A Manual For

S C A L P

A

Self-Contained ALGOL Processor

for the

LGP-30

Computation Center
Dartmouth College
Hanover, New Hampshire
1 October, 1962

S C A L P

1. Introduction

The ALGOL language is rapidly becoming a standard language for expressing numerical computational problems. Recently a large subset of ALGOL, called ALGOL-30, was implemented for the LGP-30 by a student group in the Mathematics Department at Dartmouth College. They were remarkably successful in providing for almost all of the features of ALGOL, omitting only those for which a re-allocation of storage at run time is required. (These omissions include recursive blocks and procedures, dynamic arrays, and arrays called by value.) However, the system is very large and requires two-and-a-half manual passes to complete a problem. That is, the user must load three system tapes and two program tapes before he gets his answers. Since this manual activity takes great amounts of time, it is obviously desirable to have a system that requires fewer passes, only one if possible. Such a system is SCALP.

Clearly, if nearly full ALGOL requires a system that fills two LGP-30 memories, any system that can be self-contained in the memory and still leave room for program and data will necessarily enforce even greater restrictions on ALGOL than does ALGOL-30. These restrictions, listed in section 2, are considerable, and yet every attempt was made to be judicious about omitting features of ALGOL. Compromises between expected usefulness and space requirements in the system were made. Not everyone will agree that all the restrictions made were wise, but it is hoped that the SCALP system will be useful to a large percentage of ALGOL programs with perhaps small changes here and there.

The discussions of ALGOL in this manual assume a knowledge of that language [see An ALGOL Primer, CCM-3, Dartmouth Computation Center, or any of several other expositions of ALGOL.] The user who is familiar with ALGOL-30 should be careful of several important differences between the language of SCALP and that of ALGOL-30. These differences can be discovered by a careful reading of this manual.

SCALP was prepared by Stephen J. Garland and Anthony W. Knapp. This manual was written by Thomas E. Kurtz.

2. Restrictions

The SCALP system places the following restrictions on ALGOL:

2.1 SCALP permits no blocks or procedures. The program itself, however, is a block and all variable used must be declared. Since there are no blocks or procedures, the concepts of recursive procedures, own arrays, and dynamic arrays do not arise. All arrays in SCALP must be declared using numerical integer constants to indicate the subscript bounds. The word own is not a part of SCALP, and may be used as an identifier, as may all other ALGOL words omitted from SCALP. (A list of official SCALP words appears in section 3.3.)

While procedures declarations are not allowed, provision has been made for the standard functions, a list of which follows:

sin	sqrt	read
cos	abs	print
arctan	sign	title
ln	entier	
exp	random	

The definitions of most of these functions are explained in the ALGOL-60 Report [ACM Communications, May 1960] and in the various primers, and will not be included here. random is a no-argument function which when called generates a floating point pseudo random number in the interval [0, 1). To obtain a pseudo random digit from the set (0, 1, ..., 9), use entier (10* random). Read, print, and title are input-output functions explained in sections 3.6, 3.7, and 3.8.

2.2 Conditional expressions are not permitted. However, conditional statements may be used in full generality. Thus

x:= y + if a < b then 3 else z; is not allowed, and

might be replaced by

if a < b then w:= 3 else w:= z; x:= y + w;

Along the same line, the arguments of a switch declaration must be simple labels. Thus, an argument of a switch declaration may not be another switch name or a conditional designational expression. If it is desired to employ nested switches, the second

switch must have a labelled call, the label appearing as an argument of the first switch declaration. Thus in place of

```
switch alpha:=goof, beta[n], stop;
switch beta:=loop, exit;
```

we might use something like

```
switch alpha:=goof, L1, stop;
switch beta:=loop, exit;
L1: goto beta[n];
```

2.3 Only a limited type of for statement is allowed in SCALP. Both the step-until and the general list are omitted. The only type permitted is the while element, which must be of the form

```
for <variable>:= <expression> while <relation> do S;
```

where S is the iterated statement. Admittedly, the step-until element is probably the most useful. It may be imitated as follows: instead of

```
for i:= 1 step 1 until n do S;
```

use

```
i:=0;
for i:= i + 1 while i ≤ n do S;
```

Another form avoids the use of for altogether:

```
      i:= 1;
loop: S;
      i := i + 1;
      if i ≤ n then goto loop;
```

This last form though a bit inconvenient to use, is actually executed more rapidly than the step-until would be, though at about the same speed as the while type.

