String scanning basics

Icon's string scanning facility is used for analysis of strings.

The string scanning facility allows string analysis operations to be intermixed with general computation.

String scanning is initiated with `?`, the scanning operator:

```
expr1 ? expr2
```

The value of `expr1` is established as the subject of the scan (`&subject`) and the scanning position in the subject (`&pos`) is set to 1. `expr2` is then evaluated.

```
][ "testing" ? { write(&subject); write(&pos) };
testing
1
   r := 1  (integer)
```

The result of the scanning expression is the result of `expr2`.

The procedure `snap()` displays `&subject` and `&pos`:

```
][ "testing" ? snap();
&subject =  t e s t i n g
&pos =  1  |
   r := &null  (null)
```

String scanning—`move(n)`

The built-in function `move(n)` advances `&pos` by `n` and returns the substring of `&subject` between the old and new values of `&pos`. `move(n)` fails if `n` is too large.

```
    ][ "testing" ? {
    ...     snap()
    ...     move(1)
    ...     snap()
    ...     write(move(2))
    ...     snap()
    ...     write(move(2))
    ...     snap()
    ...     write(move(10))
    ...     snap()
    ...     };

first snap():
    &subject =  t e s t i n g
    &pos =  1  |

move(1):
    &subject =  t e s t i n g
    &pos =  2    |

write(move(2)):
    es
    &subject =  t e s t i n g
    &pos =  4       |

write(move(2)):
    ti
    &subject =  t e s t i n g
    &pos =  6           |

write(move(10)):
    &subject =  t e s t i n g
    &pos =  6           |
```

String scanning—`move(n)`, continued

`&pos` can be thought of as a scanning "cursor". `move(n)` adjusts `&pos` (the cursor) by `n`, which can be negative.

A scanning expression that iterates:

```
][ "testing" ? while move(1) do {
...          snap()
...          write(move(1))
...          };

&subject =  t e s t i n g
&pos =  2    |
e

&subject =  t e s t i n g
&pos =  4        |
t

&subject =  t e s t i n g
&pos =  6            |
n

&subject =  t e s t i n g
&pos =  8                |
Failure
```

Negative movement:

```
][ "testing" ? { move(5); snap();
...              write(move(-3)); snap()};
&subject =  t e s t i n g
&pos =  6            |
sti
&subject =  t e s t i n g
&pos =  3      |
```

String scanning—`move(n)`, continued

Example: segregation of characters in odd and even positions:

```
][ ochars := echars := "";
   r := ""  (string)

][ "12345678" ? while ochars ||:= move(1) do
...                  echars ||:= move(1);
Failure

][ ochars;
   r := "1357"  (string)

][ echars;
   r := "2468"  (string)
```

Does this work properly with an odd number of characters in the subject string? How about an empty string as the subject?

String scanning—tab(n)

The built-in function tab(n) sets &pos to n and returns the substring of &subject between the old and new positions. tab(n) fails if n is too large.

```
][ "a longer example" ? {
...     write(tab(4))
...     snap()
...     write(tab(7))
...     snap()
...     write(tab(10))
...     snap()
...     write(tab(0))
...     snap()
...     write(tab(12))
...     snap()
...     };
a l   (write(tab(4))
&subject =  a   l o n g e r   e x a m p l e
&pos =  4       |

ong   (write(tab(7))
&subject =  a   l o n g e r   e x a m p l e
&pos =  7           |

er    (write(tab(10))
&subject =  a   l o n g e r   e x a m p l e
&pos = 10                 |

example (write(tab(0))
&subject =  a   l o n g e r   e x a m p l e
&pos = 17                                 |

ample (write(tab(12))
&subject =  a   l o n g e r   e x a m p l e
&pos = 12                     |
```

String scanning—`move` vs. `tab`

Be sure to understand the distinction between `tab` and `move`:

Use `tab` for absolute positioning.

Use `move` for relative positioning.

Example:

```
][ &lcase ? { write(tab(3)); write(tab(3));
...           write(move(3)); write(move(3)) };
ab

cde
fgh
```

String scanning—`many(cs)`

The built-in function `many(cs)` looks for one or more occurrences of the characters in the character set `cs`.

`many(cs)` returns the position of the end of a run of one or more characters in `cs`, starting at `&pos`.

For reference:

```
  x   x   y   z   .   .   .
|   |   |   |   |   |   |   |
1   2   3   4   5   6   7   8
```

`many` in operation:

```
][ "xxyz..." ? many('x');
   r := 3  (integer)

][ "xxyz..." ? many('xyz');
   r := 5  (integer)

][ "xxyz..." ? many('xyz.');
   r1 := 8  (integer)

][ "xxyz..." ? { move(2); many('yz') };
   r2 := 5  (integer)
```

Note that `many(cs)` fails if the next character is not in `cs`:

```
][ "xxyz..." ? many('.');
Failure

][ "xxyz..." ? { move(1); many('yz') };
Failure
```

`many(cs)`, continued

`many` is designed to work with `tab`—`many` produces an absolute position in a string and `tab` sets `&pos`, the cursor, to an absolute position.

For reference:

```
  x   x   y   z   .   .   .
|   |   |   |   |   |   |   |
1   2   3   4   5   6   7   8
```

`many` and `tab` work together:

```
][ "xxyz..." ? { p := many('xyz'); tab(p);
                                   snap() };
&subject =  x x y z . . .
&pos =  5          |

][ "xxyz..." ? { tab(many('xyz')); snap() };
&subject =  x x y z . . .
&pos =  5          |

][ "xxyz..." ? { tab(many('xyz') + 2); snap()};
&subject =  x x y z . . .
&pos =  7              |
```

Sometimes it is better to describe what is <u>not</u> being looked for:

```
][ "xxyz..." ? { tab(many(~'.')); snap() };
&subject =  x x y z . . .
&pos =  5          |
```

String scanning—`upto(cs)`

The built-in function `upto(cs)` generates the positions in `&subject` where a character in the character set `cs` occurs.

```
][ "bouncer" ? every write(upto('aeiou'));
2
3
6
Failure
```

A loop to print out vowels in a string:

```
][ "bouncer" ? every tab(upto('aeiou')) do
...              write(move(1));
o
u
e
Failure
```

A program to read lines and print vowels:

```
procedure main()
    while line := read() do {
        line ? every tab(upto('aeiou')) do
                    write(move(1))
        }
end
```

When should `upto` be used with `move`, rather than `tab`?

upto vs. many

An attempt at splitting a string into pieces:

```
][ "ab.c.xyz" ? while write(tab(upto('.'))) do
...                       move(1);
ab
c
Failure
```

A solution that works:

```
][ "ab.c.xyz" ? while write(tab(many(~'.'))) do
...                       move(1);
ab
c
xyz
Failure
```

How could we make a list of the pieces?

How could we handle many dots, e.g., "ab...c..xyz"?

How could the upto('.') approach be made to work?