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.

```pascal
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()
  }
en
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:

```plaintext
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:

```plaintext
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:

\[
\text{expr1} \ ? \ \text{expr2}
\]

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:

- \text{move}(n) \quad \text{relative adjustment; string result}
- \text{tab}(n) \quad \text{absolute adjustment; string result}

Functions typically used in conjunction with \text{tab}(n):

- \text{many}(cs) \quad \text{produces position after run of characters in } cs.
- \text{upto}(cs) \quad \text{generates positions of characters in } cs

Pitfalls:

- \text{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:

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

    return r 
end
```

A test program:

```plaintext
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 \texttt{rmchars(s, c)} that removes all characters in \(c\) from \(s\). Example:

\[
\begin{array}{l}
\text{rmchars("a test here", 'aieou');} \\
\quad r := "tst hr" \quad \text{(string)} \\
\end{array}
\]

\[
\begin{array}{l}
\text{rmchars("a test here", &letters);} \\
\quad r := " " \quad \text{(string)} \\
\end{array}
\]

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

\[
\begin{array}{l}
\text{keepchars("(520) 577-6431", &digits);} \\
\quad r := "5205776431" \quad \text{(string)} \\
\end{array}
\]
String scanning examples, continued

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

```plaintext
[[ expand("x3y4z") ;
  r := "xyyzzzzz"  (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:

<table>
<thead>
<tr>
<th>Name</th>
<th>Session</th>
<th>Date</th>
<th>Time</th>
<th>Hostname</th>
</tr>
</thead>
<tbody>
<tr>
<td>whm</td>
<td>pts/1</td>
<td>Feb 21</td>
<td>19:54</td>
<td>mesquite.CS.Arizona.EDU</td>
</tr>
<tr>
<td>cpilson</td>
<td>pts/228</td>
<td>Feb 21</td>
<td>20:30</td>
<td>tuc-tsl-8.goodnet.com</td>
</tr>
<tr>
<td>nicko</td>
<td>pts/62</td>
<td>Feb 20</td>
<td>07:44</td>
<td>raleigh.CS.Arizona.EDU</td>
</tr>
<tr>
<td>deepakl</td>
<td>pts/2</td>
<td>Feb 20</td>
<td>00:17</td>
<td>italic.CS.Arizona.EDU</td>
</tr>
<tr>
<td>ilwoo</td>
<td>pts/7</td>
<td>Feb 15</td>
<td>04:51</td>
<td>folio.CS.Arizona.EDU</td>
</tr>
<tr>
<td>siva</td>
<td>pts/135</td>
<td>Feb 21</td>
<td>21:37</td>
<td>pug.CS.Arizona.EDU</td>
</tr>
<tr>
<td>rajesh</td>
<td>pts/9</td>
<td>Feb 14</td>
<td>14:24</td>
<td>astra.CS.Arizona.EDU</td>
</tr>
<tr>
<td>muth</td>
<td>pts/8</td>
<td>Feb 19</td>
<td>09:18</td>
<td>granjon.CS.Arizona.EDU</td>
</tr>
<tr>
<td>butts</td>
<td>pts/111</td>
<td>Feb 21</td>
<td>20:41</td>
<td>nomi</td>
</tr>
<tr>
<td>ganote</td>
<td>pts/153</td>
<td>Feb 21</td>
<td>20:25</td>
<td>lectura.CS.Arizona.EDU</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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-tsl-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.
```