2.4 The special integer divide ÷ is not included in SCALP. It may be synthesized as follows: for

```
a ÷ b
```

use

```
sign (a*b)*entier (abs (a/b))
```

2.5 Boolean variables and operators are not allowed. One of the most common constructions which call for their use is

```
if x ≤ 2 ∧ x ≥ 1 then goto A else goto B;
```

to see if x lies in the closed interval [1,2]. Since the ∧ (or its transliterated form and) is not permitted in SCALP, one might use instead:

```
    if x ≤ 2 then goto L1 else goto B;
L1: if x ≥ 1 then goto A else goto B;
```

3. Preparing SCALP Program Tapes

3.1 Symbols

Besides the restrictions on ALGOL given earlier, certain ALGOL symbols are absent from the flexowriter keyboard and must be represented by other symbols.

ALGOL	SCALP
×	*
↑	Δ
<	lt
≤	lte
=	==
≥	gte
>	gt
≠	=/ or /=
'	''
10	ten
:	::
1	l

In addition, the following symbols are excluded from SCALP:

÷	≡	⊃	∨	∧	¬	‹	›
step	until	own	label	value			
Boolean	procedure	string					

All other symbols in ALGOL appear the same in SCALP.

3.2 Upper and Lower Case

Upper and lower case shifts should be used to make the typed copy more readable. They have no effect on the program to be run.

3.3 Restricted Words

No SCALP word may be used as an identifier (variable name or statement label). These include: go to, goto, if, then, else, for, do, while, comment, begin, end, integer, real, array, switch, sin, cos, arctan, sqrt, ln, exp, abs, sign, entier, random, read, print, title. Otherwise, full freedom for variable names is permitted except that only the last five characters are used. Thus squareroot and cuberoot will look the same to SCALP , since the last five letters are the same in each case. Furthermore, words like witch that conflict in their last five letters with official SCALP words, in this case switch, may not be used as identifiers.

3.4 Symbol Separators

All SCALP symbols, labels, and variable names are separated by stop codes (the key labelled STOP CODE on the flexowriter). Dyads such as := are considered single symbols, as are all SCALP words like begin, go to, and lte. Stop codes must not be omitted or misused, and great care in preparing tapes must be exercised.

3.5 Number Representation in SCALP

Numbers may be included in SCALP programs. Any legal ALGOL number is permitted except that the following conventions must be followed:

1. The number must be started and finished with an extra stop code, so that a double stop code separates numbers from adjacent symbols.

2. There must be a stop code at least every five characters in the number, but complete freedom as to how much more often they might be used is allowed.

3. The letter string ten is used in place of the ALGOL subscript $_{10}$.

4. The significant figure part must not exceed eight digits, excluding sign and decimal point. (Since only about 7.5 significant digits are available for computation, this restriction causes no loss of accuracy.

5. The digit 1 is represented by an l, a lower case L.

The following examples are correct in SCALP:

```
x'':='a'+''2'';'
mat'['-1',,'i'+''+3'']'
c'*''14.29''+'d'
xyz'/''+.357't'en-'5''
```

3.6 Input

Numbers and data may be entered when the problem is being run by using the read function. The following format is used in your program:

```
read'('var1',,'var2',,'var n')';'
```

There may be any number of arguments, and each may be a variable name representing a simple or subscripted variable of type either real or integer. For example,

```
read'('x',,'i',,'matrix'['i'+''1'']',,'bla')';'
read'('n')';'
```

At run time the read function will cause the system to call for data input through whichever input device is connected (by means of the toggle on the photoreader). Small amounts of data may be entered via the flexowriter, but large amounts should be entered through the photoreader. In either case a data tape should have been previously prepared.

