

SNOBOL4

CSC 372, Fall 2022
The University of Arizona
William H. Mitchell
whm@cs

A little SNOBOL history

Developed in the Programming Research Studies Department at Bell Telephone Laboratories.

Their interests: Automata theory, graph analysis, associative processors, high-level programming languages.

Were using SCL (Symbolic Communication Language) for symbolic integration, factoring of multivariate polynomials, and analysis of Markov chains.

First called SCL7, then SEXI (String EXpression Interpreter).

Renamed to SNOBOL, with a *backronym*
StriNg Oriented SymBolic Language

Four versions of SNOBOL from 1963-1966, culminating with SNOBOL4.

Ralph Griswold was involved with all, and was lead on SNOBOL4.

A line-numbering program

Consider a program that reads lines from standard input and writes numbered lines to standard output:

```
% cat lines
```

```
one
```

```
two
```

```
three
```

```
four
```

```
five
```

```
% snobol4 numlines.sno < lines
```

```
1: one
```

```
2: two
```

```
3: three
```

```
4: four
```

```
5: five
```

Try it! `fall22/bin/snobol4` is the executable; examples are in `fall22/snobol4`.

numlines.sno

```
*
* Read lines from standard input and write
* numbered lines to standard output, a bit
* like "cat -n"
*
    n = 1

loop   line = input           :f(end)

       output = n ' : ' line

       n = n + 1             :(loop)

end
```

Two language design decisions

String concatenation:

"The decision not to have an explicit operator for concatenation, but instead just to use blanks to separate the strings to be concatenated, was motivated by a belief in the fundamental role of concatenation in a string-manipulation language. The model for this feature was simply the convention used in natural languages to separate words by blanks."

Control structures:

"We felt that providing a statement with a test-and-goto format would allow division of program logic into simple units and that inexperienced users would find a more technical, highly structured language difficult to learn and use."

Source: *History of Programming Languages (from ACM SIGPLAN History of Programming Languages Conference, June 1-3, 1978)*

Imagine a program that prompts the user for a repetition count and a string to repeat:

```
$ snobol4 repl.sno
```

```
How many of what?
```

```
5 *
```

```
*****
```

```
How many of what?
```

```
7 abc
```

```
abcabcabcabcabcabcabc
```

```
How many of what?
```

```
11 1011
```

```
1011101110111011101110111011101110111011101110111011
```

```

define('repl(s,n)')
  :(main)
*
* function repl(s,n) is like s * n in Ruby
*
repl  eq(n,0)                                :s(return)
      repl = repl s
      n = n - 1                               :(repl)

main
      output = 'How many of what?'
      line = input                            :f(end)
      line span(&digits) . count span(' ') rem . string
      output = repl(string, count)
      output =                                :(main)

end

```

Imagine a program that finds integers in a body of text and then prints them and their sum:

```
$ cat sumints.1
```

```
There were 4 of those, 39 of these,  
along with 22 oddballs,  
and 13 were missing.
```

```
$ snobol4 sumints.sno <sumints.1
```

```
The sum of 4, 39, 22 and 13 is 78
```


sumints.sno, continued

```
*
* Read lines of text from standard input, finding and
* printing integers and their sum.
*

read  line = input                                :f(print)
loop  line break(&digits) span(&digits) . num =   :f(read)
      sum = sum + num
      nums = nums num ' , '                       :(loop)

print output = eq(sum, 0) 'No numbers found'      :s(end)
*
* nums will look like '10, 7, 181, 12, '
* We want: '10, 7, 181 and 12'
*
      nums arb . fnums ' , ' span(&digits) . last ' , ' rpos(0)
      output = 'The sum of ' fnums ' and ' last ' is ' sum
end
```

An expression *recognizer*

In formal language theory, a *language* is a possibly infinite set of strings.

A *recognizer* for a language determines whether a string is in the language.

Consider a recognizer for simple arithmetic expressions:

Expression?

3*(4+5)

OK!

Expression?

((x+y)*z)/2+((x))- (q/3)*500

OK!

Expression?

(x+3)*

Nope!

Note: we'll not handle whitespace.

