

CSc 372 — Comparative Programming Languages

34 : Icon — String Scanning

Christian Collberg
Department of Computer Science
University of Arizona
collberg+372@gmail.com

Copyright © 2005 Christian Collberg

November 30, 2005

String Parsing

1 find

- `find(x,S)` generates all the positions in `S` where the string `x` occurs.

```
][ S := "hello world";  
][ .every find("l",S);  
  3  
  4  
 10
```

2 find...

- Beware that when a string “changes”, there’s actually a new string constructed.

```
][ S := "axaxa";  
][ every i := find("x",S) do {  
    write(i); S[i]:="yy"; write(S)  
  };  
  2  
 ayyaxa  
  4  
 ayyyyxa
```

3 Removing Nested Comments

- Idea: repeatedly remove any comments that don't contain any other comments.

```
procedure decomment(S);
  while (1) do {
    if f := find("/*",S) &
       t := find("*/",S,f+2) &
       not (find("/*",S,f+2) < t) &
       not (find("*/",S,f+2) < t) then
      S[f:t+2] := ""
    else
      break
  }
  return S;
end
```

4 Nested Comments...

```
procedure main()
  write(decomment("/* hello world */"))
  write("----")
  write(decomment("foo /* hello world */ bar"))
  write("----")
  write(decomment("/* hello/* there */ world */"))
  write("----")
  write(decomment("foo /* hello/* there */ world */ bar"))
  write("----")
  write(decomment("foo /* hello */ there /* world */ bar"))
end
```

5 Nested Comments...

```
> icont comments.icn
> comments
```

```
---
foo bar
---
```

```
---
foo bar
---
```

```
foo there bar
```

6 csets

- A **cset** is a basic Icon type that describes sets of characters.
- Csets are written as a string of characters between single quotes.

- Predefined csets:
 - &digits:** digits between 0 to 9.
 - &letters:** all letters.
 - &ascii:** all ASCII characters
 - &lcase:** lower case letters.
 - &ucase:** upper case letters.
- The normal set operations can be performed using **++** (union), ****** (intersection), **--** (set difference), and **~** (complement).

7 csets...

- A string that occurs in a context where a cset is expected will be converted automatically.

```

][ write(&letters);
ABCDEFGHIJKLMNPOQRSTUVWXYZabcdefghijklmnopghij...
][ write(&ascii);
!"#$%&'()*+,-./0123456789:;<=>?@ABC...
][ x := 'abc123';
][ x ** &letters;
   r13 := 'abc' (cset)
][ "456" ++ x;
   r14 := '123456abc' (cset)

```

8 upto

- **upto(x,S)** generates all the positions in **S** where any of the characters in the cset **x** occur.

```

][ S := "hello world";
][ .every upto('l',S);
   3
   4
  10
][ write(upto('x',S));
Failure
][ every write(upto("l",S));
   3
   4
  10

```

9 many

- **many(x,S)** produces the position after the longest initial substring of **S** containing only characters in the cset **x**. **many(x,S)** fails if the first character of **S** isn't in **x**.

```

][ S := "hello 42 world";
][ write(many('hel',S));
   5
][ write(many('xyz',S));

```

```

Failure
][ write(many(&letters,S));
6
][ write(many(&letters++' ',S));
7
][ write(many('xyz',"bbbxxxxccc"));
Failure

```

10 any

- `any(x,S)` produces `2` if the first character in `S` is in the cset `x`, and fails otherwise.

```

][ S := "hello world";
][ write(any('hxl',S));
2
][ write(any('xl',S));
Failure

```

11 match

- `match(x,S)` succeeds if the string `x` is a prefix of `S`, and fails otherwise.
- On success, `match(x,s)` returns the position after `x`.

```

][ S := "hello world";
][ write(match("hell",S));
5
][ write(match("ell",S));
Failure
][ write(match("",S));
1
][ write(match(S,S));
12

```

12 Removing Whitespace

- Removing initial whitespace:

```

][ S := "  hello world";
][ S[1:many(' \t',S)] := "";
][ S;
r35 := "hello world"

