NOTES ON THE *MINILOGO* INTERPRETER

by: Charlene Paull

June, 1971

DCS Technical Report #9

Department of Computer Science
Livingston College
Rutgers University

Notes on the MINILOGO Interpreter

Introduction

MINILOGO (2) is a simplified version of LOGO, a symbol manipulating language developed at MIT (3) and Bolt, Beranek and Newman of Cambridge, Massachusetts (1). This version was implemented for IBM 360 on a TSS system as a SNOBOL 4 program. In describing the language, we will discuss (1) the objects represented and manipulated in the language, (2) the basic operations and (3) special commands.

Description of the Language

(1) Objects of the Language

There are two kinds of objects in the language; 'words' and 'sentences'.

(a) A word is a sequence of letters and/or numbers not separated by blanks.

Example of words:

"WORLD" (a sequence of letters)

"1347" (a sequence of numbers)

"W3R7" (a sequence of numbers and letters)

"" (a special word called the empty word- has no letters or numbers)

(b) A sentence is a sequence of words separated by blanks.

Examples of the sentences:

"THE GREEN APPLE"

"MK 456 H7"

"234 567"

In the following discussion of basic operations and special commands, x_i, for $i > 0$, is used to represent the arguments or parameters of the various operations and commands.

(2) Basic Operations:

These elementary operations are used to manipulate the objects of the language to create new objects in the language.

(a) WORD OF x_1 AND x_2: creates a new word by concatenating x_1 and x_2 which must be words.

Example:	Result:
WORD OF "JA" AND "CK"	"JACK"

(b) SENTENCE OF x_1 AND x_2: creates a new sentence by combining x_1 and x_2 which may be either words or sentences.

Example:	Result:
SENTENCE OF "JA" AND "CK"	"JA CK"
SENTENCE OF "HERE COMES" AND "SUN"	"HERE COMES SUN"

(c) FIRST OF x_1: creates a new word by finding the first letter of x_1 if x_1 is a word or the first word of x_1 if x_1 is a sentence.

Example:	Result:
FIRST OF "JACK"	"J"
FIRST OF "JACK JUMPED"	"JACK"

(d) LAST OF x_1: creates a new word by finding the last letter of x_1 if x_1 is a word or the last word of x_1 if x_1 is a sentence.

Example:	Result:
LAST OF "JACK"	"K"
LAST OF "JACK JUMPED"	"JUMPED"

(e) BUTFIRST OF x_1: creates a new word if x_1 is a word by finding all but the first letter of x_1 or creates a new sentence if x_1 is a sentence by finding all but the first word of x_1 (if x_1 was a sentence of two words, the result would be a new word).

Example:	Result:
BUTFIRST OF "JACK"	"ACK"
BUTFIRST OF "JACK JUMPED OVER"	"JUMPED OVER"
BUTFIRST OF "JACK JUMPED"	"JUMPED"

(f) BUTLAST OF x_1: creates a new word if x_1 is a word by finding all but the last letter of x_1 or a new sentence if x_1 is a sentence by finding all but the last word of x_1 (if x_1 is a sentence of two words, the result is a word).

Example:	Result:
BUTLAST OF "JACK"	"JAC"
BUTLAST OF "JACK JUMPED OVER"	"JACK JUMPED"
BUTLAST OF "JACK JUMPED"	"JACK"

Compound Operations

Basic MINILOGO operations can be combined to form compound operations. Since the result of a basic operation is a word or sentence, the arguments of any basic operation can be represented indirectly by specifying a MINILOGO basic operation.

Example:	Result:
FIRST OF LAST OF "JACK JUMPED"	"J"
WORD OF LAST OF "JOHN" AND BUTFIRST OF "POT"	"NOT"

Compound operations of considerable complexity can be formed in this way.

(3) Special Commands

(a) Naming Words and Sentences:

In our previous examples, words and sentences were enclosed in quotation marks. This is one acceptable way of writing objects in the language. MINILOGO objects can, however, be given names and these names used in the program to refer to the specified object. Any word (with the exception of the empty word) can be used as a name for an object. The operation:

NAME OF x_1 IS x_2

declares that x_2 is the name of word or sentence x_1.

Example:

NAME OF "JACK JUMPED" IS TOM

The name TOM can now be used throughout remainder of a MINILOGO program to refer to the sentence "JACK JUMPED".