On the data tape numbers are typed exactly as outlined in section 3.5 except the leading stop code is not needed. Thus, a data tape for reading the numbers

$$2,\ -10256,\ 2.79_{10}-23,\ 3.14159$$

would be

```
2''-10'256''2.79'ten'-23''3.141'59''
```

Notice that the double stop code is used to finish a number, just as with numbers in your SCALP program, but there are no stop codes at the beginning of each number.

The number on the data tape must agree in type with the corresponding argument of the read function. Thus, if the program is

```
begin'integer'n';'
read'('n')';'
```

a correct data tape would be

+17'' ,

whereas +17.0'' would be incorrect.

3.7 Output

Numbers are printed in SCALP by means of the print function. As with read, there may be any number of arguments and each one may be of any type. Arithmetic expressions may also appear as arguments of the print function (which is not true of the read function). The following is a correct use of the print function:

print'('x',,'n'+''2'',,''17'')';'

Expressions of type real are printed in floating point in the form

±.xxxxxxx ±yy ,

where ±.xxxxxxx stands for the seven digit fraction and ±yy stands for the power of ten. Integer expressions are printed in the form

±nnnnn

where nnnnn is an integer that may have leading spaces. In all cases, the plus signs are omitted, so that the constant pi would come out

.3141592 01

and -17 would come out as

- 17 .

After each number is printed SCALP executes a tab on the flexowriter.

3.8 Title

Alphabetic information for labelling output data and for other purposes may be printed at run time by the title function. This function treats its argument as a letter string which it prints out almost exactly as given. There are two exceptions:

1. There must be stop codes at least every five characters, except as indicated in 2.
2. Typewriter controls are indicated by special code words six characters long which must be preceded and

followed by stop codes. These codes are:

carriage return	ccarr.'
upper case	c u.c.'
lower case	c l.c.'
tab	c tab.'
backspace	c b.s.'
color shift	c c.s.'
stop code (')	c stop'

The entire message is enclosed in parentheses. The following shows a correct use of the title function:

```
title'('c u.c.'t'c l.c.'his i's a g'ood p'rogra'm.')';'
```

The printed result will be

This is a good program.

The two typewriter controls ccarr. and c tab. may also be inserted as arguments in read and print statements. They must, of course, be set off by commas. The following examples are correct:

```
read'('ccarr.',,'x')';'
print'('ccarr.',,'x',,'y',,'c tab.',,'z')';'
```

Attempting to insert any of the other typewriter controls into read or print statements will cause an illegal variable error stop.

4. An Example

As an example of a correctly typed SCALP program the quadratic equation solver procedure appearing on page 26 of the ALGOL Primer is presented as a SCALP program with read and print functions replacing the procedure heading, and with titles to improve the output.

```
begin'comment'This routine solves quadratics. It calls
for three floating point numbers to be read in and prints
out the two roots, if real. If no real roots, it so
indicates.';'
real'a',,'b',,'c',,root1',,'root2',,'z',,'disc';'
start'::'read'('ccarr.',,'a',,'b',,'c')';'
disc':='b'*'b'-''4''*'a'*'c';'
```

```
if'disc'lt''0''then'goto'none';'
z':='sqrt'('disc')'/'(''2''*'a')';'
root1':='-'b'/'(''2''*'a')'-'z';'
root2':='root1'+''2''*'z';'
print'('ccarr.',,'root1',,'root2')';'
goto'start';'
none'::'title'('ccarr.'c u.c.'n'c l.c.'o rea'l roo'ts')';'
goto'start'end'quadratic'
```

As another example, let us consider Algorithm 112, published in the ACM Communications, vol. 5, August 1962, p. 434. The original form of the algorithm, corrected, stripped of its explanatory comment and procedure heading, and with the formal parameters included in ordinary declarations, follows:

```
begin integer i, n; real x0, y0; array x, y[1:100];
Boolean b, answer;
x[n+1]:=x[1]; y[n+1]:=y[1]; b:= true;
for i:= 1 step 1 until n do
   if (y0 < y [i] ≡ y0 > y[i+1]) ∧
     x0-x[i]-(y0-y[i])X(x[i+1]-x[i])/(y[i+1]-y[i]) < 0
   then b:= ¬ b;
answer:= ¬ b;
go to exit end
```

Notice that this routine contains (1) Boolean variables, (2) Boolean operators, and (3) a step-until element. We must eliminate all these features to be able to use SCALP, which we do as follows:

(1) Define b and answer to be to type integer and let +1 be true and -1 be false.

