QuickShot

SVI·2000C
ROGO LANGUAGE CARD

Robotarm

INSTRUCTION MANUAL

SVI·2000C
ROGO LANGUAGE CARD

Robotarm

INSTRUCTION MANUAL

Scanned and converted to PDF by HansO, 2001

Published by
SPECTRAVIDEO INTERNATIONAL LTD.

First edition
First printing 1986

Copyright ©1986 by Spectravideo International Ltd.

Spectravideo International Ltd. shall not be liable in any event for claims of incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Every effort has been made to supply complete and accurate information in this manual. Nevertheless, due to our never ending commitment to improve both product design and performance, we reserve the right to change product specifications at anytime without prior notice.

No part of this publication may be stored in a retrieval system, transmitted, or reproduced in any way, including but not limited to photocopy, photograph, magnetic or other record, without the prior agreement and written permission from Spectravideo International Ltd.

Registered trademarks used in this manual are:
Spectravideo SVI-2000 and SVI-2000C are trademarks of Spectravideo International Ltd.
MSX is a trademark of Microsoft Corporation.

RADIO INTERFERENCE

This equipment generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been designed to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If this equipment does cause interference to radio or television reception, which can be determined by turning equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient the receiving antenna
- Relocate the computer with respect to the receiver
- Move the computer away from the receiver
- Plug the computer into a different outlet so that computer and receiver are on different branch circuits.

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful: "How to Identify and Resolve Radio-TV Interference Problems". This booklet is available from the U.S. Government Printing Office, Washington, DC 20402, Stock No. 004-000-00345-4.

WARNING 1:

This equipment has been certified to comply with the limits for a class B computing device, pursuant to Subpart J of Part 15 of FCC Rules.

WARNING 2:

The user is warned that the shielded cables provided with this equipment must be used. A failure to use shielded cables may result in excessive radio-frequency emissions in violation of FCC rules, for which the user would be responsible. If any extension cables are used, they must also be shielded and the shields connected by means of metal shell connectors so that there is a full 360 degrees of connection; digital connection are not good enough for radio frequencies.

TABLE OF CONTENTS

CHAPTER PAGE

APPENDICES

ILLUSTRATIONS

CHAPTER 1
INTRODUCTION

The SVI-2000C Rogo is specially designed to control the SVI-2000 Robotarm through an MSX computer. With the Rogo Cartridge and Robotarm properly installed in your MSX machine, you will have lots of fun programming the movement of the Robotarm while you are actually learning a language very close to Logo, which is another high-level language well-known for structured programming.

As a matter of fact, Rogo owes much to Logo. Both use English-like keywords and allow room for USER-DEFINED COMMANDS. The construction of a program, or PROCEDURE as it is called in Logo, is also similar. A procedure is made up of a series of user-defined commands which are composed of built-in commands or other user-defined commands.

In spite of these similarities, Rogo and Logo are somewhat different. In Rogo, the turtle graphic commands of Logo are replaced by commands that control the movement of the actual Robotarm and its image on the GRAPHIC SCREEN. Furthermore, some of the string manipulation commands of Logo are omitted to reduce the overall size of the language and thus allow more room for user-written procedures.

HOW TO USE THIS MANUAL

This manual provides a thorough description of all the primitives in Rogo as well as the ins and outs of defining and editing your own commands. It is made up of 12 chapters and 4 appendices:

Chapters 1 to 3 describe the basic operation of the Rogo Interface. They give you a brief idea of what Rogo is and help you install the Rogo Cartridge. The editing abilities of the Rogo Command Editor are discussed in Chapter 4.

The 7 chapters that follow are devoted to describing the capabilities of the Rogo primitives. Each chapter concentrates on one category of primitives. These chapters are arranged in a sequence that ensures you will pick up Rogo quickly and easily. The Rogo Graphic Screen and Commands are then left to the last chapter of the manual.

As for the appendices, they contain supplementary material intended primarily for the more advanced users. Appendix A is a table of the error messages and possible remedies. A handy reference sheet of all the Rogo commands and operations is included in Appendix B. Appendix C is a set of useful sample procedures that will, hopefully, inspire you to write more sophisticated procedures. Last but not least, Appendix D describes the recursive commands which, if handled properly, can greatly enrich your command of Rogo.

CHAPTER 2
INSTALLATION

The installation of the SVI-2000C Rogo Cartridge is very simple. The steps are as follows:

1. Switch off the power of your computer.

2. Insert the Rogo Cartridge into the cartridge slot. Make sure the Joystick Sockets on the cartridge are on your right hand side.

   CAUTION: 1. Faulty insertion may short circuit your computer.

   2. The Cartridge can only be run on MSX computers with 64K RAM or more.

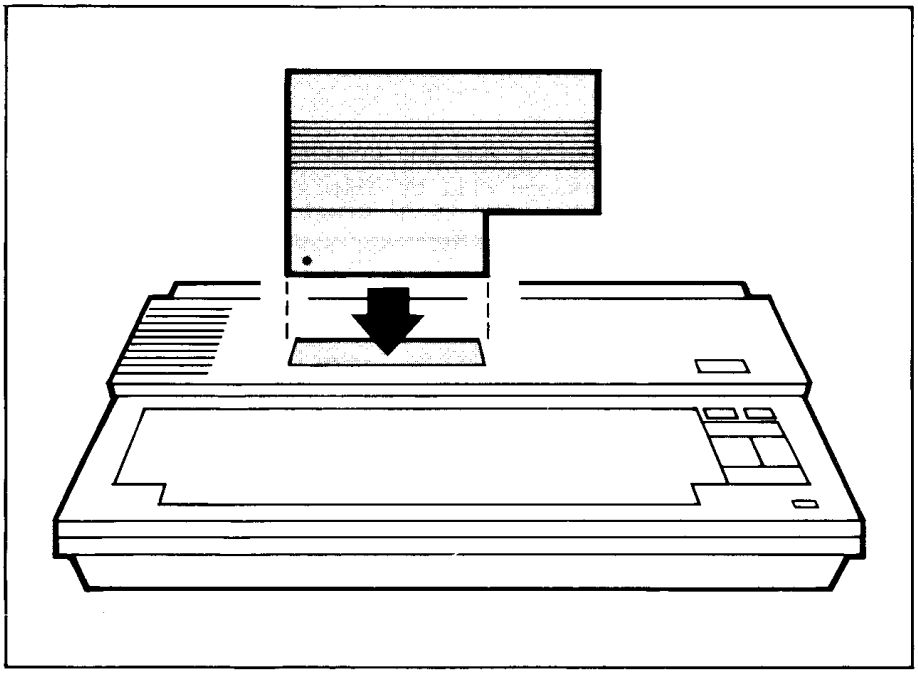


Fig. 2.1 Inserting the Rogo Cartridge

3. Connect the Rogo Cartridge to the Robotarm with the two cables provided. Note that the Joystick Socket for Joystick 2 is on top of that for Joystick 1.

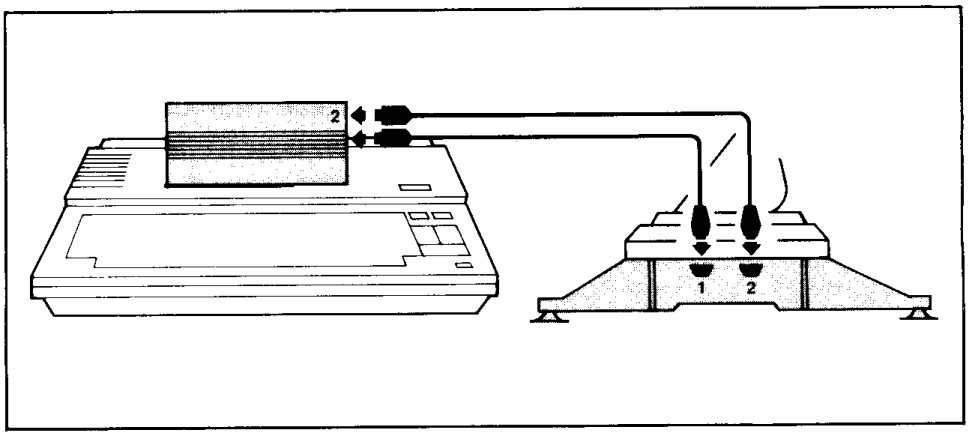


Fig. 2.2 Connecting the Rogo Cartridge to the Robotarm

4. Switch on the monitor and the computer. The Text Screen with a welcoming message will be displayed:

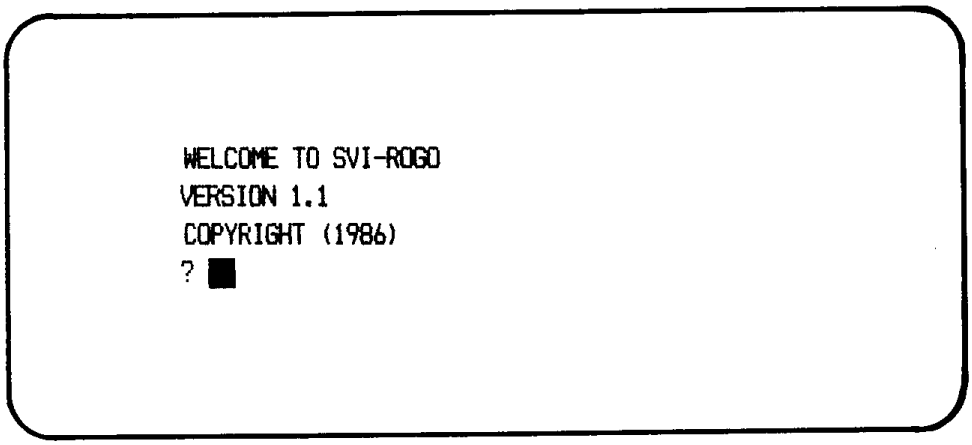


Fig. 2.3 The Text Screen and the Welcoming Message

The question mark, ?, is the prompt that tells you Rogo is ready to receive messages. The cursor next to it indicates where the next character you type will be displayed.

CHAPTER 3
OVERVIEW

3.1 WHAT IS ROGO

Rogo is a programming language designed to control the SVI-2000 Robotarm through an MSX computer. Just like any other high-level language, Rogo uses English-like keywords to interact with the users. It is easy to learn and use. What's more, Rogo encourages a modular, structured approach to programming--a technique which can be applied to other more sophisticated programming languages and which is useful for teaching logic and problem solving.

3.2 HOW ROGO WORKS

Rogo responds to 2 kinds of commands: the PRIMITIVES and the USER-DEFINED COMMANDS.

Primitives

The primitives are the built-in commands which act as the first layer of building blocks in a Rogo program. Primitives can neither be re-defined nor edited. To have a peek at the primitives, type COMMAND and press ENTER.

```
BUILT IN COMMANDS IN ROGO:
BC                      BA                      LU
LD                      FU                      FD
WC                      WA                      FC
FO                      SHOWARM                 HIDEARM
SHOWTEXT                HIDETEXT                CLS
PRINT                   OP                      LOCATE
REPEAT                  STOP                    TO
EDIT                    TEST                    IF
IFTRUE                  IFFALSE                 MAKE
COMMAND                 DIR                     SAVE
LOAD                    SELFTEST                WAIT
RUN                     THING                   READC
READL                   J1                      J2
OR                      AND                     NOT
XOR                     SUM                     DIFF
PROD                    QUOT                    INT
ROUND                   BF                      EMPTYP
FIRST                   LPUT                    SE
WORD                    LISTP                   WORDP
NUMBERP
?
```

Fig. 3.1 The Built-in Commands in Rogo

The primitives are categorized in 9 groups: Robotarm Control Commands, Screen Commands, Variables, Arithmetic Operations, Logical Operations, Control Flow Commands, I/O Commands, Word/Lists Command and To/Edit Commands. We will discuss each of these in detail later.

Executing a primitive is simple. You have already executed one primitive, COMMAND, now try another:

1. Type the name of the primitive in the Text Screen. Whatever you type will be displayed in the upper case.

   EXAMPLE:

   ? BC

   BC is the primitive to rotate the base of the Robotarm clockwise.

2. Give the appropriate inputs (arguments).

   EXAMPLE:

   ? BC 100

   BC 100 will rotate the base of the Robotarm clockwise for 100 time units.

   NOTE : Not every primitive takes an input, but many of them do. Read the respective chapters for details.

3. Press ENTER. This asks Rogo to take notice of and process what you have typed.

   NOTE : You must press ENTER at the end of each command line.