Here is a first version of a *recognizer* for arithmetic expressions.

```
    expr = '7'

loop output = 'Expression?'
    line = input                :f(end)
    line pos(0) expr rpos(0)   :f(nope)

    output = 'OK!' char(10)    :(loop)
nope output = 'Nope!' char(10) :(loop)

end
```

How can we characterize the arithmetic expressions it recognizes?

expr.sno, continued

Let's go live and extend the expressions that can be handled!

```
$ cat expr.1
75
abc
abc+3
10+abc
a+b+c
a*b+c-d/e
-20
10+a-20--x
(a-1)
-20*(3+4-(x*y))
3*(4+5)
-x*-(-(-x))
((x+y)*z)/2+(x)-(q/3)*500
```

Final result will be in fall22/snobol4/expr.sno

Summary of SNOBOL4 pattern matching

General form of a pattern match statement:

label subject pattern = replacement goto

Pattern creation:

Alternation (|) and concatenation

Value assignment with . and \$

Unevaluated expressions

Control keywords:

&anchor, &fullscan

Predefined patterns:

arb, bal, fail, fence, null, rem, succeed

Functions that produce patterns:

**any(chars), arbno(pattern), break(chars), len(n), notany(chars),
pos(n), rpos(n), rtab(n), span(chars), tab(n)**

SNOBOL4 patterns vs. regular expressions

Number of "words" reported by `wc` for...

- Summary of SNOBOL4 patterns on previous slide: 54
- Summary of Ruby 2.0 REs in "Pickaxe" book: 705
- Microsoft's quick reference for REs: 2,091
- Summary of Icon string scanning in Ruby slides: 81

Recognition capability:

- Regular expressions can recognize strings in a "regular" language—type(3) in the *Chomsky hierarchy*.
- SNOBOL4 patterns can recognize strings in an "unrestricted" language—type(0).

Notable: There are few idioms to learn with SNOBOL4 patterns.

Why did the industry end up using regular expressions when SNOBOL4 patterns are both simpler and more powerful?

SNOBOL4 implementation

SNOBOL4 is implemented in SIL—SNOBOL Implementation Language.

```
*
*      Output Procedure
*
PUTOUT PROC      ,      Output procedure
      POP      (IO1PTR, IO2PTR)  Restore block and val
      VEQLC    IO2PTR, S, , PUTV  Is value STRING?
      VEQLC    IO2PTR, I, , PUTI  Is value INTEGER?
      RCALL    IO2PTR, DTREP, IO2PTR  Get data type repr.
      GETSPC   IOSP, IO2PTR, 0     Get specifier
      BRANCH   PUTVU              Join processing
*
P̄UTV LOCSP      IOSP, IO2PTR      Get specifier
PUTVU STPRNT    IOKEY, IO1PTR, IOSP  Perform print
      AEQLC    IOKEY, 0, , COMP6   Check status
      INCRA    WSTAT, 1           Inc count of writes
      BRANCH   RTN1              Return
*
P̄UTI INTSPC     IOSP, IO2PTR      Convert INT. to STRING
      BRANCH   PUTVU              Rejoin processing
```

In essence, SIL instructions are instructions for a *virtual machine*.

SNOBOL4 implementation: [fall22/snobol4/v311.sil](#) (8,660 non-comment lines)

David R. Hanson developed RATSNO, an adaptation of RATFOR to SNOBOL4.
Here is assignment 8's **seqwords** in RATSNO: (not tested!)

```
&anchor = 1; maxlen = 100000
word = input
for (;;) {
    if (word '.')
        break
    words = words rpad(word,maxlen)
    word = input
}

while (num = input) {
    if (num '.') {
        output = line
        line =
    }
    else {
        words len((num - 1) * maxlen) len(maxlen) . word
        line = line trim(word) ' '
    }
}
output = line
```


A little more history

In 1968 the University of Arizona formed a committee to develop a computer science program. A graduate program with courses from a variety of departments was assembled.

Murray Sargent III, a UA optics professor with an interest in programming languages, had visited Bell Labs and knew of the SNOBOL family.

Sargent embarked on a one-man recruiting campaign to convince Ralph Griswold to leave Bell Labs and join the University of Arizona.

In August of 1971, Ralph Griswold joined the University of Arizona as its first Professor of Computer Science.

On his first day, Dr. Griswold arrived to find a line of students waiting outside his office for advising.

In his office was a desk, but no chair.

SNOBOL4 takeaways

I'd like you to know...

- The syntax for string and pattern concatenation
- How flow of control works
- The basic form of pattern matching, with functions and operators that build patterns, and the "dot" notation for extracting parts of matches
- That there is implicit conversion between integers and strings
- What the variables **input** and **output** represent
- That SNOBOL4 patterns are capable of recognizing an unrestricted language (type 0 in the Chomsky hierarchy)