(2) Rewrite the long conditional statement into a series of conditional statements to eliminate the Boolean operators.

(3) Replace the step-until element by a while element.

All these changes are made in the modified algorithm below:

```
begin integer i, n, b, answer; real x0, y0; array
      x, y [1:100];
x[n+1] := x[1]; y [n+1] := y[1]; b := 1; i := 0;
```

```
for i := i + 1 while i ≤ n do
begin if y0 < y[i] then goto L1;
      if y0 > y[i+1] then goto no;
L2:if x0-x[i]-(y0-y[i])X(x[i+1]-x[i])/(y[i+1]-y[i]) <0
   then goto okay else goto no;
L1: if y0 > y[i+1] then goto L2 else goto no;
okay: b := -b;
no: end;
answer := -b;
goto exit end
```

Finally, we replace X by *, < by lt, ≤ by lte, > by gt, : by ::, 1 by l, and insert stop codes to get a legal SCALP program.

```
begin'integer'i',,'n',,'b',,'answer';'real'x0',,'y0';'
   array'x',,'y'['''l''':'''100''']';'
x'['n'+'''l''']':='x'['''l''']';'
y'[n'+'''l''']':='y'['''l''']';'
b':='''l''';'
i':='''0''';'
for'i':='i'+'''l'''while'i'lte'n'do'
begin'if'yo'lt'y'['i']'then'go to'L1';'
   if'yo'gt'y'['i'+'''l''']'then'go to'no';'
   L2'::'if'xo'-x'['i']'-'('yo'-'y'['i']')'
            *'('x'['i'+'''l''']'-'x'['i']')'
            /'('y'['i'+'''l''']'-'y'['i']')'lt'''0'''
      then'go to'okay'else'go to'no';'
   L1'::'if'y0'gt'y'['i'+'''l''']'
      then'goto'L2'else'go to'no';'
   okay'::'b':='-'b';'
   no'::'end';'
answer':='-'b';'
go to'exit'end'
```

The above example, though perhaps a bit hard to plow through, illustrates how to adjust ALGOL programs for features of ALGOL not included in SCALP. This example is a "bad" one since it includes most features that SCALP excludes. You will find that for most common problems the inconvenience will not be nearly as great.

5. Preparing Source Tapes in SCALP

5.1 Cutting a Tape

To prepare a source tape for a SCALP program, use an off-line flexowriter and follow these instructions:

a. Lift all dash buttons on the flex.
b. Make sure there is both paper and tape.
c. Turn on the POWER switch.
d. Depress PUNCH ON.
e. Depress TAPE FEED until about two feet of blank tape have been produced.
f. Touch CAR RET.
g. Type your program according to the directions in section 3. You may use carriage returns, tabs, and case shifts to improve the appearance of the printed copy. However, a tab (or tabs) should be used only at the start of a new line. Avoid all use of spaces (except in go to). Avoid use of the backspace except possibly to overtype = and/ to form ≠. But never use the backspace to correct errors.
h. When finished, touch CAR RET and depress TAPE FEED until about a foot of trailing tape is produced.
i. Tear off the tape, label it clearly at the front end, and, if no one is following you, turn off POWER.

5.2 Correcting a Mistake

If you make a typing error, you may correct it as follows:

a. Using the rearward of the two knurled knobs, turn it backwards the exact number of clicks to include all the incorrect characters. Usually, one click will be sufficient, but if not, be sure to count all characters including upper or lower case shifts, carriage returns, stop codes, etc.
b. Push DELETE the same number of times.
c. Type the correct characters and continue in sequence.
d. Do NOT use backspace to correct tapes--it just won't work.

5.3 Listing a Tape

To list a tape to see, e.g., if it is correct:

a. Turn on the flex and lift all dash buttons. Check paper supply.
b. Place the tape in the reader located at the left front of the flex. Be sure the drive sprocket fits into the sprocket holes.
c. Depress COND STOP.
d. Depress START READ.