Once you have pressed ENTER, the base of your Robotarm rotates clockwise.

Whenever you enter something, Rogo checks it against its table of primitives and, if found, executes it in the way you have specified. If it is not a primitive, Rogo will continue to check against the table of commands you have defined earlier, if any. Rogo will then execute the command if found or display an error message telling you it doesn't know how to do if not (see Appendix A for details).

Rogo can execute more than one primitive simultaneously. All you have to do is to enter all of the commands on the same logical line.

A logical line can be up to 254 characters in length and is delimited by a carriage return (ENTER).

EXAMPLE:

```
? BC 200 LU 150 FU 200 WC 150 FC 200 FO 200 WA 100 FD 150 LD 80
BA 200                                                    ENTER
```

LU, FU, WC, FC, FO, WA, FD, LD, and BA are the primitives that control the various parts of the Robotarm.

NOTE : 1. A space must be inserted between a primitive and its input.
2. Rogo can only read a maximum of 254 characters at a time.
3. If ROGO encounters two or more control commands acting on the same axis, it will execute each command, one at a time, in sequence.

User-defined Commands

Rogo is a very versatile language. If you don't find the primitives adequate, you can define your own commands to suit your specific needs. Not only that. You can edit the defined commands whenever necessary.

Defining and editing a command take place in the Command Editor. Before you can edit a command, you must define it first. Let's define a new command called MOVE.

1. Type TO in the Text Screen.

   EXAMPLE:

   ? TO

   TO is the primitive that defines a new command.

2. Type the name of the command you are going to define. Precede the name with a double quotation mark, ".

   EXAMPLE:

   ? TO "MOVE

   NOTE : A command name can consist of up to 9 letters, numbers, and/or symbols, but the first character must be a letter.

3. Press ENTER. You are brought to the Command Editor:

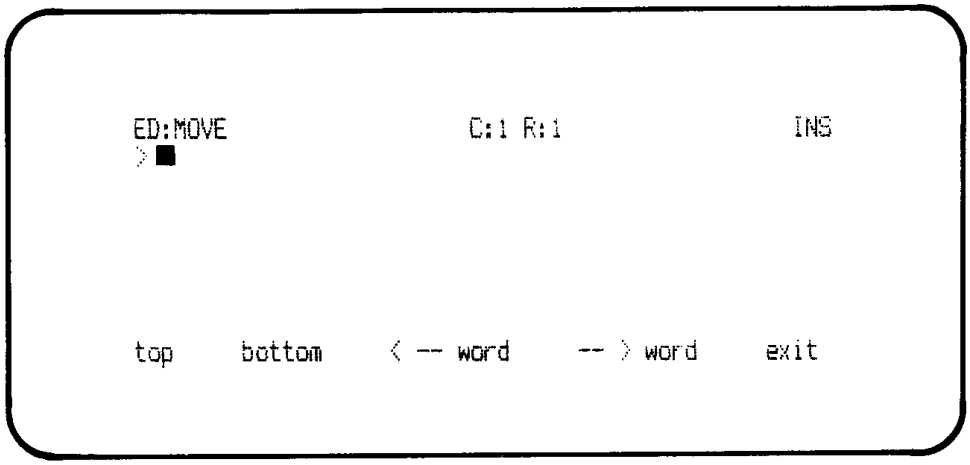


Fig. 3.2 The Command Editor

The name of the new command will appear at the top of the screen and functions of the first five function keys at the bottom. The Rogo Command Editor supports full-screen editing. You can move the cursor around the screen, type over any mistakes, as well as insert and delete text. Note that the prompt in the Command Editor becomes a >.

4. Define your command. You can define as many lines as you wish.

   EXAMPLE:

   >PRINT [HELLO, GLAD TO MEET YOU.] ENTER

   >LU 200 FU 300 ENTER

   >WC 300 ENTER

   PRINT is a primitive that tells Rogo to print its input. Apart from primitives, you can incorporate user-defined commands in building a new command. More about this in the next section.

5. Type END in a fresh line to indicate you have finished defining the command. (Then press ENTER)

6. To exit the Command Editor, enter ^Z or press F5. You are then back in the Text Screen.

   Executing a user-defined command is the same as executing a primitive. Type MOVE and press ENTER.

   If you want to edit the command MOVE.

   1. Type EDIT and enter the command name. Don't forget the space and the double quotation mark.

      EXAMPLE:

      ? EDIT "MOVE ENTER

You are then in the Command Editor and the lines you have defined in MOVE are displayed.

2. Edit your command. The steps are the same as defining a new command.

3.3 BLOCK-BUILDING IN ROGO

In Rogo, a program is better known as a procedure. A procedure can be composed of primitives only, as in the case of MOVE. But most often a procedure contains both primitives and other user-defined commands.

EXAMPLE:

? TO "MOVE2

In the Command Editor you type:

>MOVE

>BA 200 BC 300

>LD 200 FD 300

>END

>^Z

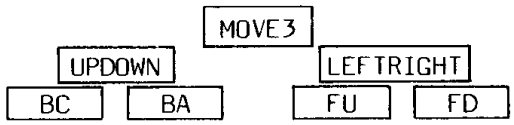
If you tell Rogo to execute MOVE2, it will execute MOVE first and then sequentially the other primitives in MOVE2.

Now you see you can use primitives as building blocks to define new commands, and then use the defined commands to build still more user-defined commands.

EXAMPLE:

```
?  TO "UPDOWN

>FU 200 FD 300

>END

>^Z

?  TO "LEFTRIGHT

>BC 200 BA 200

>END

>^Z

?  TO "MOVE3

>UPDOWN

>LEFTRIGHT

>END

>^Z
```

MOVE3 is built on top of 2 user-defined commands, UPDOWN and LEFTRIGHT. This block-building process is best explained by a diagram:



Thus, a Rogo procedure is written from the bottom up: Primitives are combined to build user-defined commands, which are, in turn, combined to produce more complex commands and so on in building block fashion until one or two commands can execute the entire procedure.

3.4 SAVING AND LOADING A PROCEDURE

When you define a procedure, the procedure is stored in the RAM (Random Access Memory) of the computer. Everything in the RAM will be lost if you switch off the computer. Surely you don't want to retype your procedures everytime you turn on the computer. You can save the procedures on cassette tapes and load them back into the RAM whenever necessary.

Preparing a Cassette Tape

1. Insert a tape into a cassette recorder that works with your MSX computer.

2. Rewind the tape.

3. Press STOP on the cassette recorder.

Saving a Workspace

A workspace is a part of the RAM where the procedures are stored. If you save a procedure, all the other procedures in the RAM will be saved too. The steps are:

1. Prepare a tape.

2. Type SAVE "CAS on the Text Screen. DON'T press ENTER yet.

3. Press PLAY and RECORD on the recorder.

4. Wait until the blank tape has passed, then press ENTER.

   Rogo will start saving all the procedures in the RAM.

5. Press STOP on the recorder.

Read 10.7 for details.

Loading a Workspace

1. Prepare the tape that holds the workspace.

2. Type LOAD "CAS on the Text Screen and press ENTER.

3. Press PLAY on the recorder.

   Every procedure in the workspace will be loaded into the RAM.

   If the workspace can't be loaded, repeat the procedure from step 1. If the problem persists, refer to Appendix A for a solution.

4. Press STOP on the recorder.

Read 10.8 for details.

3.5 THE GRAPHIC SCREEN

Apart from controlling the Robotarm, Rogo can simulate the actual movement on the Graphic Screen. To invoke the Graphic Screen, type SHOWARM.

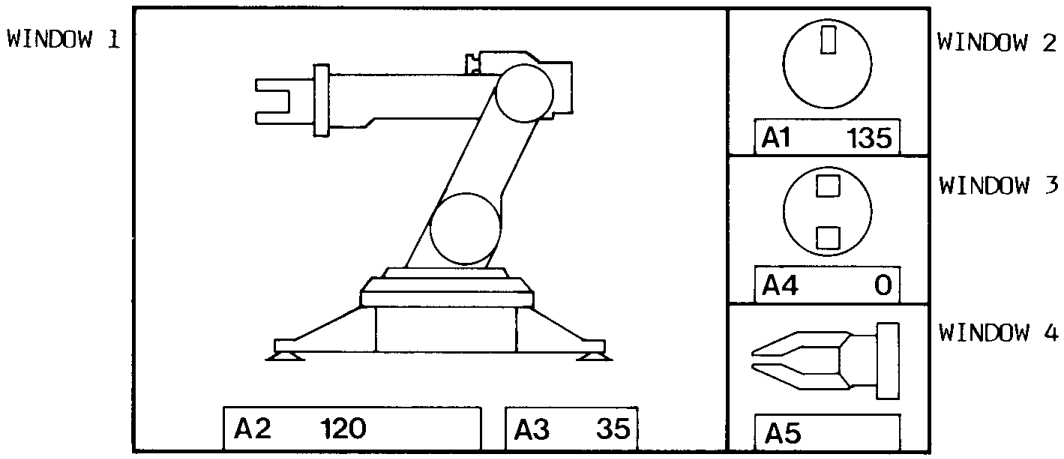


Fig. 3.3 The Graphic Screen

The Graphic Screen is made up of 4 windows, each of which displays a different perspective of the Robotarm. The Graphic Screen can work with the Screen Commands to preview the movement of the Robotarm. To exit the Graphic Screen, type HIDEARM. The details will be discussed in Chapter 12.

3.6 ROGO TERMINOLOGY AND SYNTAX

Before we look at specific commands, there are 2 things we have to make clear: the definition of some Rogo terms and the usage of punctuations in Rogo.

Definitions

COMMAND

A command is a keyword that causes Rogo to perform a specific function. It may require you to enter certain inputs to make it effective. There are 2 kinds of commands in Rogo, the PRIMITIVES and the USER-DEFINED COMMANDS.

The primitives are the BUILT-IN COMMANDS which cannot be altered. There are totally 64 primitives in Rogo.

The user-defined commands are the commands that you define on top of the primitives or other user-defined commands.

A command name can consist of up to 9 letters, numbers and/or symbols, but the first character must be a letter.

WORD

A word is distinguished from a command by a double quotation mark preceding it. It must not exceed 254 characters.

EXAMPLE:

```
? PRINT "HELLO
```

HELLO is a word and PRINT is a primitive. A word can be made up of any combination of alphanumeric characters and is delimited by a space. A word need not be preceded by a double quotation mark if it begins with a number.

LIST

A list is enclosed by a pair of square brackets, [ ]. It consists of unquoted words. The maximum number of characters in a list is 254.

EXAMPLE:

```
? PRINT [HOW OLD ARE YOU? I'M 100.]
```

HOW, OLD, ARE, YOU, ?, I'M, 100, ., are all elements of the list.

INSTRUCTIONLIST

An instructionlist is an instruction or set of instructions enclosed by a pair of square brackets, [ ].

EXAMPLE:

```
? IF 2 > 1 [PRINT [GREATER THAN]]
```

IF is a conditional primitive and it takes a condition, 2 > 1 in this case, and an instruction (PRINT [GREATER THAN]) in form of a list.

VARIABLE

A variable is a "container" which holds a certain value.

EXAMPLE:

```
? MAKE "COUNT 1
```

MAKE is a primitive that assigns a value to a variable. In the above example, COUNT is the variable and 1 is its value.

OBJECT — A Rogo object can be a word, list, the content of a variable, or the result of a Logical Operation

OPERATION — An operation is a primitive that causes Rogo to output a certain result. It is different from a command in that it does not tell Rogo what to do with the output. All arithmetic and logical manipulations are operations.

EXAMPLE:

```
? SUM 2 2
```

SUM is an Arithmetic Operation that adds 2 inputs. If you enter the above example alone, Rogo will calculate and put the result in its memory without telling you anything. If you want to print the result, you must say so.

EXAMPLE:

```
? PRINT SUM 2 2
```

You will get 4 in this case.

Punctuations

There are 3 punctuation marks that have special meanings to Rogo: the double quotation mark, ", the square brackets, [ ], and the colon, :.

" — The double quotation mark is used immediately before a WORD to distinguish it from a command.

[ ] — Square brackets are used to delimit a LIST and an INSTRUCTIONLIST composed of words or lists which in total do not exceed 254 characters. The square brackets must be in pairs. Otherwise, an "IMBALANCED PARENTHESES!" error will occur.

