String scanning—`find(s)`

The built-in function `find(s)` generates the positions in `&subject` (starting at `&pos`) where the string `s` begins.

```
][ "infringing on infinity" ? {
...   every posn := find("in") do
...     write(posn)
...   };
1
5
8
15
18
Failure
```

A fragment to print lines on standard input that contain "error" at or beyond position 10:

```
while line := read() do {
    line ? if (p := find("error")) >= 10 then
                write("Found at ", p)
           else
                write("Not found")
    }
```

Interaction:

```
1234567890
Not found
an error here
Not found
here is another error
Found at 17
error error error
Found at 13
```


String scanning—`find(s)`, continued

A different approach for the previous example:

```
while line := read() do {
    line ? if tab(10) & p := find("error") then
                write("Found at ", p)
           else
                write("Not found")
    }
```

Problem: Write a program `anyof` to print lines that contain any of the strings named as command line arguments. Example:

```
% anyof read write < (code above)
while line := read() do {
                write("Found at ", p)
                write("Not found")
```

String scanning—`find(s)`, continued

A routine to replace one text string with another:

```
procedure replace(s, from_str, to_str)
  new_str := ""

  s ? {
     while new_str ||:= tab(find(from_str)) do {
        new_str ||:= to_str
        move(*from_str)
        }
     new_str ||:= tab(0)
     }

    return new_str
end
```

Example:

```
replace("to be or not to be", "be", "eat")
```


Standalone use of `find`

`find` actually accepts four arguments:

```
find(s1, s2, i1, i2)
```

It generates the locations of `s2` between positions `i1` and `i2` where `s1` occurs. These defaults are used:

```
s2  &subject
i1  &pos if s2 defaulted, 1 otherwise
i2  0
```

Example:

```
][ every write(find("in", "infinite"));
1
4
Failure
```

Another version of the `anyof` program:

```
procedure main(args)
    while line := read() do {
            if find(!args,line) then
                write(line)
        }
end
```

String scanning—`match(s)`

The built-in function `match(s)` succeeds if the string `s` appears next.

```
][ "infinite" ? match("in");
   r := 3  (integer)

][ "infinite" ? tab(match("in"));
   r := "in"  (string)

][ "finite" ? tab(match("in"));
Failure

][ "finite" ? { move(3); tab(match("it"));
                write(tab(0)) };
e
   r := "e"  (string)
```

The expression `tab(match(s))` is very common; `=s` is a synonym for it:

```
][ "infinite" ? ="in";
   r := "in"  (string)

][ "mat" ? =(!"cmb"||"at");
   r := "mat"  (string)
```

Like `find`, `match` accepts four arguments, with defaults for the last three. It is commonly used to see if a string is a prefix of another:

```
if match("procedure"|"global", line) then ...
```


Problem: Comment stripper

Write a program that strips comments from Java source code. It should handle both forms (`//` and `/* ... */`). Ignore the potential of string literals having the sequences of interest.

String scanning—`pos(n)`

The built-in function `pos(n)` succeeds if `&pos` is equivalent to `n`. Either a right- or left-running position may be specified.

Here is a program that reads standard input and prints non-blank lines:

```
procedure main()
    while line := read() do
        line ?
          if not (tab(many(' \t')) & pos(0)) then
             write(line)
end
```

Question: Is the `pos` function really needed? Why not just compare to `&pos`?

Problem: What are two shorter solutions that don't use scanning?

String scanning—`any(cs)`

The built-in function `any(cs)` succeeds if the next character is in the character set `cs`. `&pos+1` is returned if successful.

A procedure to see if a string consists of a digit followed by a capital letter, followed by a digit:

```
procedure NCN(s)
    s ? {
        *s = 3 &
        tab(any(&digits)) &
        tab(any(&ucase)) &
        tab(any(&digits)) &
        return }
end
```

A driver:

```
while line := (writes("String? ") & read()) do
    if NCN(line) then
        write("ok")
    else
        write("not ok")
```

Interaction:

```
String? 8X1
ok
String? 9x2
not ok
String? 4F22
not ok
```

Question: How could `pos()` be used in this procedure?

Summary of string scanning functions

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`
`find(s)`	generates positions of `s`
`match(s)`	produces position after `s`, if `s` is next
`any(cs)`	produces position after a character in `cs`

Other functions:

`pos(n)` tests if `&pos` is equivalent to `n`

`bal(s, cs1, cs2, cs3)`

similar to `upto(cs)`, but used for working with "balanced" strings. (Not covered; included for completeness.)

The functions `any`, `find`, `many`, `match`, and `upto` each accept four arguments, the last three of which default:

```
<fcn>(s1, s2, i1, i2)
```


Problem: `is_assign(s)`

Problem: Write a procedure `is_assign(s)` that succeeds iff `s` has the form `<identifier>=<integer>`.

```
][ is_assign("x4=10");
   r := 6  (integer)

][ is_assign("4=10");
Failure

][ is_assign("abc=10x");
Failure

][ is_assign("_123=456");
   r := 9  (integer)

][ is_assign("_123 = 456");
Failure
```

split.icn

This is the source for `split`:

```
procedure split(s, dlms, keepall)
   local w, ws, addproc, nullproc

   ws := []
   /dlms := ’ \t’

   addproc := put
   if \keepall then
      otherproc := put
   else
      otherproc := 1

   if dlms := (any(dlms, s[1]) & ~dlms) then
      otherproc :=: addproc

   s ? while w := tab(many(dlms := ~dlms)) do {
      addproc(ws, w)
      otherproc :=: addproc
      }

   return ws
end
```

Two test cases:

```
"  just  a   test   right  here  "


"while w := tab(many(dlms := ~dlms)) do"
```