5.4 Duplicating a Tape

To duplicate a tape:

a. Turn on the flex, lift all dash buttons, check paper and tape supply.
b. Depress PUNCH ON.
c. Depress TAPE FEED until you have two feet of blank tape.
d. Go to step b in section 5.3 above (listing a tape).

5.5 Correcting While Duplicating

a. Proof read carefully the typed copy of your tape, noting all errors and the necessary corrections.
b. Set up for duplicating a tape as in section 5.4. Duplicate.
c. As you near the first error, lift COND STOP, and press START READ enough times to duplicate the last correct word or symbol and the following stop code.
d. Lift PUNCH ON.
e. Press START READ just enough times to bypass the incorrect words or symbols. They will be printed but not punched.
f. Depress PUNCH ON.
g. Type the exact correct sequence corresponding to the bypassed incorrect one, including all stop codes, and carriage returns, if any.
h. Depress COND STOP and START READ to continue

duplicating until you approach the next error; then go back to stop c above.

i. List the corrected tape and proof read the listing. (Do not proof read from the listing produced while duplicating and correcting; there may be errors on the tape that don't show on the paper because of punching errors.

6. Running a Program

6.1 General Procedure

We now assume that your program and data tapes have been correctly prepared. The following steps are all that is necessary to compile and run your problem.

a. Make sure the SCALP system is in memory, and that the computer is ready to go. If not, consult a Computation Center assistant who will prepare the computer.

b. Load the program tape upside down in the photoreader. (Again, a CC assistant can help if necessary.) Turn the toggle to "typewriter".

c. Turn on the 6 BIT input button and perform OCNS (ONE OPERATION, CLEAR COUNTER, NORMAL, START).

d. Type "compile" and press START.

e. Turn toggle to "READER" and press START, (The program is now being compiled.)

f. At the end of compilation the system will type out either "done" or "load". If "done", press START, and continue to next step (g). If "load", the system is calling for the library tape containing all the special functions except abs, sign, read, print, and title. Load the library tape in the photoreader, make sure the input toggle is set to READER, release 6 BIT, and press START. When the library tape has been loaded, the system will type "done". Then you should depress 6 BIT, turn the input toggle to TYPEWRITER, press START, and continue to the next step.

g. Type "start", press START, depress BREAKPOINT 8

(to get continuous running).

h. Load the data tape, if any, in the reader; press START. (Your problem is now running.)

6.2 Errors at Compile Time

There are a number of grammatical and similar errors that are detected at compile time. They may be caused by typographical errors, misplaced stop codes, too many left or right parentheses, and so forth. A list of possible error indications together with their meaning and suggested remedy is given below. In all cases, reprogramming is necessary.

error indication	meaning	remedy
ovflo const	Too many constants (maximum 30)	Treating some of the constants as input data.
ovflo expr	Expression too complicated	Break long expression into parts; look for deep nesting of parentheses or functions.
ovflo store	Storage exceeded	Cut out non-essential parts of program, or reprogram for ALGOL-30.
ovflo symb	Too many symbols (maximus 41 to 53)	Eliminate unneeded labels or replace a set of variables by a subscripted variable.
ovflo temp	Temporary storage exceeded	Replace long statement by two or more shorter ones.
array xxxxx	Array not declared or not subscripted	Declare arrays; make sure all subscripted variables are followed by '[.

array xxxxx stands for the offending identifier, which is printed.

array bound	Upper bound < lower bound	Check array bounds.
declr xxxxx	Illegal symbol in declaration	Check declarations for illegal appearance of xxxxx.

label xxxxx	Label defined twice; variable used as label; goto if.	Making sure each label is defined exactly once.
var. xxxxx ???	Illegal variable; may indicate missing variable.	Look out for illegal expressions or adjacent operators.
var. adj.	Two adjacent variables	Look especially for a missing multiply operator *.
xxxxx xxxxx	Syntactical error; second symbol is one just read in.	Look for a statement containing the offending pair of symbols and check its syntax. Only lower case is printed; this (will show as a ␣␣␣␣9, where ␣ stands for a space. ␣␣␣␣ 9 ␣␣␣␣; suggests an incorrect parentheses count, as also does read xxxxx.
digit	Significant figure-part $\geq 10^8$ or exponent ≥ 100	Round off offending constant or scale as necessary.
input	overflow (real number $> \sim 10^{38}$)	Eliminate real constants that are too large.