: A colon must be placed immediately before a word which is to be treated as a VARIABLE. The name of a variable can be any combination of letters and numbers as long as it begins with a letter and does not exceed 9 characters.

EXAMPLE:

```
? MAKE "SVI "ROGO

? PRINT :SVI

ROGO
```

Notice the colon after PRINT. It specifies the word SVI is a variable and PRINT asks Rogo to output the value of that variable.

CHAPTER 4
THE COMMAND EDITOR

4.1 THE SCREEN AND EDITING KEYS

The Command Editor is the screen where you define a new command or edit a defined command. Editing in the Command Editor is different from what you do in the Text Screen where you use BACKSPACE to move back and type over an error. Before we introduce the editing keys, we would like to say a few words on the screen.

The Editing Screen

The Editor displays 22 lines at one time. When the cursor gets to the bottom of the screen, the screen automatically scrolls the whole screen up one line. If you want the screen to scroll down one line, move the cursor to the top of the screen and press ↑ once. Keep pressing ↓ will scroll the screen up continuously, and the screen will scroll down continuously if you keep pressing ↑.

Every line on the screen can hold a maximum of 254 characters, but only 39 characters, including the prompt, can be displayed simultaneously. If you enter more than 38 characters, the screen will scroll rightward 8 characters each time to make room for you. To show what we mean, type the following:

```
? TO "TRIAL

>PRINT [EVERY LINE ON THE SCREEN CAN HOLD A MAXIMUM OF 254
 CHARACTERS, BUT ONLY 39 CHARACTERS, INCLUDING THE PROMPT, CAN
 BE DISPLAYED SIMULTANEOUSLY.]
```

Note how Rogo scrolls the screen rightward. An "+" sign is displayed at the rightmost column to indicate that some characters are beyond the display of the screen. If more than 254 characters are entered for a single line, Rogo will ignore the extras. You can use ← and → to scroll rightward and leftward.

The cursor will not move to the next prompt unless you press ENTER to show you have finished defining the line.

Editing Keys

In the Editor, the following function and control keys are applicable:

KEY	DESCRIPTION
F1 or ^T	Moves the cursor to the beginning of the procedure.
F2 or ^B	Moves the cursor to the bottom of the procedure.
F3 or ^A	Moves the cursor one word to the left.
F4 or ^F	Moves the cursor one word to the right.
F5 or ^Z	Exits the Command Editor.
F6 or ^P	Moves the screen up 10 lines.
F7 or ^N	Moves the screen down 10 lines.
F8 or ^D	Deletes the line containing the cursor.
F9 or ^I	Inserts a line where the cursor stands.

KEY	DESCRIPTION
F10 or ^Q	Aborts the Command Editor without saving. An "Abort edit (Y or N)?" message will be displayed to make sure you really want to abort the Editor without saving the procedure. Confirm by typing Y.
DEL	Deletes the character immediately to the left of the cursor.
INS ON/OFF	Toggles between INS ON/OFF. Inserts a character where the cursor stands when ON and overwrites when OFF. A message "INS" is displayed on top of the Editor when the mode is INS ON. INS ON is the default mode.
→	Moves the cursor one character to the right.
←	Moves the cursor one character to the left.
↑	Moves the cursor one line up.
↓	Moves the cursor one line down.

For quick reference, the following information is displayed at the bottom of the Command Editor:

top bottom ← word → word exit

which indicate the application of F1, F2, F3, F4, and F5 respectively.

On pressing SHIFT, the information changes to:

page ▲ page ▼ del-ln ins-ln abort

which indicate the application of F6, F7, F8, F9, and F10 respectively.

Two other messages are displayed at the top of the screen:

C: and R:

The number following C: shows the column number of the cursor and that following R: is the row number.

Overflow Warning

There are times when you define or edit an excessively long procedure. If the procedure gets too long, the Editor will probably run out of memory and display an overflow warning telling you how many character-spaces are left in the memory. The warning will appear whenever the space is less than or equal to 20 characters.

EXAMPLE:

WARNING: SPACES LEFT ARE LESS THAN 18

It means that the Editor for that particular procedure can accept less than 18 characters. You are advised to exit the Editor. When the memory is full, no more characters can be entered unless you delete some lines.

4.2 SAMPLES SECTION

EXAMPLE:

```
?  TO "HELLO
```

You are now in the Command Editor. Notice that there is a ED:HELLO message displayed at the top left-hand corner reminding you that you are defining a new command HELLO. Type:

```
>PRINT [WHAT'S THE PRODUCT OF 123 AND 321?]
```

Notice how it fits in the screen. Let's say we have finished defining HELLO. Type:

```
>END

>^Z
```

Don't forget you can press F5 instead of CTRL-Z. If you want to edit HELLO, type:

```
? EDIT "HELLO
```

Note that a ED:HELLO message is displayed on the Command Editor and the lines you have defined previously are listed. Let's add a few lines to the procedure. The cursor is at the beginning of the first line. Use the editing keys to move the cursor to the beginning of END. Overtype with the following lines:

```
>PRINT PROD 123 321

>END

>^Z
```

You have defined quite a number of procedures. If you want to review what they are, type DIR and press ENTER. The screen will be cleared and a message PROCEDURES INSIDE ROGO: will be displayed. Underneath the message is the directory of all the procedures stored in the RAM. The names of the procedures will be listed in the order in which they were defined.

4.3 A POINT TO REMEMBER

Never use the same command name twice in defining a new command; otherwise, it will erase the previous definitions of the command. For instance, type TO "HELLO and press ENTER. See what happens. All the lines you have defined and edited earlier are gone.

CHAPTER 5
ROBOTARM CONTROL COMMANDS

This category consists of 10 primitives: BC, BA, LU, LD, FU, FD, WC, WA, FO, and FC. They rotate the base of the Robotarm, and move the forearm, lowerarm, wrist, and forceps of the Robotarm for a specified period of time. Before we go on, let's look at the various parts of the Robotarm and the respective axes and primitives that control the parts.

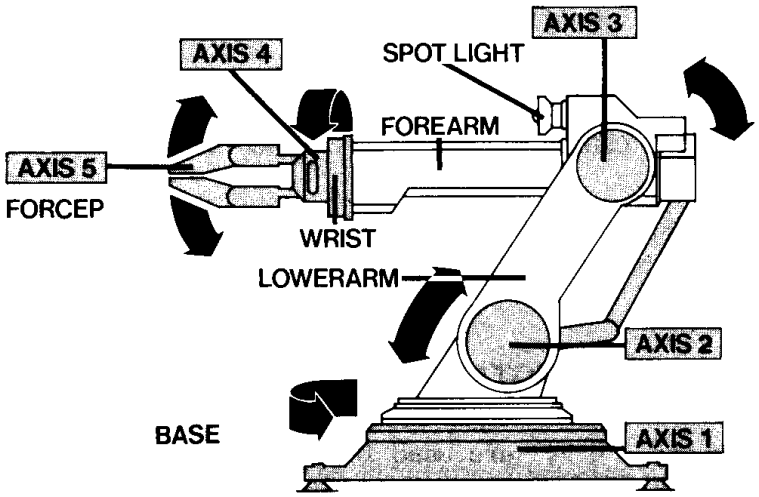


Fig. 5.1 The Parts and Axes of the Robotarm

Primitives controlling the base:	BC, BA
Primitives controlling the lowerarm:	LU, LD
Primitives controlling the forearm:	FU, FD
Primitives controlling the wrist:	WC, WA
Primitives controlling the forceps:	FO, FC

To get your "feet wet", here is a procedure using all of the Robotarm Control Commands:

```
? TO "GREETINGS

>PRINT [BASE CLOCKWISE] BC 250

>PRINT [BASE ANTICLOCKWISE] BA 250

>PRINT [LOWERARM UP] LU 200
```

```
>PRINT [LOWERARM DOWN] LD 200

>PRINT [FOREARM UP] FU 300

>PRINT [FOREARM DOWN] FD 300

>PRINT [WRIST CLOCKWISE] WC 200

>PRINT [WRIST ANTICLOCKWISE] WA 200

>PRINT [FORCEPS OPEN] FO 150

>PRINT [FORCEPS CLOSE] FC 150

>END

>^Z
```

NOTES ON SYNTAX NOTATION

Angle brackets, < >, are used in describing the syntax of the primitives. They indicate the possible inputs to complete the command or operation concerned.

The nature of the primitive, whether it is a command or an operation, is enclosed in parentheses next to the name of the primitive. Just in case you have forgotten the difference between a command and an operation, we have defined them again below:

COMMAND - A command with appropriate inputs tells Rogo to perform a certain function and then Rogo will give you an explicit response.

OPERATION - An operation with appropriate inputs tells Rogo to output a certain result without specifying what to do with the result.

5.1 ROTATING THE BASE

Rogo can rotate the base of the Robotarm in 2 directions, clockwise and anticlockwise, through a maximum of 270°.

BC (COMMAND)

Rotates the base of the Robotarm clockwise for a specified time period.

Syntax: BC <time period>

NOTE : <time period> can be expressed in a positive real number, a variable, or an Arithmetic Operation.

EXAMPLE:

? BC 100

It rotates the base of the Robotarm clockwise for 100 time units.

EXAMPLE:

? MAKE "TIMEUNIT 100

? BC :TIMEUNIT

MAKE is a primitive to assign a value to a variable. In the above example, TIMEUNIT is the name of the variable whose value is 100. :TIMEUNIT outputs the value of the variable and BC :TIMEUNIT rotates the base for 100 time units.

EXAMPLE:

? BC SUM 70 30

SUM 70 30 is an Arithmetic Operation that adds the two numbers following it and outputs the result. BC SUM 70 30 bears the same result as BC 100.

NOTE : The angle formed by the movement relative to the initial position is directly proportional to the length of time the Robotarm is moved; hence, the longer the time unit, the wider the angle.

BA (COMMAND)

Rotates the base of the Robotarm anticlockwise for a specified time period.

Syntax: BA <time period>

EXAMPLE:

? BA 75

It rotates the base anticlockwise for 75 time units.

EXAMPLE:

? MAKE "T 75

? BA :T

It has the same effect as BA 75.

EXAMPLE:

? BA DIFF 100 25

DIFF outputs the difference of the two numbers following it. BA DIFF 100 25 is the same as BA 75.

5.2 MOVING THE LOWERARM

Rogo can raise the lowerarm of the Robotarm and lowers it through a maximum of 90°.

LU (COMMAND)

Raises the lowerarm for a specified time period.

Syntax: LU <time period>

EXAMPLE:

? LU 50

It raises the lowerarm for 50 time units.

EXAMPLE:

? MAKE "TIME1 50

? LU :TIME1

EXAMPLE:

? LU PROD 10 5

PROD outputs the product of the two numbers following it. LU PROD 10 5 is the same as LU 50.

LD (COMMAND)

Lowers the lowerarm for a specified time period.

Syntax: LD <time period>

EXAMPLE:

? LD 25

It lowers the lowerarm for 25 time units.

? MAKE "TIME2 25

? LD :TIME2

EXAMPLE:

? LD QUOT 50 2

QUOT outputs the quotient of the two numbers following it. LD QUOT 50 2 is the same as LD 25.

5.3 MOVING THE FOREARM

Rogo can raise and lower the forearm of the Robotarm through a maximum of 85°.

FU (COMMAND)

Raises the forearm for a specified time period.

Syntax: FU <time period>

EXAMPLE:

```
? MAKE "VARIABLE 100

? FU 50 FU :VARIABLE FU SUM 30 90
```

It raises the forearm for a total of 270 time units.

FD (COMMAND)

Lowers the forearm for a specified time period.

Syntax: FD <time period>

EXAMPLE:

```
? FD 100

? FD PROD 10 10

? MAKE "A  50

? FD :A
```

5.4 ROTATING THE WRIST

Rogo can move the wrist of the Robotarm clockwise and anticlockwise without any limitation. The spotlight of the Robotarm is lit when the wrist moves.

WC (COMMAND)

Rotates the wrist clockwise for a specified time period.

Syntax: WC <time period>

EXAMPLE:

```
? WC 400

? MAKE "ANGLE 200

? WC :ANGLE WC DIFF 500 200
```

WA (COMMAND)

Rotates the wrist anticlockwise for a specified time period.

Syntax: WA <time period>

EXAMPLE:

```
? MAKE "ANGLE 300

? WA :ANGLE WA 200 WA QUOT 100 2
```

5.5 MOVING THE FORCEPS

Rogo can open and close the forceps of the Robotarm through a maximum of 108°.