```

String Scanning

13 String Scanning

- The expression `s ? e` makes `s` the subject to which string processing operations in `e` apply.
- The program below prints 3, 13, and 23:

```
line := "a fish is a fish is a fish"
every line ? write(find("fish"))
```

14 String Scanning...

- All the string manipulation functions above (`match`, `many`, etc.) can be used in string scanning.
- When we initiate a string scanning expression `s ? e`, Icon sets a special variable `&subject` to `s`, and another variable `&pos` (the current position) to 1.
- `match`, `many`, etc. operate directly on `&subject` and `&pos`.
- Note that `find` gets its argument implicitly:

```
][ "hi there" ? {write(&pos);write(&subject)};
1
hi there
][ "hi there" ? {write(find("th"))};
4
```

15 move

- `move(i)` advances the position by `i` characters.
- `move` returns the substring of the subject that is `matched` as a result of changing the position.
- The program below sets `t` to a string containing the characters of `line` followed by periods:

```
t := ""
line ? while t := t || move(1) || "."
```

16 Snapshots

- Use `snap()` in `ie` to show the current subject and position:

```
][ "hi there" ? {move(2);snap();move(3);snap()};
&subject = h i   t h e r e
&pos = 3     |
&subject = h i   t h e r e
&pos = 6     |
```

- You can do this in your own programs by saying `link scan` and calling the function `snapshot()`.

17 move...

```
][ "hi there" ? {s := move(3); snap(); write(s)};
&subject = h i   t h e r e
&pos = 4         |
hi
```

18 move...

- Split up a string in odd and even characters.

```
procedure sep(S)
  O := E := ""
  S ? while O ||:= move(1) & E ||:= move(1)
  suspend O | E
end
```

```
procedure main()
  every i := sep("a1b2c3d4e5") do write(i)
end
```

```
> icont sep.icn
> sep
abcde
12345
```

19 tab

- **tab(i)** moves to position **i** in the subject and returns the substring between the old and new positions.

```
][ "hi there" ? {s := tab(5); snap(); write(s)};
&subject = h i   t h e r e
&pos = 5         |
hi t
```

20 String Scanning Functions

- The other string scanning functions behave the same as previously shown, except that they operate on **&subject** and **&pos** implicitly.
- **upto(s)** returns the position of any of the characters in **s**, starting at the current position (**&pos**).
- **many(s)** returns the position following the longest possible substring containing only characters in **s** starting at the current position.

```
][ "xxyyxxxxzzz" ? {tab(5); write(many('x'))};
10
][ "abxxyzzz" ? {tab(4); every write(upto('xy'))};
4
5
6
```

21 Extracting Vowels

- Generate all the vowels in a string.

```
procedure vowels(S)
  S ? every tab(upto('aeiou')) do suspend move(1)
end
```

```
procedure main()
  every i := vowels("foobar") do write(i)
end
```

```
> iconv vowels.icn
> vowels
o
o
a
```

22 String Scanning Functions...

- **any(c)** succeeds if the first character in the subject string is in the cset **c**.

```
][ "booyah" ? {write(any('b'))};
2
][ "booyah" ? {write(any('c'))};
Failure
```

23 String Scanning Functions...

- **match(t)** succeeds if **t** matches the initial characters of the subject string and returns the position after the matched part.

```
][ "booyah" ? {write(match("boo"))};
4
  r33 := 4 (integer)
][ "booyah" ? {write(match("koo"))};
Failure
```

24 Combining String Scanning Functions

- It's common to combine **tab** and **move** with the other string scanning functions to extract pieces of text.

```
][ "booyah" ? {write(tab(match("boo"))); snap()};
boo
&subject = b o o y a h
&pos = 4      |
][ "xxx123yyy" ? {tab(many(&ascii--&digits));
                    snap()};
&subject = x x x 1 2 3 y y y
&pos = 4      |
```

```

r36 := &null (null)
][ "xxx123yyy" ? {tab(many(&ascii--&digits));
                    write(tab(many(&digits)))};
123