6.3 Run Time Errors

Even if a program is grammatically correct, certain errors can occur at run time. Such things as trying to take the square root of a negative number and generating a number too large for the computer are typical. A list of the run time error indications together with their meaning and suggested remedy is given below.

In all cases, the remedy is to reprogram, or to use the execute or patch option and continue.

error indication	meaning	description and/or remedy
div 0	Division by zero has been attempted	Error may occur in a divide operation, or in a power operation of the form realΔinteger where real = 0 and integer < 0.
ovflo	Real number too large (must be $<\sim 10^{38}$ in absolute value).	Occurs only on a temporary or assigned hold. An underflow (real number < 10^{-38} in absolute value) will replace the number by zero, but will not give an error stop.
swtch	Argument of switch not in range	Argument must be from 1 to n, inclusive, where n is the number of positions in the switch declaration.
power	Error in aΔ expression	Look for 0Δ0 where zeros may be of any type, 0Δi where i is a negative integer and the zero is of type integer, aΔb where a < 0, or a = 0 and b < 0 and where a and b are real.
ptype	Mixed type in iΔj	Occurs if i and j are of type integer, but j is negative and term is not next to a variable or term of type real. In reprogramming, rearrange the expression to place the iΔj next to a real variable.
print	Integer too large to print (>99999)	This is a hard one to get around. Scaling may help, or test before printing. But it is difficult to work with integers larger than the maximum size.
input	Real number too large on data input. *	Scale input data.
digit	Too many digits in input data. *	Round input data, or scale if necessary.

* see corresponding error stops in section 6.2

sin cos	Number too large to allow computing the sin, or cos, even to one digit of accuracy	No comment.
sqrt	Negative argument in square root function	Test before taking square root.
entr	Number too large in absolute value as argument of entier	No comment.
ovflo exp	Argument of exponent function $\geq 2^8$ or so	Scale
ln	Argument of logarithm function (ln) ≤ 0.	Test before taking logarithm.
ovflo intgr	Integer too large ($\geq$32,768) from an iΔj or a float operation.	Reprogram in some way to make the numbers smaller.
order 4 bit	Illegal word after OCNS Attempting to load library tape with 6 bit down	Check 6 BIT, spelling. Lift 6 BIT, press START.

7. Debugging

An important part of any program is debugging, which involves detecting and removing logical errors (grammatical errors are detected at compile-time). Even if there are no run-time error stops, the answers may be incorrect because of a programming error. Sometimes these errors are easily pinpointed, sometimes not. Occasionally, these errors may be caused by round-off error in the computations themselves; there may then be no easy remedy.

Debugging includes two phases--finding the error, and correcting it. With the features discussed below SCALP offers the user valuable tools to aid in these tasks.

7.1 Tracing

One of the simplest ways to track down a source of error is by tracing. Normally, only arguments of the print function are printed out. But by depressing TRANSFER CONTROL you can cause the value of every assignment statement and all statement labels to be printed as they are encountered. You can thus follow your computation as it proceeds statement by statement. The following

program, when run with TRANSFER CONTROL down, will produce the trace shown below.

```
integer'n';'real'x';'
        .
        .
        .
n':='1';'x':='1';'
loop'::'x':='x'*'n';'
n':='n'+'1';'
if'n'lte'3''then'goto'loop';'
next'::'
        .
        .
        .
                .
                .
                .
                n       1
                x    .1000000  01
loop            x    .1000000  01
                n       2
loop            x    .2000000  01
                n       3
loop            x    .6000000  01
                n       4
next            .
                .
                .
```

It should be noted that only the variable name of a subscribed variable is printed, the subscripts themselves are not traced.