FO (COMMAND)

Opens the forceps for a specified time period.

Syntax: FO <time period>

EXAMPLE:

```
? FO DIFF 250 50 FO 200

? MAKE "ANGLE 100

? FO :ANGLE
```

FC (COMMAND)

Closes the forceps for a specified time period.

Syntax: FC <time period>

EXAMPLE:

```
? MAKE "TIME 50

? FC 100 FC :TIME FC SUM 50 50
```

5.6 CHAPTER SUMMARY

PRIMITIVE	SYNTAX	DESCRIPTION
BC	BC <time period>	Rotates the base clockwise.
BA	BA <time period>	Rotates the base anticlockwise.
LU	LU <time period>	Raises the lowerarm.
LD	LD <time period>	Lowers the lowerarm.

PRIMITIVE	SYNTAX	DESCRIPTION
FU	FU <time period>	Raises the forearm.
FD	FD <time period>	Lowers the forearm.
WC	WC <time period>	Rotates the wrist clockwise.
WA	WA <time period>	Rotates the wrist anticlockwise.
FO	FO <time period>	Opens the forceps.
FC	FC <time period>	Closes the forceps.

CHAPTER 6
VARIABLES

MAKE and THING are two primitives that are frequently used in manipulating variables. MAKE names a variable and assigns a value to that variable. THING outputs the content of a variable. Let's begin with some examples:

```
? MAKE "WORD "UNICORN

? PRINT :WORD

UNICORN

? MAKE "LIST [UNICORNS LIVE IN LEGENDS]

? PRINT :LIST

UNICORNS LIVE IN LEGENDS

? MAKE "NUMBER 1

? PRINT THING "NUMBER

1

? MAKE "NUMBER SUM 2 4

? PRINT THING "NUMBER

6
```

6.1 MAKE (COMMAND)

Names a variable and assigns a value to that variable.

Syntax: MAKE <variable name> <value>

NOTE : 1. The name of a variable should begin with a letter and must not exceed 9 characters. If a variable name is longer than that, Rogo will flag an error message. Numbers and symbols are accepted.

2. The name of a variable is regarded as a word in Rogo. Therefore, it must be preceded by a double quotation mark, ".

3. If the same variable name is used twice, the second value will replace the first.

4. The value of a variable can be a word, a number, a list, an Arithmetic Operation, or an I/O Operation. A word must be preceded by a double quotation mark, ", whereas a list should be enclosed by a pair of square brackets, [ ].

The above notes are applicable to both MAKE and THING.

EXAMPLE:

```
? MAKE "SVI "ROGO

? PRINT :SVI

ROGO
```

In the above example, the variable name is SVI whose value is ROGO. Since the value is a word, it is preceded by a double quotation mark. The :SVI in the PRINT command asks Rogo to output the content of the variable SVI and the primitive PRINT prints the result.

EXAMPLE:

```
? MAKE "SVI 2000

? PRINT :SVI

2000
```

As the value of the variable in this case is a number, it needs not be preceded by a double quotation mark.

EXAMPLE:

```
? MAKE "GREETINGS [HOW ARE YOU?]

? PRINT :GREETINGS

HOW ARE YOU?
```

EXAMPLE:

```
? MAKE "RESULT SUM 2 3

? PRINT :RESULT

5
```

SUM is an Arithmetic Operation. Since Arithmetic Operations can take variables as inputs, the value of the variable in a MAKE command can also be in the following format:

EXAMPLE:

```
? MAKE "A 2

? MAKE "B 3

? MAKE "RESULT SUM :A :B

? PRINT :RESULT

5
```

The value of a variable can also be an I/O Operation that concerns true or false:

EXAMPLE:

```
? TO "LOG

> MAKE "LOGIC1 J1 1

> PRINT :LOGIC1

> END

> ΛZ
```

J1 is an I/O operation that reads the direction of the joystick in Joystick Port 1 (to connect the joystick, read 10.5). It outputs true when the joystick is in the direction it specifies. Otherwise, it outputs false. In the above example, if the joystick is being pushed forward while LOG is being executed, Rogo outputs and prints TRUE.

EXAMPLE:

```
? TO "READ

> MAKE "I/O READC

> PRINT :I/O

> END

> ΛZ
```

READC is an I/O operation. It reads and outputs the first word typed. Read 10.3 for details.

6.2 THING (OPERATION)

Outputs the value of the variable named.

Syntax: THING <variable name>

EXAMPLE:

```
? MAKE "SVI "ROGO

? PRINT THING "SVI

ROGO

? MAKE "SVI 2000

? PRINT THING "SVI

2000

? MAKE "GREETINGS [HOW ARE YOU?]

? PRINT THING "GREETINGS

HOW ARE YOU?

? MAKE "A 5

? MAKE "B 6

? MAKE "RESULT PROD :A :B

? PRINT THING "RESULT

30
```

At a glance the operation THING "SVI works exactly like :SVI. But THING can do much more. Consider the following:

EXAMPLE:

```
? MAKE "SVI "ROGO

? MAKE "ROGO 2000

? PRINT THING :SVI

2000
```

In the first MAKE command, SVI is the variable name and ROGO is the value. In the second MAKE command, the value of the previous variable becomes a variable itself, having a value of 2000. THING :SVI then outputs 2000 as the transformed value of SVI. If you use the colon command without THING, the outcome will be completely different.

```
? PRINT :SVI

ROGO

? PRINT :ROGO

2000
```

The colon command alone can only recognise the value assigned in one MAKE command.

6.3 DEFINING A PROCEDURE USING VARIABLES

Variables are very useful in defining procedures. Let's start with a simple procedure using one variable:

EXAMPLE:

```
? TO "MOVE5 :TIMEUNIT

>PRINT [ROTATE BASE CLOCKWISE ] BC :TIMEUNIT

>PRINT [RAISE LOWERARM ] LU :TIMEUNIT

>PRINT [RAISE FOREARM ] FU :TIMEUNIT

>PRINT [ROTATE WRIST CLOCKWISE ] WC :TIMEUNIT

>END

>^Z
```

In the above example, MOVE5 is the name of the procedure and TIMEUNIT is the variable name. The colon before TIMEUNIT tells Rogo MOVE5 takes a variable called TIMEUNIT. When you execute MOVE5, you have to enter the value of TIMEUNIT too.

EXAMPLE:

```
? MOVE5 100
```

It tells Rogo to execute MOVE5 which takes a variable having a value of 100. The base, lowerarm, forearm, and wrist will all move for 100 time units and on the screen you will read:

```
ROTATE BASE CLOCKWISE

RAISE LOWERARM

RAISE FOREARM

ROTATE WRIST CLOCKWISE
```

Try to execute MOVE5 without giving the value of the variable. You will certainly get an error message.

Now you have no trouble defining a procedure using one variable. Using two variables is more or less the same as using one:

EXAMPLE:

```
? TO "BOX :SIDE :WIDTH

>PRINT [THE SIDE OF THE BOX IS:] PRINT :SIDE

>PRINT [THE WIDTH OF THE BOX IS:] PRINT :WIDTH

>PRINT [THEN THE AREA OF THE BOX IS :]

>MAKE "RESULT PROD :SIDE :WIDTH

>PRINT :RESULT

>END

>^Z
```

When you execute BOX, don't forget to enter 2 values for SIDE and WIDTH.

EXAMPLE:

```
? BOX 4 5

THE SIDE OF THE BOX IS:

4

THE WIDTH OF THE BOX IS:

5

THEN THE AREA OF THE BOX IS:

20
```

6.4 CHAPTER SUMMARY

PRIMITIVE	SYNTAX	DESCRIPTION
MAKE	MAKE <variable> <value>	Names a variable and assigns a value to the variable.
THING	THING <variable>	Outputs the value of the variable named.

CHAPTER 7
ARITHMETIC OPERATIONS

In Rogo, addition, subtraction, multiplication, and division are performed through the primitives of SUM, DIFF, PROD, and QUOT respectively. It can also output the integer portion of or integer round off from a number using INT and ROUND. All Arithmetic Primitives are operations.

EXAMPLE:

```
? TO "EXERCISE

>MAKE "A SUM 30 30 PRINT :A BC :A

>MAKE "B DIFF 100 50 PRINT :B LU :B

>MAKE "C PROD 20 4 PRINT :C FU :C

>MAKE "D QUOT 200 2 PRINT :D WC :D

>MAKE "E INT PROD 15.6 4.8 PRINT :E FO :E

>MAKE "F INT QUOT 100 3 PRINT :F FC :F

>MAKE "G ROUND DIFF 200 0.1 PRINT :G FD :G

>MAKE "H ROUND SUM 50.3 30.1 PRINT :H LD :H

>END

>^Z
```

7.1 SUM (OPERATION)

Outputs the sum of the two numbers that follow.

Syntax: SUM <real number> <real number>

EXAMPLE:

```
? SUM 2 2
```

You will not get a 4 displayed on the screen. This is because SUM is an operation. You have to specify what to do with the output; otherwise Rogo simply outputs and remembers the addition without working further on it. If you want to read the output on the screen, tell Rogo to print it:

EXAMPLE:

```
? PRINT SUM 2 2

4
```

SUM can add only 2 numbers at a time. If you want to add 3 numbers, see what happens:

EXAMPLE:

```
? PRINT SUM 1 2 3

3

I DON'T KNOW HOW TO 3
```

The right way to add 3 numbers is:

EXAMPLE:

```
? PRINT SUM SUM 1 2 3

6
```

SUM SUM 1 2 3 is equivalent to SUM (SUM 1 2) 3. SUM 1 2 outputs 3 and SUM 3 3 outputs 6.

EXAMPLE:

```
? PRINT SUM SUM SUM 1 2 3 4

10
```

SUM accepts variables as its inputs:

EXAMPLE:

```
? MAKE "A 5

? PRINT SUM :A 10

15

? MAKE "B 3

? PRINT SUM :A :B

8
```

NOTE : The above rules also apply to DIFF, PROD, QUOT, INT, and ROUND.

7.2 DIFF (OPERATION)

Outputs the difference of the two numbers that follow.

Syntax: DIFF <real number> <real number>

EXAMPLE:

```
? PRINT DIFF 10 5

5

? PRINT DIFF DIFF 100 10 10

80
```

The arithmetic notation of DIFF DIFF 100 10 10 is:

(100-10)-10=80.

```
? MAKE "ANS DIFF DIFF DIFF 100 10 20 30

? PRINT SUM 1 :ANS

41
```

The set of commands in MAKE is equivalent to [(100-10)-20]-30=40.

7.3 PROD (OPERATION)

Outputs the product of the two numbers that follow.

Syntax: PROD <real number> <real number>

EXAMPLE:

```
? PRINT PROD 3 5

15

? PRINT PROD PROD 3 5 7

105

? MAKE "ANSWER PROD SUM DIFF 6 4 8 10

? MAKE "RESULT :ANSWER PRINT PROD :RESULT :RESULT

10000
```

The arithmetic equivalent for the first MAKE command would be [(6-4)+8]x10=100.

7.4 QUOT (OPERATION)

Outputs the quotient of the two numbers that follow.

Syntax: QUOT <real number> <real number>

EXAMPLE:

```
? PRINT QUOT 50 2

25

? PRINT QUOT QUOT 50 2 5

5

? MAKE "A 0 PRINT QUOT 100 :A

0

? PRINT DIFF SUM PROD QUOT 100 4 2 5 30

25
```

By now, you should recognise the above as the ROGO equivalent of [(100÷4)x2+5]-30=25

7.5 INT (OPERATION)

Outputs the integer of a number.

Syntax: INT <real number>

EXAMPLE:

```
? PRINT INT 3.01

3

? PRINT INT 3.99

3

? PRINT INT -3.01

-3

? PRINT INT -3.99

-3
```

INT also takes a variable or an Arithmetic Operation as inputs:

EXAMPLE:

```
? PRINT INT SUM PROD 4.3 2.1 6

15

? MAKE "A -1.23

? PRINT INT :A

-1
```

7.6 ROUND (OPERATION)

Outputs the integer round off from a number.

Syntax: ROUND <real number>

EXAMPLE:

```
? PRINT ROUND 5.9

6

? PRINT ROUND 5.1

5

? PRINT ROUND -5.9

-6

? PRINT -5.1

-5
```

Similar to INT, ROUND also takes a variable or an Arithmetic Operation as its input.