```

25 Combining String Scanning Functions

- `tab(match(S))` is so common that a shorthand has been created.
- `=S` returns the string `S` if it matches the beginning of `&subject`, and also moves `&pos` to the position after `S`.

```

][ "booyah" ? {write(="foo");snap()};
&subject = b o o y a h
&pos = 1 |
][ "booyah" ? {write(="boo"); snap()};
boo
&subject = b o o y a h
&pos = 4 |

```

26 Extracting Words

```

procedure getword(str)
    str ? while tab(upto(&letters)) do {
        word := tab(many(&letters))
        suspend word
    }
end

```

- `tab(upto(&letters))` advances the position up to the next letter.
- `tab(many(&letters))` matches the word and assigns it to `word`.
- The `while` terminates when `tab(upto(&letters))` fails because there are no more words in `str`.

27 Extracting Words...

- The program below lists the most commonly used words in its input and their frequencies of occurrence.

```

procedure main(args)
    k := integer(args[1]) | 10
    words := table(0)
    while line := read() do
        every words[getword(line)] += 1
    words := sort(words, 4)
    every 1 to k do
        write(pull(words), "\n", pull(words))
end

```


Summary

28 Summary — Position Functions

- These functions take strings or `csets` as arguments and either fail or return exactly one position in the string as result.

<code>any(c)</code>	<u>Returns</u> 2 if the first character in <code>&subject</code> is in the <code>cset c</code> .
<code>many(c)</code>	<u>Returns</u> the position following the longest initial substring of <code>&subject</code> consisting only of characters from the <code>cset c</code> .
<code>match(s)</code>	If the string <code>s</code> occurs at the beginning of <code>&subject</code> then <u>returns</u> the position following <code>s</code> .

29 Summary — Position Generators

- These functions take strings or `csets` as arguments and generate zero or more positions as results.

<code>find(s)</code>	<u>Generates</u> all the positions in <code>&subject</code> at which the string <code>s</code> occurs.
<code>upto(c)</code>	<u>Generates</u> all the positions in <code>&subject</code> containing characters from the <code>cset c</code> .

30 Summary — Position Movers

- These functions take a position as argument and move to a new position (if it exists), returning the substring from the initial to the new position as result.

<code>move(p)</code>	<u>Moves</u> <code>p</code> characters forward in <code>&subject</code> . <u>Returns</u> the substring which was passed over during the move.
<code>tab(p)</code>	<u>Moves</u> to position <code>p</code> in <code>&subject</code> . <u>Returns</u> the substring which was passed over during the move.

31 Examples — Position Functions

<code>"foo" ? any('f')</code>	Succeeds and returns 2.
<code>"foo" ? any('b')</code>	Fails.
<code>"oodles" ? many('od')</code>	Succeeds and returns 5.
<code>"noodles" ? many('od')</code>	Fails.
<code>"foobar" ? match("foo")</code>	Succeeds and returns 4.
<code>"boofar" ? match("foo")</code>	Fails.

32 Examples — Position Generators

<code>"fooboo" ? find("oo")</code>	Generates the positions {2,5}.
<code>"fooboo" ? find("aa")</code>	Fails.
<code>"foobar" ? upto('ao')</code>	Generates the positions {2,3,5}.
<code>"foobar" ? upto('xy')</code>	Fails.

33 Examples — Position Movers

`"foobar" ? write(move(3))` Moves three steps forward (i.e., sets `&pos:=&pos+3` (4)) and writes `"foo"`.

`"foobar" ? write(tab(3))` Sets `&pos` to 3 and writes `"fo"`.

34 Readings and References

- Read `Christopher`, pp. 53--55, 57--58.

35 Acknowledgments

- Some material on these slides has been modified from William Mitchell's Icon notes: <http://www.cs.arizona.edu/classes/cs372/fall03/handouts.html>.
- Some material on these slides has been modified from Thomas W Christopher's Icon Programming Language Handbook, <http://www.tools-of-computing.com/tc/CS/iconprog.pdf>.