A compiler for the SNOBOL4 Programming Language is currently under development at the Computer Center. The compiler incorporates practically all features of the language as described in (1). The anticipated completion date is the beginning of the fall quarter.

In the following discussion a short summary will be given of the design considerations and the expected performance of the program, hereafter referred to simply as SNOBOL. Suggestions, remarks, or criticism are welcome.

SNOBOL will be programmed for the CDC-6400 machine in COMPASS to run under the SCOPE-3 operating system. It will require a minimum field length of roughly 10000 octal. A typical student job (similar to those in (1)) will be able to run in that space and will use approximately 1 second of CPU time.

The source language, SNOBOL4, is very much like SNOBOL3. The main difference lies in the improved pattern matching rules.

The new idea in the string searching is the introduction of pattern values. Patterns may be manipulated (e.g. concatenated) independent of the actual pattern match itself. This capability, among others, gives provision to read in patterns as data. Several standard variables and standard procedures provide special pattern values, e.g. a pattern which will match any string or any balanced string etc. (c.f. ** or *()* of SNOBOL3).

Using the deferred evaluation feature, patterns can reference themselves recursively. In fact one can describe a syntactic class in Backus notation to produce a pattern which will match any element of that class. For complicated syntax or for long strings, run time may be prohibitively large if this method is used, however. Due to

a new pattern matching algorithm (already tested in ALGOL) SNOBOL will be able to match all syntactically correct patterns.

A limited facility is provided for real (floating point) arithmetic. The four basic operations are defined between real values. Standard procedures convert strings to real values and vice versa.

To obtain arbitrary precision, string arithmetic is handled on a character by character basis. Although integers less than 11 digits will be represented in binary form to gain speed, the user need not be aware of this. Therefore, programs to factor large numbers or calculate series expansions to very high precision can be easily written.

A set of values can be organized into arrays or programmer-defined data structures. SNOBOL supports structures with an arbitrary number of dimensions or fields. The size of an array may be defined at run-time; e.g. it may depend on input data.

Users should be wary of introducing such structures into procedures, as doing so will cause a slight overhead.

The compilation during execution feature will be implemented. It allows calling SNOBOL from a running program to compile a string. The created code is stored as the value of a variable, and may be jumped to later. The string will be compiled as an appendix to the set of code already created and thus may reference all variables, labels, data structures or procedures which have been introduced in this code.

The standard procedures READ and PRINT will be used to associate a variable (even if it is an element of a data structure) with a SCOPE file. Instead of a data set reference number, the name of the file will be used to designate the file.

The FORTRAN IV format in the PRINT procedure will be omitted because it is an empty gimmick anyway. The allowed X, A or H format specifications can be expressed in SNOBOL4 very efficiently. A carriage control parameter may be introduced instead. Another possible restriction will be the omission of the 'truth predicates' partly because they use unique characters without adding even a tiny bit to the power of the language. For similar reasons the exotic 'keywords' will be degraded to simple variables.

About the internal organization of SNOBOL: the compiler will analyze its input text using a finite-state machine algorithm, and will produce object code in the form of 60 bit micro instructions. Expressions will be translated into Reverse Polish from. The compilation process will consist of a forward scan of the source text, generating the final code and the distribution of label and literal references. If the source program does not refer to it, the compiler will be purged after compilation.

At run time a static area, a stack, and a dynamic area will be maintained. The static area will contain a descriptor for every variable, procedure, data structure, array, or field. It will also contain the compiled micro instructions. The stack will be used to store information displaced by local entities when calling procedures. Operands for a binary operator will be brought to the top of the stack. The stack will be used extensively during pattern matching since the algorithm makes use of a single recursive procedure.

All strings will be stored in the dynamic area, 7 characters and a pointer per word. The pointers will link the words to form a string. In this way the need for garbage collection is almost completely eliminated. Nevertheless a simple garbage collection scheme will be used to compress the dynamci area occasionally (before a compilation, for example).

Charles Simonyi

References:

(1) Griswold, Poage and Polonsky
Preliminary report on the SNOBOL4 Programming Language
Bell Telephone Lab.

```
<program>  :: = END | <rule><semicolon><program>

<rule>  :: = <label part><string part><go to part>

<label part>  :: = <empty>|<identifier>

<string part>  :: = <empty>|<string reference>
                     <pattern reference><right part>

<string reference>  :: = <primary>
<pattern reference> :: = <empty>|<pattern expression>
<right part>  :: = <empty>| = | = <pattern expression>

<go to part>  :: = <empty>| :  <designational expression>|

<designational expression>  :: = <condition>(<identifier>)|
                                 <condition>($<primary>)|
                                 <condition>[<variable value>]

<condition>  :: = <empty>|S|F

<identifier>  :: = <letter>|<identifier><letter>
                   <identifier><digit<|<identifier>.

<simple variable>  :: = <identifier>

<variable value>  :: = <simple variable>|<name function>|
                        <field designator>|<array element>|
                        $<primary>

<primary>  :: = <variable value>|.<variable value>|
                 <literal>|<function designator>|<real number>|
                 (<string expression>)

<literal>  :: = <string>|<unsigned integer>

<real number>  :: = <unsigned integer>.|<unsigned integer>
                                       .<unsigned integer>

<factor>  :: = <primary>|<factor>**<primary>
<term>  :: = <factor>|<term><multiplying operator><factor>
<arithmetic expression>  :: = <term>|<adding operator><term>|
                              <arithmetic expression><adding operator><term>

<adding operator>  :: = + | -
<multiplying operator>  :: = *|/

<string expression>  :: = <arithmetic expression>|
                          <string expression><arithmetic expression>
```

```
<pattern primary>  :: = <primary>|*<variable value>|
                        (<pattern expression>)|
                        <pattern primary>_<naming operator>_
                                          <variable value>

<naming operator>  :: = .|$

<pattern term>  :: = <pattern primary>|<string expression>|
                     <pattern term>_<pattern primary>|
                     <pattern term>_<string expression>

SIMPLE
<pattern expression>  :: = <pattern term>|
                           <pattern expression>V<pattern term>
  <pattern expression> ::= <empty> | <simple p. e.>
<name function>  :: = <function designator>
<data declaration>  :: = <function designator>
<field designator>  :: = <function designator>

<function designator>  :: = <identifier>(<actual parameter list>)
<actual parameter list>  :: = <empty>|<pattern expression>|
                              <actual parameter list>,<pattern expression>

<array element>  :: = <simple variable>[<subscript list>]
<subscript list>  :: = <string expression>|<subscript list>,<string ex
                                                        expression>
```

(where _ denotes a blank space, i.e. one or more blanks)