EXAMPLE:

```
? PRINT ROUND QUOT PROD 45.1 3.2 8

18

? MAKE "B 0.99 PRINT ROUND :B

1
```

7.7 CHAPTER SUMMARY

PRIMITIVE	SYNTAX	DESCRIPTION
SUM	SUM <real number> <real number>	Adds two numbers.
DIFF	DIFF <real number> <real number>	Subtracts the second number from the first one.
PROD	PROD <real number> <real number>	Multiplies two numbers.
QUOT	QUOT <real number> <real number>	Divides the first number by the second number.
INT	INT <real number>	Outputs the integer of the number.
ROUND	ROUND <real number>	Rounds off the number.

CHAPTER 8
LOGICAL OPERATIONS

Ten Logical Operations are designated to look after manipulations that concern only TRUE or FALSE. Rogo outputs TRUE when the given condition is satisfied and FALSE when it is not. The Logical Operations we are going to introduce are <, >, =, <=, >=, <>, AND, NOT, OR, and XOR.

If you have 2 joysticks connected to your MSX computer, you can try the following procedure. If you are not sure how to connect the joysticks to your computer, read section 10.5 of this manual or the relevant section of the User's Manual of your computer.

EXAMPLE:

```
? TO "START

>PRINT [PRESS ONE OR BOTH FIREBUTTONS.]

>PRINT [I CAN TELL WHICH FIREBUTTON HAS BEEN PRESSED.]

>WAIT 60 PRINT [ ]

>REPEAT 30 [GAME ASK]

>END

>ΔZ


? TO "GAME

>PRINT [GOOD.....]

>REPEAT 20 [MAKE "JOY1 J1 9 MAKE "JOY2 J2 9]

>IF AND :JOY1  :JOY2 [PRINT [YOU'VE PRESSED BOTH FIREBUTTONS.]]

>IF XOR :JOY1  :JOY2 [PRINT [YOU'VE PRESSED ONE OF THE
 FIREBUTTONS.  LET ME SEE...]]

>IF :JOY1 [PRINT [IT IS JOYSTICK 1 !]]
```

```
>IF :JOY2 [PRINT [IT IS JOYSTICK 2 !]]

>IF NOT OR :JOY1  :JOY2 PRINT [YOU HAVEN'T PRESSED ANY
 FIREBUTTONS.]]

>END

>ΔZ

? TO "ASK

>PRINT [TRY AGAIN? "YES" OR "NO."]

>MAKE "ANS READC

>PRINT [ ]

>IF :ANS = "NO [NO]

>IF :ANS < > "YES [PARDON]

>END

>ΛZ


? TO "NO

>PRINT "BYE

>STOP

>END

>ΛZ


? TO "PARDON

>PRINT [PARDON! "YES" OR "NO" ?]
```

```
>MAKE "ANS READC PRINT [ ]

>IF :ANS < > "YES [PARDON]

>END

>^Z
```

J1, J2, and READC are I/O Commands whereas IF, REPEAT, STOP, and WAIT are Control Flow Commands. J1 and J2 recognise the direction of the joysticks, IF determines whether an instructionlist should be executed, REPEAT repeats the instructionlist for a specified number of times, STOP halts the procedure, and WAIT makes Rogo wait for a specified time period.

Don't worry if you find the above description hard to understand. Details are forthcoming. We include these commands here because Logical Operations go hand in hand with other commands, especially Control Flow Commands.

The above example is a procedure that consists of 5 simple modules, START, GAME, ASK, NO, and PARDON. START initiates the procedure. You have 60 time units to press the firebuttons. Then GAME checks which button you have pressed. After that, ASK sees whether you want to continue. If you do, you will be brought to GAME again. If you don't, NO will stop the procedure.

PARDON will be executed if you do not give a concrete answer in ASK. Note that PARDON calls itself within the same procedure. It is a RECURSIVE command. It will loop back to the beginning of PARDON if you do not indicate "YES" or "NO". More about recursive commands in Appendix D.

8.1 < (OPERATION)

Outputs true when <condition 1> is less than <condition 2>; otherwise false.

Syntax: <condition 1> < <condition 2>

NOTE : <condition 1> and <condition 2> can be numbers, Arithmetic Operations, letters, or variables. The same rules apply to <, >, =, <=, <>, and >=.

EXAMPLE:

```
? IF 2 < 3 [PRINT [2 IS LESS THAN 3]]

2 IS LESS THAN 3
```

IF is a Control Flow Command that executes an instructionlist when a certain condition is met. In the above example, 2 < 3 is the condition for the IF command and [PRINT [2 IS LESS THAN 3]] is the instructionlist. Since 2 < 3 is true, the instructionlist is executed. We will discuss the details of Control Flow Commands in the next chapter. Let's concentrate on Logical Operations for the time being.

In 2 < 3, 2 is <condition 1> and 3 is <condition 2>. As <condition 1> is less than <condition 2>, Rogo outputs true.

EXAMPLE:

```
? IF 10 < 1 [PRINT [WHO SAYS 10 IS GREATER THAN 1?]]
```

If you enter the above example, nothing will be displayed on the screen because 10 < 1 is false. In 10 < 1, 10 is <condition 1> and 1 is <condition 2>. As <condition 1> is not less than <condition 2>, Rogo outputs false.

EXAMPLE:

```
? IF SUM 20 5 < QUOT 100 2 [PRINT "YES]

YES

? IF 0 < PROD 3 4 [PRINT [0 IS LESS THAN 12]]

0 IS LESS THAN 12
```

EXAMPLE:

```
? IF "A < "B [PRINT [THE ASCII CODE OF A IS LESS THAN THAT OF
B.]]

THE ASCII CODE OF A IS LESS THAN THAT OF B.
```

Letters are compared according to their respective ASCII codes. ASCII stands for American Standard Codes for Information Interchange (read your User's Manual for details). The ASCII of A is 65 and that of B is 66.

NOTE : Letters must be compared with letters only. Make sure you put a quotation mark before the letter which is regarded as a word in Rogo.

EXAMPLE:

```
? IF "ABC < "D [PRINT [ABC IS LESS THAN D]]

ABC IS LESS THAN D
```

When the conditions comprise a number of characters, they are compared character to character, from left to right. Since the first character and only character in <condition 2> is D (ASCII code = 68) which is greater than A (ASCII code = 65), ABC is less than D.

EXAMPLE:

```
? IF "ABC < "ABCD [PRINT [ABC IS LESS THAN ABCD]]

ABC IS LESS THAN ABCD
```

<condition 1> and <condition 2> are equal up to the third character, so Rogo goes on comparing the fourth character. As there is not a fourth character in <condition 1>, it is regarded as lesser.

The conditions in Logical Operations also take variables:

EXAMPLE:

```
? MAKE "A 10 MAKE "B 12

? IF :A < :B [PRINT [VARIABLE A IS LESS THAN VARIABLE B]]

VARIABLE A IS LESS THAN VARIABLE B

? IF SUM :A :B < 15 [PRINT [HOW CAN IT BE?]]
```

8.2 > (OPERATION)

Outputs true when <condition 1> is greater than <condition 2>; otherwise false.

Syntax: <condition 1> > <condition 2>

EXAMPLE:

```
? IF 5 > 3 [PRINT [5 IS GREATER THAN 3]]

5 IS GREATER THAN 3

? IF DIFF 10 3 > SUM 10 7 [PRINT "NO!!]
```

```
? IF "B > "A [PRINT [B IS GREATER THAN A]]

B IS GREATER THAN A

? IF "ABCD > "ABC [PRINT "TRUE]

TRUE

? MAKE "A 100 MAKE "B 10

? IF QUOT :A :B > PROD :A :B [PRINT [ARE YOU SURE?]]
```

8.3 = (OPERATION)

Outputs true when <condition 1> is equal to <condition 2>; otherwise false.

Syntax: <condition 1> = <condition 2>

EXAMPLE:

```
? IF 1 = 1 [PRINT [THAT'S TRUE]]

THAT'S TRUE

? IF PROD SUM 2 8 10 = 100 [PRINT "YES]

YES

? IF "A = "A [PRINT [A IS EQUAL TO A]]

A IS EQUAL TO A

? MAKE "A 10

? IF :A = SUM 5 :A [PRINT [DON'T KID ME!]]
```

8.4 <= (OPERATION)

Outputs true when <condition 1> is less than or equal to <condition 2>; otherwise false.

Syntax: <condition 1> <= <condition 2>

EXAMPLE:

```
? IF 3 <= 5 [PRINT [IT'S TRUE BECAUSE 3 IS LESS THAN 5]]

IT'S TRUE BECAUSE 3 IS LESS THAN 5

? IF 5 <= 5 [PRINT [ROGO OUTPUTS TRUE AS 5 IS EQUAL TO 5]]

ROGO OUTPUTS TRUE AS 5 IS EQUAL TO 5

? IF SUM 2 2 <= 10 [PRINT "TRUE]

TRUE

? IF "ABC <= "A [PRINT [THE WORLD IS FLAT]]

? MAKE "A 1 MAKE "B 2

? IF SUM :A :B <= 3 [PRINT [ROGO OUTPUTS TRUE]]

ROGO OUTPUTS TRUE
```

8.5 >= (OPERATION)

Outputs true when <condition 1> is greater than or equal to <condition 2>; otherwise false.

Syntax: <condition 1> >= <condition 2>

EXAMPLE:

```
? IF PROD 10 10 >= SUM 90 10 [PRINT "TRUE]

TRUE

? IF "ABC >= "ABCDE [PRINT "FALSE]

? MAKE "A 10 MAKE "B 20

? IF :B >= :A [PRINT [ROGO OUTPUTS TRUE AS 20 > 10]]

ROGO OUTPUTS TRUE AS 20 > 10
```

8.6 < > (OPERATION)

Outputs true when <condition 1> is not equal to <condition 2>; otherwise false.

Syntax: <condition 1> <> <condition 2>

EXAMPLE:

```
? IF SUM 2 2 <> DIFF 2 2 [PRINT [SUM 2 2 IS NOT EQUAL TO DIFF
2 2]]

SUM 2 2 IS NOT EQUAL TO DIFF 2 2

? MAKE "A 45 MAKE "B 5

? IF QUOT :A :B <> 10 [PRINT [QUOT 45 5 SHOULD EQUAL 9]]
QUOT 45 5 SHOULD EQUAL 9
```

8.7 AND (OPERATION)

Outputs true when both <condition 1> and <condition 2> is true; otherwise false.

Syntax: AND <condition 1> <condition 2>

NOTE : The conditions in AND, NOT, OR, and XOR must be operations concerning true or false.

EXAMPLE:

```
? IF AND SUM 2 2 = 4 10 < 100 [PRINT [BOTH CONDITIONS ARE TRUE]]

BOTH CONDITIONS ARE TRUE
```

In the above example, SUM 2 2 = 4 is <condition 1> and 10 < 100 is <condition 2>. Since both conditions are true, Rogo outputs true and executes the instructionlist of IF.

```
? MAKE "A 1 MAKE "B 2

? IF AND PROD :A :B = 2  5 >= 8 [PRINT "TRUE]
```

Rogo will not execute the instructionlist since only <condition 1> is true.

8.8 NOT (OPERATION)

Outputs true when a certain condition is not satisfied; otherwise false.

Syntax: NOT <condition>

EXAMPLE:

```
? IF NOT SUM 2 2 = 4 [PRINT [SUM 2 2 IS NOT EQUAL TO 4]]
```

SUM 2 2 = 4 is the condition. Since SUM 2 2 = 4 is true, NOT SUM 2 2 = 4 is not satisfied. Therefore, Rogo will not execute the instructionlist.

EXAMPLE:

```
? IF NOT PROD 3 4 <= 0 [PRINT [12 IS GREATER THAN 0]]

12 IS GREATER THAN 0
```

As the condition in the NOT operation is not true, Rogo outputs true and consequently, the instructionlist is executed.

8.9 OR (OPERATION)

Outputs true when either or both <condition 1> and <condition 2> are true; otherwise false.

Syntax: OR <condition 1> <condition 2>

EXAMPLE:

```
? IF OR 2 > 1 PROD 5 5 = 20 [PRINT [CONDITION 1 IS TRUE]]

CONDITION 1 IS TRUE
```

In the OR operation, 2 > 1 is <condition 1> and PROD 5 5 = 20 is <condition 2>. As one of the conditions is satisfied, Rogo outputs true and executes the instructionlist.