(b) Using Names:

Once a word or sentence has been named, we can refer to that word or sentence by using either the command:

OBJECT NAMED x_1

where x_1 is the name of a sentence or word or by simply writing:

$/x_1/$

The empty word has a special name; EMPTY. Slashes may or may not be used when referring to the empty word by name.

(c) The Conditional:

The conditional instruction is not one instruction but two; (1) the IS command and (2) the IF. The IS matches two objects which may be either words or sentences and the IF tests the result of this matching. If the two objects are identical, the result of the IS will be YES and NO otherwise.

Form of the Conditional:

IS x_1 EQUAL TO x_2

IF YES x_3

IF NO x_4

As was noted, the IF tests the result of the match. If the match was successful, i.e. if YES was the result of the IS, then do x_3 where x_3 may be any operation or command. If the match was unsuccessful, i.e. the result of the IS was NO, then do x_4 where x_4 may be any operation or command. Both tests need not be present in a conditional.

Examples of Conditionals:	Result:
IS "JA" EQUAL TO "JACK"	NO
IF NO FIRST OF "TOM"	"T"
IS "JA" EQUAL TO "JA"	YES
IF YES LAST OF "TOM"	"M"
IS "1" EQUAL TO "2"	NO
IF YES LAST OF "JOHN"	—
IF NO LAST OF "TOM"	"M"

(d) Defining New Operations:

The TO command allows new MINILOGO operations called procedures to be defined. A procedure is defined by giving the command TO, followed by the name of the procedure and the names of the objects(arguments) that the procedure will use.

Form of the TO command:

TO (procedure name) x_1 AND x_2 AND ...

Following the TO command comes the body of the definition; a series of basic operations, commands or other procedures. The End command terminates the definition of a procedure.

Example of a Procedure Definition:

```
TO ADD Z AND W

NAME OF SENTENCE OF /Z/ and /W/ IS SUM

END
```

This procedure creates a sentence of the objects with the names Z and W and names this new sentence SUM.

Once a procedure has been defined, it may be used where any other MINILOGO operation may be used. Naturally, a procedure must be defined before it is used.

To use a procedure, simply use its name and declare the objects it will use (each object declared separated by the word AND). Using our previous

example, to call or use the procedure ADD, one might write:

ADD "2" AND "3"

If a procedure is used without declaring a value for all of its arguments, those not declared will be treated as if they were the empty word. If a procedure is used and more values are declared than are needed, the excess values will be ignored.

(e) PRINT x_1: This command cause x_1 to be typed out at the terminal. x_1 may be a word, sentence or the result of any operation.

Examples:

PRINT "JACK"

PRINT "JACK JUMPED"

PRINT LAST OF "TOM"

PRINT /TOM/

(f) STOP: This command terminates the execution of a MINILOGO program.

(g)RETURN: This command is normally used within a procedure and is used to halt the execution of a procedure.

(h) END: Indicates the end of a procedure definition.

(i) .END: Indicates the end of a program.

(j) REQUEST: This command allows objects that are to be used in a program to be typed in from the terminal during the execution of that program.

Example: NAME OR REQUEST IS NOW

Idiosyncrasies of the Language

Use of Quotation Marks:

If the quotation mark should occur at the end of a line of program (i.e. is the last symbol in that line), that line will be ignored. This happens because the quotation mark is used by the TSS system as the "line kill" symbol. To avoid this problem, add one or more blanks to the end of the line of program involved.

The Operation of the MINILOGO Interpreter

The interpreter is a SNOBOL4 program which accepts MINILOGO programs as input from a TSS terminal. It then analyzes and executes the program line by line. Some limited editing facilities are available.

The user types in a program line by line; each line being given a number by the interpreter which can be used for editing.

Example of the Format of MINILOGO programs:

```
MINILOGO      6/70                          (title printed by interpreter)
1 > NAME OF "ART" IS FINE
2 > PRINT /FINE/
3 > .END
```

Note: the symbol > indicates that the interpreter is waiting for input from the terminal.

Analysis and execution of the program does not begin until the .END command is encountered. At present, the size of a MINILOGO program is limited to onethousand instructions. This can be increased by increasing the size of the input buffer of the interpreter. However, this interpreter was designed with the expectation that the type of programs it would deal with would not be of a complex or sophisticated nature nor would they be of any considerable length since the language itself is limited. It was felt that the current

size was a reasonable one.

