Version 3 of SNOBOL4

Installation Information for the IBM 360
Operating under OS

Note: This document is a revision of a document originally prepared in July 1971. No substantive changes have been made since that date. The information here is obviously somewhat dated, but it should be functional.

1. Installation Instructions

1.1. Installation Requirements

SNOBOL4 normally operates best in a partition of about 200K bytes. It will run in a minimum partition of about 165K bytes, depending on buffer requirements, nonresident system modules, external functions, and so forth. Such a small partition will not give satisfactory service, however.

The SNOBOL4 load module for Version 3 requires 21 tracks on a 2314 disk.

1.2. Installation of Version 3.0

Version 3.0 is distributed on a 9–track, 1600 bpi tape with standard label **S4V3**. The first file(**LKED**) contains input to the link editor. The second file (**TEST**) contains three sample programs that may be used to test Version 3.0. Both files have **RECFM=FB**, **LRECL=80**, **BLKSIZE=3200**. **LKED** contains 2458 logical records, while **TEST** contains 119.

1.2.1. Load Module Formation

Typical JCL to form the SNOBOL4 load module using a link edit cataloged procedure is:

```
// EXEC LKED
//LKED.SYSLMOD DD DSN=SNOBOL,UNIT=DISK,DISP=(NEW,KEEP),
// VOL=SER=XXXXXX,SPACE=(TRK,(20,5,1))
//LKED.SYSIN DD DSN=LKED,DISP=OLD,UNIT=9TRACK,VOL=SER=S4V3
```

The data set and volume names for **SYSLMOD** vary from installation to installation. The load module is typically placed on **SYS1.LINKLIB** (rather than SNOBOL) to avoid JOBLIB cards. Other JCL details may have to be changed to conform to installation conventions. The result of the link edit is an executable load module with member name SNOBOL4.

1.2.2. Punching Test Programs

The test programs on file 2 can be punched by JCL similar to the following:

```
// EXEC PGM=IEEPTPCH
//SYSPRINT DD SYSOUT=A
//SYSUT2 DD UNIT=PUNCH
//SYSUT1 DD DSN=TEST,UNIT=9TRACK,VOL=SER=S4V3,
// DISP=OLD,LABEL=(2,SL)
```

The three test programs should be separated at the ‘‘end–of–file’’ cards that divide them, and then run individually.

The first file may be punched by similar JCL, although that normally is not necessary.

1.2.3. Cataloged Procedure

A cataloged procedure for SNOBOL4 may be created by JCL similar to the following:

```
// EXEC PGM=IEBUPDAT,PARM=MOD
//SYSUT1 DD DSN=SYS1.PROCLIB,DISP=OLD
//SYSUT2 DD DSN=SYS1.PROCLIB,DISP=OLD
//SYSPRINT DD SYSOUT=A
//SYSIN DD DATA
./     ADD   SNOBOL4,01,0,1
// EXEC PGM=SNOBOL4
//FT05F001 DD DDNAME=SYSIN
//FT06F001 DD SYSOUT=A
//FT07F001 DD UNIT=PUNCH
./       ENDUP
/*
```

If a public library of external functions is available, a reference to it with the ddname SNOLIB should be included in the cataloged procedure.

1.2.4. Executing SNOBOL4

With the cataloged procedure given above, SNOBOL4 can be run with the following deck setup:

```
//JOBLIB DD DSN=SNOBOL4,VOL=SER=XXXXXX,DISP=OLD,UNIT=DISK
// EXEC SNOBOL4
//SYSIN DD *
...
 SNOBOL4 Source Deck
...
        data
/*
```

If SNOBOL4 is link edited into **SYS1.LINKLIB**, a JOBLIB is not necessary.

2. Updating to Version 3.5

Five updates presently apply to Version 3.0, yielding Version 3.5. Version 3.5 is obtained from Version 3.0 by applying SUPERZAP corrections to the 3.0 load module. SUPERZAP is an IBM type 3 program (PID 360D–03.0.003).