EXAMPLE:

```
? MAKE "X 3  MAKE "Y 5

? IF OR PROD :X :Y = 15 SUM :X :Y > 0 [PRINT [BOTH CONDITIONS
ARE TRUE]]

BOTH CONDITIONS ARE TRUE
```

```
? IF OR QUOT :X :Y = 0 DIFF :X :Y >= 0 [PRINT [NEITHER CONDITION
IS TRUE]]
```

8.10 XOR

Outputs true when either <condition 1> or <condition 2> is true; otherwise false.

Syntax: XOR <condition 1> <condition 2>

EXAMPLE:

```
? IF XOR SUM 2 8 = 10 DIFF 5 3 < 0 [PRINT [ONE CONDITION IS
TRUE]]

ONE CONDITION IS TRUE

? MAKE "X 12  MAKE "Y 3

? IF XOR PROD :X :Y > QUOT 12 3  SUM :X :Y > DIFF :X :Y [PRINT
[BOTH CONDITIONS ARE TRUE]]
```

Since both conditions are true, Rogo outputs false and the instructionlist is not executed.

8.11 CHAPTER SUMMARY

PRIMITIVE	SYNTAX	DESCRIPTION
<	<condition 1> < <condition 2>	Outputs true when <condition 1> is less than <condition 2>; otherwise false.

PRIMITIVE	SYNTAX	DESCRIPTION
>	<condition 1> > <condition 2>	Outputs true when <condition 1> is greater than <condition 2>; otherwise false.
=	<condition 1> = <condition 2>	Outputs true when <condition 1> is equal to <condition 2>; otherwise false.
<=	<condition 1> <= <condition 2>	Outputs true when <condition 1> is less than or equal to <condition 2>; otherwise false.
>=	<condition 1> >= <condition 2>	Outputs true when <condition 1> is greater than or equal to <condition 2>; otherwise false.
< >	<condition 1> < > <condition 2>	Outputs true when <condition 1> is not equal to <condition 2>; otherwise false.
AND	AND <condition 1> <condition 2>	Outputs true when both <condition 1> and <condition 2> are true; otherwise false.
NOT	NOT <condition>	Outputs true when the condition is false; otherwise false.

PRIMITIVE	SYNTAX	DESCRIPTION
OR	OR <condition 1> <condition 2>	Outputs true when either or both <condition 1> and <condition 2> are true; otherwise false.
XOR	XOR <condition 1> <condition 2>	Outputs true when either <condition 1> or <condition 2> is true; otherwise false.

CHAPTER 9
CONTROL FLOW COMMANDS

IF, TEST, IFTRUE, IFFALSE, REPEAT, STOP, WAIT, OP, and RUN are the primitives that control the flow of instructions in Rogo. They help Rogo decide if a certain instructionlist should be executed when a certain condition is met.

EXAMPLE:

```
? TO "WARMUP

>PRINT [CHOOSE A DIRECTION TO MOVE:  BC, BA, LU, LD, FU, FD, WC,
 WA, FO, OR FC.]

>MAKE "ANS READL

>IF :ANS = [BC] [BC 200] IF :ANS = [BA] [BA 200]

>IF :ANS = [LU] [LU 200] IF :ANS = [LD] [LD 200]

>IF :ANS = [FU] [FU 200] IF :ANS = [FD] [FD 200]

>IF :ANS = [WC] [WC 200] IF :ANS = [WA] [WA 200]

>IF :ANS = [FO] [FO 200] IF :ANS = [FC] [FC 200]

>PRINT [TRY AGAIN? "YES" OR "NO".]

>MAKE "ANSWER READL

>TEST :ANS = [YES]

>IFFALSE [REPEAT 3 [PRINT [TOO BAD!!]] STOP]

>IFTRUE [WARMUP]

>END

>^Z
```

Note that the instructionlist of the IFTRUE command of the above procedure recurs the execution of the procedure. When a procedure calls itself, it is RECURSIVE. We will discuss recursive commands in detail in Appendix D.

9.1 IF (COMMAND)

Executes the instructionlist if the <condition> is true.

Syntax: IF <condition> <instructionlist>

NOTE : The <condition> in IF can be a number, a word, a list, a variable, a Logical Operation, or an I/O Command involving J1 or J2.

EXAMPLE:

```
? MAKE "JOYSTICK1 J1 1

? IF :JOYSTICK1 [PRINT [JOYSTICK 1 IS PUSHED FORWARD]]

JOYSTICK 1 IS PUSHED FORWARD
```

We suppose you have connected a joystick to Joystick Port 1 of your machine. If you haven't, try to connect it now (read 10.5 for details). If you push your joystick forward as you enter the first line, J1 1 outputs true which becomes the value of JOYSTICK1. As :JOYSTICK1 is true, the instructionlist of IF is executed. The above example is the same as the one below:

EXAMPLE:

```
? IF J1 1 [PRINT [JOYSTICK 1 IS PUSHED FORWARD]]
```

EXAMPLE:

```
? MAKE "A 4 MAKE "B 5

? IF PROD :A :B > SUM :A :B [PRINT DIFF :B :A]

1
```

In this example, PROD :A :B > SUM :A :B is the condition of IF, whereas [PRINT DIFF :B :A] is the instructionlist. Since the condition outputs TRUE, the instructionlist is executed.

EXAMPLE:

```
? TO "WAVE

>LD 200 FD 200 LU 200 FU 200 WC 400 FO 150

>END

>^Z

? TO "ASK

>PRINT [WHAT IS THE PRODUCT OF 12 AND 5?]

>MAKE "ANS READL

>IF :ANS = [60] [WAVE]

>END

>^Z
```

READL is an I/O command which reads the keyboard and outputs what you have typed (read 10.4 for details). Whatever is read by READL is regarded as a list; this explains why 60 (the product of 12 and 5) must be enclosed in square brackets though it is a number by nature. Notice that the instructionlist of IF in the above procedure is a user-defined command defined earlier.

When you execute ASK, Rogo poses the question: "WHAT IS THE PRODUCT OF 12 AND 5?" If you get it right, Rogo will execute the instructionlist and moves the Robotarm as specified in WAVE.

9.2 TEST (OPERATION)

Remembers if a specified condition is true or false for future use in IFTRUE and IFFALSE commands.

Syntax: TEST <condition>

NOTE : The <condition> can be a Logical Operation or an I/O Operation concerning true or false (e.g. J1).

Since TEST is closely related to IFTRUE and IFFALSE, we include all three in the following examples to give you a clearer picture of how they work.

EXAMPLE:

```
? TEST SUM 2 2 = QUOT 20 5

? IFTRUE [PRINT [SUM 2 2 IS EQUAL TO QUOT 20 5]]

SUM 2 2 IS EQUAL TO QUOT 20 5
```

IFTRUE is a Control Flow Command which executes an instructionlist if the output in the latest TEST operation is true (read 9.3 for details). In the above example, the condition of TEST is SUM 2 2 = QUOT 20 5 which is true. Rogo outputs true. The IFTRUE command that follows checks whether the output of the TEST operation is true. It is, so the instructionlist is then executed. Enter the line below and see what happens:

```
? IFFALSE [PRINT [SUM 2 2 IS NOT EQUAL TO QUOT 20 5]]
```

IFFALSE is another Control Flow Command that executes an instructionlist if the output in the latest TEST operation is false (read 9.4 for details). Since the output of TEST is true, Rogo outputs false in the IFFALSE command and the instructionlist is not executed.

You can edit the ASK procedure as follows:

```
? EDIT "ASK

>PRINT [WHAT IS THE PRODUCT OF 12 AND 5?]

>MAKE "ANS READL

>TEST :ANS = [60]

>IFTRUE [WAVE] IFFALSE [PRINT "SORRY!]

>END

>^Z
```

Before you execute ASK, make sure you have defined WAVE. The TEST operation in ASK checks if the value of ANS equals 60. If it does, the instructionlist of IFTRUE is executed; otherwise the instructionlist of IFFALSE is executed instead.

9.3 IFTRUE (COMMAND)

Executes the instructionlist if the output of the latest TEST operation is true.

Syntax: IFTRUE <instructionlist>

Similar to IF, the instructionlist of IFTRUE can be a user-defined command or any Rogo command or operation. The same rule applies to IFFALSE.

EXAMPLE:

```
? TEST 3 > 2

? IFTRUE [PRINT "TRUE] IFFALSE [PRINT "FALSE]

TRUE
```

```
? IFTRUE [LU 100 FU 300 WC 200] IFFALSE [PRINT "FALSE]
```

Rogo prints TRUE on screen as soon as you enter the first IFTRUE and IFFALSE commands. The Robotarm moves as specified after the second IFTRUE and IFFALSE commands have been entered. Though it is the second IFTRUE used in the above example, Rogo still reads the output in TEST 3 > 2. For IFTRUE and IFFALSE commands, Rogo always refers to the latest TEST operation.

EXAMPLE:

```
? MAKE "A 2 MAKE "B 4

? TEST SUM :A :B = 6

? IFTRUE [FD 150 WC 300 FC 50]

? TEST PROD :A :B > 0

? IFTRUE [WA 200 LD 100 BC 200 BA 200]
```

9.4 IFFALSE (COMMAND)

Executes the instructionlist if the output of the latest TEST is false.

Syntax: IFFALSE <instructionlist>

We have tried quite a number of examples in the last two sections. If you still have a joystick connected to the Joystick Port 1 of your machine, you can try the following:

EXAMPLE:

```
? TO "JOY
> TEST J1 1 IFTRUE [FU 100] IFFALSE [WC 100]
> TEST J1 3 IFTRUE [BC 100] IFFALSE [WC 100]
> TEST J1 5 IFTRUE [FD 200] IFFALSE [WC 100]
> TEST J1 7 IFTRUE [BA 200] IFFALSE [WC 100]
> ^Z
? REPEAT 10 [JOY]
```

9.5 REPEAT (COMMAND)

Repeats the instructionlist for a specified number of times.

Syntax: REPEAT <number of times> <instructionlist>

NOTE : You must enter a positive whole number to specify the number of times to repeat the instructionlist.

EXAMPLE:

```
? REPEAT 2 [PRINT [ROGO IS INTERESTING AND VERSATILE]]
ROGO IS INTERESTING AND VERSATILE
ROGO IS INTERESTING AND VERSATILE
```

The above example repeats the instructionlist twice.

EXAMPLE:

```
? REPEAT 10 [LU 300 LD 300 FU 300 FD 300]
```

The Robotarm will not stop until the instructionlist has been executed 10 times. The example below does the same job:

EXAMPLE:

```
? TO "PLAY
>LU 300 LD 300 FU 300 FD 300
>END
>^Z
? REPEAT 10 [PLAY]
```

EXAMPLE:

```
? MAKE "COUNT 1
? REPEAT 5 [MAKE "COUNT SUM :COUNT :COUNT PRINT :COUNT LD 200 FD
200 FU 200 LU 200 WC 100]
2
4
8
16
32
```

In the MAKE command, the value of COUNT is 1. When it comes to the instructionlist of REPEAT, the value of COUNT becomes the summation of itself; that means 2. 2 is then printed and Rogo repeats the instructionlist again. The value of COUNT doubles, the result is printed, and Rogo repeats the instructionlist and so on.

9.6 STOP (COMMAND)

Stops a running procedure and returns control to the top level.

Syntax: STOP

STOP is very helpful if you want to stop the execution of a procedure when a specified condition occurs:

EXAMPLE:

```
? MAKE "BACK 5

? REPEAT 10 [MAKE "BACK DIFF :BACK 1 PRINT :BACK LD 300 FU  300
  FD 300 LU 300 WC 200 IF :BACK = 0 [STOP]]

4

3

2

1

0
```

As you may notice, the instructionlist of REPEAT is executed 5 times instead of 10. When the value of BACK becomes 0, Rogo executes the instructionlist of IF and thus stops executing the instructionlist of REPEAT.

EXAMPLE:

```
? TO "PLAY

>PRINT [WANT ME TO MOVE? "YES" OR "NO".]

>MAKE "ANS READL

>IF :ANS = [NO] [PRINT [TOO BAD!!] STOP]

>REPEAT 10 [PRINT "GOOD LU 200 FD 200 FU 300 LD 200 WC 100]

>END

>^Z
```

The STOP in the IF command of the above procedure stops the procedure when the list read is NO. The rest of the procedure will not be executed.

9.7 WAIT (COMMAND)

Waits for a specified time period .