The interpreter operates in two distinct modes. First each line of program is analyzed (parsed) and then the indicated operation(s) are performed. Since MINILOGO instructions can be compound, the normal mode in analyzing an instruction is to consider it compound. Each line of program is examined left to right for basic operations, commands, or procedure calls. For each operation, command or procedure, there is a corresponding SNOBOL4 pattern definition. A line of program is searched for a match with one of these pattern definitions. When a match is found, an entry is made in each of two lists; (1) the OPLIST and (2) the PARAMLIST. An entry in the OPLIST contains the name of the operation, command or procedure found and the address(es) of its parameters(arguments) which are stored in the PARAMLIST. To handle compound instructions, each entry also has an address in the PARAMLIST where the result of this operation is to be stored.

Consider the following line of program:

WORD OF "JACK" AND "JOHN"

After pattern matching, the OPLIST and PARAMLIST would have the following entries:

	PARAMLIST	OPLIST
1	JACK	1,WORD,1,2
2	JOHN	

address in PARAMLIST where result is to be stored

addresses of parameters in PARAMLIST

Each parameter in PARAMLIST is then examined for operations, commands or procedures. More entries would be made in both lists for each subsequent pattern match. This process would continue until no more patterns can be matched. The following example shows how the two lists would look for a compound instruction.

table. If ADD was the name of a procedure, the definition of ADD would be stored in ADD2, ADD3, etc.

When a procedure call is recognized, the value of the call arguments are assigned the names of the definition arguments. The definition arguments are considered to be the names of MINILOGO objects, namely the values of the call arguments of a procedure. If a call for another procedure is made within the body of a procedure, the values of the call arguments are saved in a push down list the name of which is the name of the procedure concatenated with a *. When a return is made to a procedure from another procedure, the call values are restored and the push down list popped. This allows for recursive use of procedures and for more than one procedure using the same argument names in their definitions.

Analysis and execution of the procedure definition takes place until the END or RETURN command is recognized.

There are no transfer commands such as GO TO (label) in this version of the interpreter nor is any facility provided for continuation of one line of program onto the next.

Editing Facility

A limited editing facility is provided. At any point while typing in a program or at the end of the execution of a program, when the symbol > appears, the command EDIT can be given. This allows the user to change any part of his program without retyping the whole program. After the EDIT command is given, the interpreter responds with a >. The user then specifies the number of the line he wishes to change, followed by the change. He must, however, retype the whole line.

Example:

```
1  >  TO TEST
2  >  FIRSTT OF "JACK"
3  >  EDIT
>  2  FIRST OF "JACK"
>  END
3  >
```

In this example, the user has noticed a mistake in line 2 of his program. He gave the EDIT command and after the interpreter responded he specified the line number, 2, and the change, FIRST OF "JACK". The interpreter will keep asking for more editing changes until the command END is given indicating that all changes have been completed. The interpreter then responds with the next line number, in our example; 3. Any editing done after the execution of a program will cause an automatic rerun or re-execution of the program.

Interpreter Commands

(a) EDIT: allows changes to be made in the program

(b) END: signifies the end of any editing

(c) EXECUTE: This command can be used at the end of the execution of a program and allows the user to rerun his program. Particularly useful with programs using the REQUEST command.

If, at the termination of a program, any command, other than the ones listed above, is given the interpreter will give a termination message and stop.

Diagnostic Messages and Error Terminations

INCORRECT SYNTAX OR UNDEFINED INSTRUCTION
INSTRUCTION NUMBER(line number given by interpreter)

Execution of program is halted with the first such error and user is given opportunity to edit.

INCORRECT SYNTAX OR UNDEFINED INSTRUCTION
INSTRUCTION (number) OF PROCEDURE (name)

Same as above.

PARAMLIST OVERFLOW
OPLIST OVERFLOW
OUT OF RANGE OF PARAMLIST

These will cause the interpreter to stop completely. No opportunity is given for editing since these errors are usually internal to the interpreter.

MISSING END COMMAND FOR PROCEDURE DEFINITION

Execution halts if no END command is found before .END command. This occurs while defining a procedure, i.e. during the execution of a TO command. Opportunity is given to edit.