Typical JCL to convert to Version 3.5 is

```
// EXEC PGM=SUPERZAP
//SYSLIB DD DSN=SNOBOL4,VOL=SER=XXXXXX,DISP=OLD,UNIT=DISK
//SYSPRINT DD SYSOUT=A
//SYSIN DD *
...
  SUPERZAP control cards
...
/*
```

A unlabelled tape containing the SUPERZAP control cards for the five updates is included separately from the distribution tape.

In addition to correcting errors in Versions 3.0, 3.1, 3.2, 3.3, and 3.4, these control cards change the program titling information to identify Version 3.5.

3. Information for Programmers

3.1. Language Description

The basic reference is the Second Edition of the Prentice–Hall book, *The SNOBOL4 Programming Language* (1971), which describes Version 3.

3.2. Use of the PARM Field

The **PARM** field of the SNOBOL4 **EXEC** card can be used to modify the default allocations of space within the SNOBOL4 partition. Three parameters may be specified:

L the low bound on dynamic storage

H the high bound on dynamic storage

R the amount of storage reserved for other uses

All values are specified in bytes. Default values are **L=35200**, **H=200000**, **R=10000**. A typical **EXEC** card, specifying nonstandard parameters, is

```
// EXEC SNOBOL4,PARM='R=15000,L=64000,H=150000'
```

The parameters may be given in any order and any parameter may be omitted, in which case the corresponding default is used. The PARM field is processed from left to right. A syntactic error terminates processing of the **PARM** field at the point of error and without comment.

After **L**, **H**, and **R** are determined, a variable–length **GETMAIN** is executed to get an amount of storage between **R+L** and **R+H** bytes. A **FREEMAN** is then executed to release bytes. If **R+L** bytes are not available, the run is terminated with system completion code 804.

The following guidelines should be observed in overriding the default parameters:

1. Programs creating large numbers of different strings may run more slowly with a larger value of **H**.

2. **L** should not be set below the default value except in special circumstances.

3. **R** reserves space for I/O buffers, nonresident access methods, external functions and so forth. **R** may have to be set above the default value if many highly–blocked data sets are open simultaneously.

4. **R** ordinarily should not be set below the default value. If **R** is too small, termination during execution may occur with a completion code such as 804 and 80A.

3.3. Version 3 Errors Corrected in Version 3.5

3.0.1: In conversion to **CODE**, a statement that is erroneous because of improper termination does not cause the conversion to fail. If the erroneous statement is subsequently executed, Program Error 28 occurs.

3.0.2: If Program Error 9 is treated as nonfatal because **&ERRLIMIT** is greater than zero, a program interrupt results.

3.0.3: The **FIELD** function does not handle its second argument properly. **FIELD(DT,N)** returns the (**N**–1)st field of **DT** as value. If **N** is 1, the data type name is returned. **FIELD** fails as it should if **N** is greater than the number of fields defined for **DT**.

3.0.4: The **CONVERT** function does not handle the conversion of a string to **STRING** properly. The result is always the formal identification **STRING** rather than the string itself.

3.0.5: The **DATE** function is not properly linked to the procedure that produces the date. Instead a meaningless result is returned, which is interpreted by the SNOBOL4 system as an external data type.

3.1.1: A program interrupt occurs if storage regeneration is required during conversion from **TABLE** to **ARRAY**.

3.1.2: The **DATA** function erroneously permits a prototype with no fields (e.g. **DATA("MODE()")**). Corresponding data objects can be created, but cause a program interrupt during storage regeneration.

3.1.3: Program malfunction may occur if a nonfatal program error occurs during execution of a programmer–defined trace procedure which is invoked to trace another nonfatal error.

3.1.4: **EVAL(null)** returns the integer 0, not the null string.

3.1.5: **CODE(null)** and **CONVERT(null,"CODE")** generate Error 4. They should return the null statement.

3.1.6: **EVAL(" ")** returns an object of data type **CODE**, rather than the null string.

3.2.1: If failure occurs during evaluation of an unevaluated expression that is the argument of a pattern–valued function, a program interrupt (**CUT BY SYSTEM**) occurs in some circumstances.

3.2.2: If **CUT BY SYSTEM** occurs during SNOBOL4 termination because of a FORTRAN I/O error, a loop occurs.