Syntax: WAIT <time period>

EXAMPLE:

```
? TO "SAMPLE1

>WA 20

>WA 20

>WA 20

>END

>^Z
```

```
? TO "SAMPLE2

>WA 20

>WAIT 120

>WA 20

>WAIT 120

>WA 20

>END

>^Z
```

In SAMPLE1, the wrist of the Robotarm will rotate anticlockwise continuously for 60 time units. In SAMPLE2, the wrist will rotate for 20 time units, wait awhile, rotate for another 20 time units, wait again and then rotate for 20 time units more.

WAIT is useful in delaying the flow of a procedure. It gives you some time to read or think:

EXAMPLE:

```
? TO "FAKE

>PRINT [DON'T COME NEAR!! YOU HAVE 5 SECONDS TO ESCAPE.]

>WAIT PROD 5 60

>WA 400 FO 200 PRINT [WOW!! GOTCHA!]

>END

>^Z
```

Note that apart from positive integers, WAIT also accepts Arithmetic Operations as inputs.

9.8 OP (COMMAND)

Makes a specified Rogo object an output of a procedure.

Syntax: OP <object>

NOTE : Though OP is a command, the procedure in which it is defined is an operation and OP is applicable ONLY in a procedure.

EXAMPLE:

```
? TO "ELEANOR

>OP "HELEN

>END

>^Z

? PRINT ELEANOR

HELEN
```

OP is a command that helps you to define an operation. In the above procedure, ELEANOR is defined to hold HELEN as an output. So, when you print ELEANOR, Rogo prints the output instead. OP also accepts a list:

EXAMPLE:

```
? TO "LINK :NAME

>PRINT [HOW ARE YOU?] OP :NAME

>END

>^Z
```

```
? PRINT LINK "SAM

HOW ARE YOU?

SAM

? TO ABS :N

>TEST :N < 0

>IFTRUE [OP DIFF 0 :N] IFFALSE [OP :N]

>END

>^Z

? PRINT ABS -4

4

? PRINT ABS DIFF 4 5

1
```

In the above example, DIFF 4 5 is -1. Rogo then prints the absolute value of -1 which equals 1. Note that OP will exit the procedure automatically once it is executed.

NOTE : OP is not supported in Version 1.0 of the Rogo Interface.

9.9 RUN (COMMAND/OPERATION)

Runs an instructionlist.

Syntax: RUN <instructionlist>

NOTE : <instructionlist> can be a user-defined command, a primitive, or an operation. If the specified <instructionlist> is an operation, RUN will output whatever the <instructionlist> outputs.

EXAMPLE:

```
TO "CALCULATE :INPUT

>PRINT RUN :INPUT

>END

>^Z

? CALCULATE [SUM 2 3]

5

? CALCULATE [PROD 22.1 5]

110.5
```

In the CALCULATE procedure, RUN functions as an operation. It outputs the result of the calculation which is then printed by the PRINT command.

What makes RUN indispensable is that it takes a list as an instruction. Try to edit CALCULATE and omit RUN in the PRINT command. Then execute CALCULATE again and notice the difference it makes:

```
? CALCULATE [SUM 2 3]

SUM 2 3
```

EXAMPLE:

```
? TO EXAM :CONDITION :INSTRUCTL
>TEST RUN :CONDITION IFFALSE [STOP]
>RUN :INSTRUCTL
>EXAM :CONDITION :INSTRUCTL
>END
>^Z
? MAKE "NUM 1
? EXAM [NUM < 5] [PRINT :NUM MAKE "NUM SUM :NUM 1]
1
2
3
4
```

EXAM is a procedure that takes 2 inputs, one is a condition, the other is an instructionlist. The RUN in TEST is an operation, whereas the one after the IFFALSE command is a command since it will execute the instructionlist if the condition is true.

To execute EXAM, you have to specify the condition and instructionlist first. In the above example, the condition is [:NUM < 5] and the instructionlist is [PRINT :NUM MAKE "NUM SUM :NUM 1]. The number to begin is specified in the MAKE command.

Suppose we begin with 1 which is less than 5. RUN outputs TRUE and so does TEST. The second RUN then executes the instructionlist which prints the number, increases it by 1, prints the result and so on. Note that EXAM is recursive. It loops back to the beginning of the procedure as long as the number is less than 5. The IFFALSE command is added as a control. When the number equals 5, the procedure stops.

9.10 CHAPTER SUMMARY

PRIMITIVE	SYNTAX	DESCRIPTION
IF	IF <condition> <instructionlist>	Executes the instructionlist if the condition is true.
TEST	TEST <condition>	Remembers if the condition is true or false for future use by IFTRUE and IFFALSE commands.
IFTRUE	IFTRUE <instructionlist>	Executes the instructionlist if the output of the latest TEST operation is true.
IFFALSE	IFFALSE <instructionlist>	Executes the instructionlist if the output of the latest TEST operation is false.
REPEAT	REPEAT <number of times> <instructionlist>	Repeats the instructionlist for a specified number of times.
STOP	STOP	Stops execution and returns control to the top level.
WAIT	WAIT <time period>	Waits for a specified time period.

PRIMITIVE	SYNTAX	DESCRIPTION
OP	OP <object>	Makes a specified Rogo object an output of a procedure.
RUN	RUN <instructionlist>	Runs an instructionlist.

CHAPTER 10
I/O COMMANDS AND OPERATIONS

The primitives that display inputs on screen, locate the print position, read the keyboard, and communicate with joysticks and cassette recorders are known as I/O primitives. Rogo has 8 of these: PRINT, LOCATE, READC, READL, J1, J2, SAVE "CAS, and LOAD "CAS.

EXAMPLE:

```
TO "CHECK :LIMIT

>IF :NUM2 = :LIMIT [PRINT [TOTAL CORRECT ANSWERS:] PRINT  :SCORE
 PRINT [THAT'S ALL FOLKS! SEE YOU LATER.] STOP]

>END

>^Z


TO "RIGHT

>REPEAT 3 [PRINT "RIGHT FU 50 FD 50]

>MAKE "SCORE SUM :SCORE 1

>END

>^Z


TO "WRONG

>REPEAT 3 [PRINT "WRONG BC 50 BA 50]

>END

>^Z
```

```
? TO "TRY :NAME

>PRINT [TRY AGAIN? "YES" OR "NO".]

>MAKE "RESPONSE READL

>TEST :RESPONSE = [YES]

>IFFALSE [PRINT [TOTAL CORRECT ANSWERS:] PRINT :SCORE PRINT [BYE
 FOR NOW.] STOP]

>IFTRUE :NAME

>END

>^Z

TO "TT :NUM1

>CLS PRINT [WHAT IS THE PRODUCT OF ] LOCATE 24 1 PRINT :NUM1

>LOCATE 28 1 PRINT [X] LOCATE 30 1 PRINT :NUM2

>MAKE "ANS READC

>IF  :ANS  = PROD :NUM1 :NUM2 [RIGHT CHECK 10  MAKE  "NUM2   SUM
 :NUM2 1 TRY [TT :NUM1]]

>WRONG CHECK 10

>MAKE "NUM2 SUM :NUM2 1

>TRY [TT :NUM1]

>END

>^Z
```

```
? TO "START

>MAKE "NUM2 1

>MAKE "SCORE 0

>PRINT  [WHAT NUMBER SHALL WE START WITH?] PRINT [(YOU CAN ENTER
 NOT MORE THAN 3 DIGITS.)]

>MAKE "NUM1 READC

>TT :NUM1

>END

>^Z
```

The above example is a typical Rogo procedure which is made up of tiny modules. Though each of them can stand on its own, they are linked in such a way that they can be started by typing only one command. The 6 modules that constitute the procedure are START, TT, TRY, WRONG, RIGHT, and CHECK. Together they form a quiz that checks how good you are at the times table.

START is the command that starts all the modules. It sets NUM2 to 1, total score to 0, reads a number from you (NUM1) and sends it to TT. TT poses a question asking for the product of NUM1 and NUM2. If you get it right, TT sends you to RIGHT which moves the forearm, and registers your score. If you were wrong, TT executes WRONG which moves the base.

TT then calls CHECK to check whether NUM2 has reached the limit of 10. If it has, CHECK will display the total score and stop the procedure. If it hasn't, TT increases NUM2 by 1 and calls TRY to see if you would like to go on. If you do, TT will pose another question and if you don't, TRY will display your total score and stop the procedure.

10.1 PRINT (COMMAND)

Prints its input on screen.

Syntax: PRINT <input>

PRINT is a very useful command whose input can be a word, a list, a number, an Arithmetic Operation, or a variable.

EXAMPLE:

? PRINT "HELLO

HELLO

A word must be preceded by a double quotation mark; otherwise you will get an error message.

EXAMPLE:

? PRINT [HELLO, HOW ARE YOU?]

HELLO, HOW ARE YOU?

A list must be enclosed in a pair of square brackets. If the list is empty, PRINT will leave a blank line on the screen.

EXAMPLE

? PRINT 1234

1234

A number needs not be preceded by a double quotation mark; nor should it be enclosed in a pair of square brackets.

EXAMPLE:

```
? PRINT SUM QUOT 20 4 6

11
```

SUM QUOT 20 4 6 is an Arithmetic Operation. PRINT prints its output on screen. The input of PRINT can also be a variable:

EXAMPLE:

```
? MAKE "CANADA "OTTAWA

? PRINT "CANADA   PRINT  :CANADA

CANADA
OTTAWA
```

In the MAKE command, "CANADA" bears a value of "OTTAWA". The first PRINT command asks Rogo to print the word "CANADA" and the second its value "OTTAWA".

10.2 LOCATE (OPERATION)

Locates the character position for PRINT.

Syntax: LOCATE <column> <row>

A pair of co-ordinates represents a point on the 40-column screen. The value of the column ranges horizontally from 1 through 40, whereas that of row ranges vertically from 1 through 24.

EXAMPLE:

```
? LOCATE 20 10

? PRINT [WHAT ARE YOU DOING?]
```

The list in PRINT will be printed at the 20th column and 10th row. The position to print is specified in the LOCATE operation. Now, enter another PRINT command:

EXAMPLE:

```
? PRINT [WHY SHOULD I TELL YOU?]
```

Oops! The list is printed back at the default position. You have to use a LOCATE operation for every PRINT command and the LOCATE position must be specified before the PRINT command.

EXAMPLE:

```
? LOCATE 5 5 PRINT [WHAT ARE YOU DOING?]

? LOCATE 25 20 PRINT [WHY SHOULD I TELL YOU?]
```

If the position specified in the LOCATE operation is occupied by other display, the printed list will write over the original display. To avoid confusion, you had better clear the Text Screen first before printing something:

EXAMPLE:

```
? CLS LOCATE 20 10 PRINT "BONJOUR!!
```

CLS is a Screen Command that clears the screen and locates the cursor at the upper left hand corner of the screen (read 12.1 for details).

10.3 READC (OPERATION)

Reads the keyboard and outputs the first word entered.

Syntax: READC

EXAMPLE:

```
? READC
```

Rogo will read the keyboard and output the first word typed. The word is delimited by a space.

EXAMPLE:

```
? PRINT READC

HOW ARE YOU

HOW
```

Say you enter HOW ARE YOU. When Rogo reads the keyboard, it notices that there is a space after HOW which is then regarded as the first word and whatever follows the space is discarded. When Rogo prints READC, only HOW is printed.

EXAMPLE:

```
? PRINT READC

A1234567890*"[]

A1234567890*"[]

? PRINT READC

1ABC123

1ABC123
```

The word read and output by READC can be made up of any alphanumeric characters and symbols as long as it does not exceed 254 characters. But the concept is a little bit different when a word read by READC becomes the value of a variable:

EXAMPLE:

```
? MAKE "A READC PRINT :A

ABC123

ABC123

? MAKE "A READC PRINT :A

123ABC456

123
```

You enter 123ABC456 and MAKE it into the value of A. Since it begins with a number, the value of A becomes a number. So, when you print the content of A, only 123 is printed and the rest is ignored.

EXAMPLE:

```
? MAKE "A READC

123

? IF :A = 123 [PRINT [123 IS A NUMBER]]

123 IS A NUMBER

? MAKE "B READC

AB1

? IF :B = "AB1 [PRINT [AB1 IS A WORD]]

AB1 IS A WORD
```

Notice that in IF commands, a word read by READC must be preceded by a double quotation mark.