Though somewhat time-consuming on a long problem, tracing is the simplest and often the most effective tool for the debugger's use.

7.2 Stop and Continue

At any point during the running of your program you may stop it by lifting BREAKPOINT 8. Performing OCNS will then permit you to use one of the debugging features.

When you wish to continue running your program, make sure

BREAKPOINT 8 and 6 BIT are depressed, then perform OCNS, type "continue", and press START twice.

7.3 Execute Option

At any point in your program you may stop its running and execute any ALGOL statement you wish to type. This option is exercised by lifting BREAKPOINT 8, performing OCNS, depressing BREAKPOINT 8, and typing "execute". Press START twice. You may now type in any ALGOL statement, just as if you were beginning your problem. After typing in, the statement is executed, and further instructions from the typewriter are awaited. Examples of different ways to use the execute option are:

1. Dumping After typing execute, type

'print'('x')';

to cause the printing of the value of variable x. More complicated expressions may be printed; for example

print'('x'+'y'*''17'')';'

will compute and print the value of x + 17y.

2. Changing Values The value of any variable, simple or subscripted, may be changed by executing an assignment statement. For example,

x':=''17.53'429'';'

will change the value of x to 17.53429, while

'x':=''2''*'x';'

will double the current value of x.

3. Transferring You may transfer to any labelled statement by executing a go to statement. For example, to transfer to the statement labelled goof, use

goto'goof';' .

A go to statement may be included with an assignment statement. For example:

begin'x':=''35.2'';'goto'loop'end';'

will assign the value 35.2 to x and then cause a transfer to the statement labelled loop. Notice that since we want to execute two statements we had to group them with a begin ... end.

After executing any statement, you may continue the running of your program by typing "continue" and pressing START twice.

A great deal of freedom and power is available to you under the execute option. In fact, using it permits you to treat SCALP as if it were a desk calculator, although such use would be time-consuming and therefore inefficient. In planning your program, it might be wise to occasionally label a statement solely in anticipation of the possible use of "execute" later on. If no statements in your program are labelled, then after you use an "execute" you type "continue" or restart the running of your program at its beginning by typing "start". It is important to remember that statement labels cannot be twice defined. Therefore, do not use as a statement label any label already defined in your main program.

Execute operates by placing the statement in an unused portion of the machine's memory. After executing the statement, the system "forgets" about it so that the same memory space is available for subsequent executes.

When typing in a program or piece of program through the flexowriter keyboard, you MUST touch START COMP exactly once for each stop code(') you type.

7.4 Patching Your program may be permanently modified by patching. Patching is similar to executing, but here the system places the patch, typed in ALGOL, permanently at the end of your program. A sequence of patches can soon exhaust memory, whereas a sequence of executes cannot.

A patch is a compound statement, and therefore must begin with a begin and end with an end. Declarations of new variables may be made. Patching is always tied to statement labels. Control normally returns to the start of the labelled statement after each time the patch is executed.

As an example, suppose in a for statement you forgot to initialize the running variable, as with

```
        :
    loop':='for'i':='i'+''i'' while'i'lte'n'do'S';'  .
```

We correct by lifting BREAKPOINT 8, performing OCNS, depressing BREAKPOINT 8, typing "patch", pressing START twice, then typing

```
loop'begin'i':=''0''end'
```

The net result will be the equivalent of the following:

```
loop':='goto'patch';'loop 1':='for'i':='i'+''1''while'.  .
```

```
. . . . i'lte'n'do'S';'
patch'::'i':='0';'goto'loop 1';'
```

Thus, we see we can make insertions easily and quickly, but they must always be tied to labelled statements.

Many times we will not want to merely make an insertion, but will want to correct an erroneous statement. We can proceed in exactly the same way but the last statement in the patch must be a goto. Thus, to correct

```
check'::'if'disc'lt''0''then'goto'error';'
    e':='sqrt'('disc')'/''2''*'a';'
    root 1':='-'b'/'('2''*'a')'-'e';'
    root 2':='root1'+''2''*'e';'
here'::'print'---------------
```

one would have to include in the patch all that lies between the labels check and here. After typing patch and pressing START twice, one would type

```
check'[illegible]begin'if'disc'lt''0''then'goto'error';'
        e':='sqrt'('disc')'/'('2''*'a')';'
        root 1':='-'b'/'('2''*'a')'-'e';'
        root 2':='root 1'+''2''*'e';'
        goto'here'end'
```

Obviously, patching would be easier if your program were dotted with unneeded labels, for they could then be available for anchoring patches.