3.2.3: An error in the implementation design of tables causes the following problems.

1. Assignment to a table reference is not made if the object of that assignment involves a computation that causes extension of the referenced table.

2. If the unary name operator (**.**) is applied to a table reference, the resulting name is disconnected if the table is extended. Furthermore, the space occupied by the table before its extension is not collected during storage regeneration.

3.3.1: Failure of a programmer–defined trace procedure because of return by **FRETURN** produces Error 17 (**ERROR IN SNOBOL4 SYSTEM**), rather than being ignored as it should be.

3.3.2: An error in the storage allocation procedure used in the concatenation of strings may cause a subsequent program interrupt (**CUT BY SYSTEM**) during storage regeneration.

3.3.3: If **APPLY** is used to invoke a programmer–defined function or data object creation function and no argument is supplied, Error 17 occurs because such functions are not prepared to cope with the lack of an argument. For example

```
APPLY("F")
```

produces Error 17 if, e.g., **F** is a programmer–defined function. However

```
APPLY("F",)
```

performs properly since one (null) argument is provided. Error 25 should be given in the former case.

3.4.1: Some termination messages do not allow enough space for the printing of statement numbers that are more than three digits long.

3.4.2: In **DUPL(S,N)** if the size of **S** multipled by **N** is very large, undetected overflow can cause various malfunctions.

3.4.3: If the argument to **EVAL** is a string that ends (erroneously) with a unary operator, Error 17 occurs rather than **EVAL** failing as it, should. Similar errors occur if a statement in the source language or in conversion to **CODE** ends with a unary operator.

3.4.4: If the argument of the unary **@** operator or the second argument of the binary **$** operator is an unevaluated expression, and if the result of evaluating that expression during pattern matching is not a variable, program malfunction results.

3.4. Errors not Corrected in Version 3.5

The following errors are known to exist in Version 3.5.

2.0.2: The number of arguments with which an external function can be called is limited to 20.

2.0.3: If an external function is unloaded using the **UNLOAD** function, the external function is undefined, but any function **OPSYN**ed to it is not. Subsequent use of a function **OPSYN**ed to an unloaded external function may cause a program interrupt.

3.5.1: The array function leaves a negative–length string that occasionally causes malfunction during a subsequent storage regeneration.

3.5.2: In conversion from **TABLE** to **ARRAY**, if a storage regeneration is required to get space for the array, the resulting prototype is erroneous and may cause malfunction during a subsequent storage regeneration.

3.5.3: Use of **OPSYN** for operators erroneously advances a pointer in the SNOBOL4 data region. Repeated use of operator **OPSYN**ing eventually causes data to be overwritten with resulting program malfunction.

3.5.4: If the argument of **ANY**, **BREAK**, **NOTANY**, or **SPAN** is null, this error is not detected until pattern matching is performed.

3.5.5: If failure occurs during evaluation of an unevaluated expression that was the argument of **ANY**, **BREAK**, **NOTANY**, or **SPAN**, a program interrupt (**CUT BY SYSTEM**) occurs. This is the same type of problem that is mentioned in Error 3.2.1.

3.5.6: If an illegal data type results from evaluating an expression during pattern matching that was the argument of any pattern–valued function except **ARBNO**, **CUT BY SYSTEM** results.

3.5.7: An error in the quickscan heuristics causes erroneous failure to match in some cases when **BREAK** occurs in an unevaluated expression.

3.5.8: If the argument of the binary **$** operator is an unevaluated expression and it fails when evaluated during pattern matching, **CUT BY SYSTEM** results during subsequent processing.

3.6.1: If the argument to **POS**, **RPOS**, **TAB**, **RTAB**, or **LEN** is a negative integer, this error is not noted when the pattern is formed, but only when pattern matching is performed.

3.6.2: If the second argument of **OPSYN** has no definition, **OPSYN** performs no operation. In this case **OPSYN** should make the first argument an undefined function.

The errors listed above will not be corrected by patches to the distributed object modules. Users or Version 3.5 should be aware of these errors and attempt to find ways to program around them.