LOGO INPUT BUFFER OVERFLOW

The input program is too big for the allotted program storage of the interpreter. At this time maximum size allowed is one thousand lines of program. Opportunity is given to edit.

INVALID INSTRUCTION NUMBER

This occurs during editing. The number specified is not recognizable by the interpreter, i.e. it is not an integer.

INCORRECT FORMAT CHANGE CANCELLED

The correct format for specifying a change while editing was not observed. The correct format is an integer, blanks, the correction.

References

(1) Programming-Languages as a Conceptual Framework for Teaching Mathematics; Bolt, Beranek and Newman Inc.; Report No. 1889; November 30, 1969; W. Feurzeig, S. Papert, M. Bloom, R. Grant, C. Solomon

(2) Notes on a Procedural Language called Minilogo, and its Use In Problem Solving; Lecture Notes; Dept. of Computer Science, Livingston College; November 7, 1969; Dr. S. Amarel

(3) Conceptual Advances Derived From the Muzzey Experiment; MIT Artificial Intelligence Laboratory; January 29, 1970; S. Papert and C. Solomon

NOTE: Since the interpreter is a SNOBOL4 Program, if it gets into difficulty a SNOBOL4 error message and termination may occur.

Summary

The main purpose of the MINILOGO interpreter was to give a student unfamiliar with programming a facility for studying recursive techniques in a non-sophisticated languge. Consequently, this version does not have any arithmetic or transfer commands. It does not allow the user to create program files nor can it read programs from any file. All input is expected to be typed in from a TSS terminal. Program size is limited but could be incrcased if so desired. Some sample programs are included in Appendix A.

APPENDIX A:

```
MINILOGO          6/70
1 > to hanoi no. and p1 and p2 and p3
2 > is /no./ equal to 1
3 > if yes mprint /p1/ and /p2/
4 > if yes return
5 > hanoi butfirst of /no./ and /p1/ and /p3/ and /p2/
6 > hanoi 1 and /p1/ and /p2/ and /p3/
7 > hanoi butfirst of /no./ and /p3/ and /p2/ and /p1/
8 > end
9 > to mprint 11 and 12
10 > print "move one disk from"
11 > print sentence of /11/ and sentence of "to" and /12/
12 > end
13 > hanoi 111 and peg1 and peg2 and peg3
14 > .end
MOVE ONE DISK FROM
PEG1 TO PEG2
MOVE ONE DISK FROM
PEG1 TO PEG3
MOVE ONE DISK FROM
PEG2 TO PEG3
MOVE ONE DISK FROM
PEG1 TO PEG2
MOVE ONE DISK FROM
PEG3 TO PEG1
MOVE ONE DISK FROM
PEG3 TO PEG2
MOVE ONE DISK FROM
PEG1 TO PEG2
>
TERMINATION MINILOGO
```

```
MINILOGO          6/70
1 > to reverse string
2 > is /string/ equal to empty
3 > if yes print /answer/
4 > if yes stop
5 > name of word of first of /string/ and /answer/ is answer
6 > reverse butfirst of /string/
7 > end
8 > reverse request
9 > .end
> computer
RETUPMOC

>
TERMINATION MINILOGO
```

```
MINILOGO          6/70
1 > to triangle it
2 > is /it/ equal to empty#
3 > edit
> 2 is /it/ equal to empty
> end
3 > if yes stop
4 > print /it/
5 > triangle butfirst of /it/
6 > end
7 > triangle request
8 > .end
> polyhedron
POLYHEDRON
OLYHEDRON
LYHEDRON
YHEDRON
HEDRON
EDRON
DRON
RON
ON
N


> execute
> square
SQUARE
QUARE
UARE
ARE
RE
E

>
TERMINATION MINILOGO
```

APPENDIX B: Summary of Language

Basic Operations:

WORD OF x_1 AND x_2

SENTENCE OF x_1 AND x_2

FIRST OF x_1

LAST OF x_1

BUTFIRST OF x_1

BUTLAST OF x_1

Special Commands

NAME OF x_1 IS x_2

OBJECT NAMED x_1

/x_1/

IS x_1 EQUAL TO x_2

IF YES x_3

IF NO x_4

TO (name of procedure) x_1 AND x_2 ...

PRINT x_1

STOP

RETURN

END

.END

REQUEST

Editing Commands

EDIT

END

EXECUTE