction: You can't use `sortf`.

String scanning examples, continued

`who` output format:

```
whm       pts/1        Feb 21 19:54    (mesquite.CS.Arizona.EDU)
```

A solution:

```
    procedure main()
        who := open("who", "rp") # open pipe to read

        lines := []
        while line := read(who) do {
            line ? {
                user := tab(many(~' '))
                tab(many(' '))              # (A)
                term := tab(many(~' '))
                tab(many(' '))
                time := move(12)            # (B)
                tab(upto('(') + 1)
                sys := tab(upto(')'))
                }
            put(lines, sys || "\x00" ||
                left(user,15) || sys)
            }

        every line := !sort(lines) do
          line ? {
            tab(upto('\x00')+1)
            write(tab(0))
            }
    end
```

Shortcut: Since `term` and `time` aren't used, lines (A) through (B) could be deleted.