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


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! spring18/bin/snobol4 is the executable; examples are in spring18/snobol4.
* Read lines from standard input and write numbered lines to standard output, a bit like "cat -n"

\[
\begin{aligned}
    n &= 1 \\

    \text{loop} & \quad \text{line} = \text{input} : f(\text{end}) \\
    \text{output} &= n ': ' \text{line} \\
    n &= n + 1 : (\text{loop}) \\

\end{aligned}
\] 

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
```
10111011101110111011101110111011101110111011
```

CSC 372 Spring 2018, SNOBOL4 Slide 6
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 = :f(main)

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)*
Huh?

Note: we'll not handle whitespace.
Here is a first version of a recognizer for arithmetic expressions.

```
nl = char(10)
expr = 'x'

loop
    line = input                :f(end)
    line pos(0) expr rpos(0)   :f(huh)

    output = 'ok: ' line nl     :(loop)
    huh output = '??: ' line nl  :(loop)

end
```

How can we characterize the arithmetic expressions it recognizes?

How does the above version behave wrt. input and output?
Let's extend the expressions that can be handled!

```sno
$ cat expr.1
x
abc
abc+x
a+b+c
a*b+c-d/e
10
-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
```

```
var = span(&lcase)
int = span(&digits)
op = any('+-/*')
expr = int | var | *expr op *expr
       | '-' *expr | '(' *expr ')

loop
line = input :f(end)
line pos(0) expr rpos(0) :f(huh)
output = 'ok: ' line nl :
(loop)
...
```
Summary of SNOBOL4 pattern matching

General form of a pattern match statement:

\[
label \quad subject \ pattern = replacement \quad goto
\]

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 is the industry using regular expressions when SNOBOL4 patterns are both simpler and more powerful?
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

PUTV 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

PUTI INTSPC IOSP,IO2PTR Convert INT. to STRING
BRANCH PUTVU Rejoin processing

In essence, SIL instructions are instructions for a virtual machine.
SNOBOL4 SIL implementation: spring18/snobol4/v311.sil (12,189 lines)
David R. Hanson developed RATSNO, an adaptation of RATFOR to SNOBOL4. Here is assignment 5's seqwords in RATSNO: (not tested!)

```sno
&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
```
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.
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)