Don't forget that for every stop code (') you must press START exactly once--if you are entering you patch through the flexowriter.

8. ALGOL Check List

Some errors have occurred frequently enough in the past to justify calling them to your attention. Most of them are grammatical or typographical errors.

A stop code must follow every variable, constant, label, or ALGOL-[illegible] symbol, but stop codes must not be used to break up the longer ALGOL-[illegible] words.

Correct	Wrong
integer'	int'eger'
goto'	go'to'
a'['i']	a['i']'
loop'::'	loop::'

In general, be on the lookout for missing or superflous stop codes. A single incorrect appearance or absence of a stop code will almost certainly produce an error.

In general, overuse of the statement separator ; is all right. Its use is superfluous if it is followed by an end. For instance end' may be replaced by end';' , and ;'end' may be shortened to end'. But do not make the mistake of dropping the ;' in end';' -- it may be necessary, especially in for statements.

All array and type declarations must be at the beginning of your program.. If they are not, error stops will occur. Furthermore, all variables used as arrays must be so declared.

Be sure each begin has a matching end. Untold grief and/or error stops can result if the number of begins is not equal to the number of ends. The same goes, of course, for parentheses and brackets.

Never use anything but simple or subscripted variables on the left side of an assignment operator.

In print, read, and title statements the argument (s) is surrounded by parentheses. In particular, a ')' ends a title, and should not be used in title except at the end of the title. Remember that typewriter controls take six letters or characters, but that otherwise no more than five letters between stop codes should be used. Also, don't forget that if you want spaces in your title, you must touch the space bar at the proper point. For upper case, lower case, and other typewriter controls you must type a stop code, insert a typewriter control code, type another stop code, and then continue the title.

All multiplications must be indicated by a * ; mere juxtaposition of variable names is not sufficient, and will cause an error stop.

correct	wrong
a'+tau'*'b';'	a'+'tau'b';'

Don't forget that a'/'b'*'c' in ALGOL stands for ('a'/'b')'*'c' : if you want to divide by the product b*c, you might use a'/'('b'*'c')' or perhaps a'/'b'/'c' .

Spaces must not be used in SCALP programs as they change the meaning of the symbols in which they appear (excepting, of course, in the argument of the title function.) The single exception is that either goto' or go to' may be used interchangeably.

9. SCALP Operating Instructions

Using SCALP is easy. The system tape is loaded using the short bootstrap with the photoreader. If 10.4, the program input routine, is in memory, loading SCALP is done as follows:

1. Bring the computer to NORMAL operate mode with all breakpoint, 6-BIT, and TRANSFER CONTROL switches up; turn on the flexowriter; turn CONNECT to ON.

2. Set the input toggle to TYPEWRITER and turn on the photoreader; press READER STOP to prevent creep.

3. Load the SCALP tape in the photoreader.

4. Perform OCNS.

5. Do: ";0000000", START, ",0000003", START, "i0000", START, "c000$_j$", START, "i0000", input toggle to READER, START.

6. A check sum is taken every 64 instructions. If a check sum error stop occurs, back up the tape until the previous gap of 20 spaces is over the reading mechanism, then press START. If the error persists, consult a Computation Center assistant.

7. Loading may be speeded up by depressing TRANSFER CONTROL, which will cause the check summing to be ignored.

8. After the system tape has been loaded, depress 6 BIT. You are now ready to go.

9. Be sure that the input toggle is set correctly and load your program. If you type it in by hand, you must

press START whenever you would type a STOP CODE ('), which is optional in this case.

10. Don't forget, when loading the library tape of functions, lift 6 BIT. After that, depress 6 BIT.

11. Be sure to depress BREAKPOINT 8 to get continuous running of your program.