EXAMPLE:

```
? TO "DANCE

>PRINT [WANT ME TO DANCE? "Y" OR "N".]

>MAKE "ANS READC

>IF :ANS = "N [PRINT [FORGET IT] STOP]

>IF :ANS = "Y [PRINT "GOOD]

>REPEAT 10 [BC 200 LD 200 FU 300 LD 200 WC 200 FC 150]

>END

>^Z
```

10.4 READL (OPERATION)

Reads the keyboard and outputs the line entered as a list.

Syntax: READL

NOTE : When READL is executed, the line to be read is terminated by pressing ENTER.

EXAMPLE:

```
? READL
```

Whatever you type will be read and output by READL.

EXAMPLE:

```
? MAKE "LINE READL REPEAT 3 [PRINT :LINE]

I'LL NEVER TALK IN CLASS AGAIN

I'LL NEVER TALK IN CLASS AGAIN

I'LL NEVER TALK IN CLASS AGAIN

I'LL NEVER TALK IN CLASS AGAIN
```

The first line is entered by you. Rogo reads the line and repeats printing it thrice as specified.

EXAMPLE:

```
? TO "QUIZ

>PRINT [WHERE IS THE HOTTEST PLACE ON EARTH?]

>MAKE "ANS READL

>TEST :ANS = [DESERT]

>IFFALSE [PRINT "SORRY! BA 300 BC 300 STOP]

>IFTRUE [PRINT "GOOD! FD 300 FU 300 STOP]

>END

>^Z
```

Since the answer you type is output in a list, the word DESERT must be enclosed in square brackets.

10.5 J1 (OPERATION)

Outputs true if the specified switch of Joystick Port 1 of the computer is closed.

Syntax: J1 <switch number>

J1 is used to recognise the direction of the joystick connected to the Joystick Port 1 of your MSX computer.

Before we go on, let's connect the joysticks first:

1. Switch off your computer and monitor.

2. Connect a joystick to the Joystick Port 1 of your computer. Do the same for Joystick Port 2. Make sure the joysticks are compatible with your machine. Read your User's Manual for details.

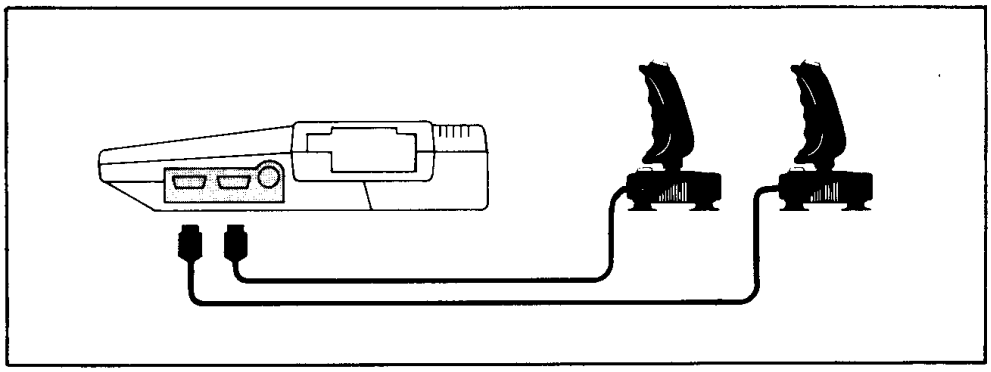
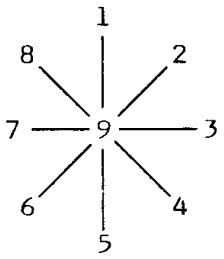


Fig. 10.1 Connecting Joysticks

A joystick can move in 9 directions:



EXAMPLE:

J1 2 - Outputs true when the joystick in Joystick Port 1 moves to the Northeast.

J1 5 - Outputs true when the joystick in Joystick Port 1 moves to the South.

Since J1 and J2 work alike, we will give you a thorough example in 10.6.

10.6 J2 (OPERATION)

Outputs true if the specified switch of Joystick Port 2 of the computer is closed.

Syntax: J2 <switch number>

EXAMPLE:

```
? TO "J_CTRL

>IF J1 1 [LD 200] IF J2 1 [FD 200]

>IF J1 3 [BA 200] IF J2 3 [WA 200]

>IF J1 5 [LU 200] IF J2 5 [FU 200]
```

```
>IF J1 7 [BC 200] IF J2 7 [WC 200]

>END

>^Z

? REPEAT 1000 [J_CTRL]
```

The procedure J_CTRL can only detect one move every time it is executed. REPEAT 1000 [J_CTRL] repeats the execution of J_CTRL for a thousand time so that you can play with your joysticks for a while.

10.7 SAVE "CAS (COMMAND)

Saves a workspace onto a tape.

Syntax: SAVE "CAS

NOTE : SAVE "CAS saves all the procedures in the workspace. The workspace is a part of the RAM where the procedures are stored.

STEPS

1. Prepare a tape (read 3.4 for details).

2. Type SAVE "CAS on the Text Screen. DON'T press ENTER yet. The following message will be displayed on the screen:

   PLEASE REWIND TAPE TO THE BEGINNING

   PRESS 'ENTER' WHEN READY.....

   Since you have rewinded the tape in Step 1, you can go to the next step.

3. Press PLAY and RECORD on the recorder.

4. Press ENTER.

   Rogo will start saving all the procedures which are currently in the RAM. When a part of the workspace is being saved, a SAVING: WORKSPACE message is displayed. After that part of the workspace has been saved, a WRITE OK message appears.

   EXAMPLE:

   SAVING: WORKSPACE WRITE OK

   SAVING: WORKSPACE

   When all the procedures have been saved, the "SAVING COMPLETED" message and the prompt sign are displayed.

   If an error occurs in the course of saving, an error message "WRITE ERROR" will appear and Rogo will stop saving. You will be brought back to the Text Screen. Repeat the steps from the beginning. If the error persists, refer to Appendix A for a solution.

   The motor of the recorder will be stopped automatically when all the procedures in the workspace have been saved.

5. Press STOP on the recorder.

10.8 LOAD "CAS (COMMAND)

Loads a workspace into the RAM from a tape.

Syntax: LOAD "CAS

STEPS

1. Prepare the tape that holds the workspace (read 3.4 for details).

2. Type LOAD "CAS on the Text Screen. The following message will be displayed:

   PLEASE REWIND TAPE TO THE BEGINNING

   PRESS 'ENTER' WHEN READY.....

   Since you have rewinded the tape in Step 1, you can go to the next step.

3. Press ENTER.

4. Press PLAY on the recorder.

   Every procedure in the tape will be loaded into the RAM. When a part of the saved workspace is found, a FOUND: WORKSPACE is displayed and a READ OK message appears when the loading of that part of the workspace has been completed.

   EXAMPLE:

   FOUND: WORKSPACE READ OK

   FOUND: WORKSPACE

   If Rogo encounters an error in loading, the READ ERROR message will be displayed and the loading will be stopped. Repeat from Step 1. If the problem persists, refer to Appendix A for a solution.

   When all the procedures saved on the tape are loaded, the END OF TAPE message and prompt sign are displayed. If the message does not appear at the end of the tape, an error must have occured. Switch the computer off and on again. Repeat from Step 1. If the problem persists, refer to Appendix A for a solution.

5. STOP the recorder.

10.9 CHAPTER SUMMARY

PRIMITIVE	SYNTAX	DESCRIPTION
PRINT	PRINT <input>	Prints its input on the screen.
LOCATE	LOCATE <column> <row>	Locates the character position for PRINT.
READC	READC	Reads the keyboard and outputs the first word entered.
READL	READL	Reads the keyboard and outputs the line entered as a list.
J1	J1 <switch number>	Outputs true if the specified switch of Joystick Port 1 of the computer is closed.
J2	J2 <switch number>	Outputs true if the specified switch of Joystick Port 2 of the computer is closed.
SAVE	SAVE "CAS	Saves a workspace into a tape.
LOAD	SAVE "CAS	Loads a workspace into the RAM from a tape.

CHAPTER 11
WORDS AND LISTS PROCESSING

Nine additional primitives are supported in Version 1.1 of the Rogo Cartridge to help manipulate words and lists. They are BF, FIRST, WORD, SE, LPUT, NUMBERP, WORDP, LISTP, and EMPTYP. All of them are operations.

Below are two little games that demonstrate some of the potential of these operations. The first is a quiz. The second is a story generator.

EXAMPLE:

```
? TO "QUIZ

>MAKE "COUNTRY [AUSTRALIA BELGIUM BRAZIL CANADA CHINA FRANCE
 ITALY JAPAN NORWAY SPAIN SWEDEN U.K. U.S.A.]

>MAKE "CAPITAL [CANBERRA BRUSSELS BRASILIA OTTAWA BEIJING PARIS
 ROME TOKYO OSLO MADRID STOCKHOLM LONDON WASHINGTON]

>CLS

>REPEAT 13 [PRINT SE SE [WHAT'S THE CAPITAL OF] FIRST :COUNTRY
 "? TEST READC = FIRST :CAPITAL IFTRUE [FD 150 FU 150 PRINT
 "RIGHT!] IFFALSE [BC 150 BA 150 PRINT "SORRY!] PRINT [] MAKE
 "COUNTRY BF :COUNTRY MAKE "CAPITAL BF :CAPITAL]

>END

>AZ
```

QUIZ only gives you the framework of writing a quiz about countries and capitals. You can add your own list of countries. But, note that the number of times to REPEAT should correspond with the total number of countries.

EXAMPLE:

```
? TO "STORY :NAME

>PRINT SE SE [A LONG LONG TIME AGO, IN A FAR AWAY GALAXY, THERE
 WAS A] FIRST :NAME ".

>PRINT SE SE [HE WAS TELLING] BF :NAME [A STORY.]

>PRINT [HE SAID,]

>IF NOT EMPTYP BF :NAME [STORY BF :NAME]

>END

>ΛZ

? TO "ASK

>PRINT [HAVE YOU EVER HEARD OF THE NEVER-ENDING STORY OF THE
 GALAXY?] WAIT 100

>PRINT [LET ME TELL YOU.] WAIT 50

>PRINT [ENTER A LIST OF MONTERS.]

>MAKE "ENTRY READL

>PRINT [] PRINT [*****]

>END

>ΛZ
```

```
? TO "START

>ASK

>STORY :ENTRY

>PRINT [.....]

>END

>^Z
```

The story generator consists of 3 modules. You begin with START. After you have entered a list of monsters, STORY tells you a story about the monsters. Similar to QUIZ, START can be edited as you like. See if you can make up an interesting story.

11.1 BF (OPERATION)

Outputs all but the first element of the input.

Syntax: BF <object>

NOTE : <object> can be a word, a list, a number or a variable whose value is a word, a list or a number.

EXAMPLE:

```
? PRINT BF "ABC

BC

? PRINT BF [U.S.A. WASHINGTON D.C.]

WASHINGTON D.C.

? PRINT BF [[U.S.A. WASHINGTON D.C.] [CANADA OTTAWA]]
```

```
[CANADA OTTAWA]

? PRINT BF 123

23

? MAKE "A [THIS IS A LOVELY WORLD]

? PRINT BF :A

IS A LOVELY WORLD
```

Outputing the content of an empty word or list is always erroneous:

EXAMPLE:

```
? PRINT BF "

SYNTAX ERROR IN BF

? PRINT BF []

SYNTAX ERROR IN BF
```

EXAMPLE:

```
? TO "BEGIN :OBJ

>MAKE "INV []

>REV :OBJ

>END

>^Z
```

```
? TO "REV :OBJ

>IF EMPTYP :OBJ [PRINT :INV STOP]

>MAKE "INV SE FIRST :OBJ :INV

>REV BF :OBJ

>END

>^Z

? BEGIN [WHO ARE YOU]

YOU ARE WHO

? BEGIN [BUNNY]

BUNNY

? BEGIN "BUNNY

Y N N U B
```

EMPTYP is a primitive that outputs true if the input is an empty word or empty list. FIRST outputs the first element of the input, whereas SE outputs the inputs as a list (read 11.2, 11.4 and 11.9 for details). In REV, EMPTYP acts as a control. Let's say you enter WHO ARE YOU as the input. FIRST outputs WHO and SE links it with the content of INV which has been initialized as an empty list. Then BF outputs ARE YOU which becomes the input of REV. FIRST outputs ARE and SE links it with WHO. YOU is now the only element of the input, SE links it with ARE WHO. When the input becomes empty, the content of INV, YOU ARE WHO in this case, is printed and the loop stops.

11.2 FIRST (OPERATION)

Outputs the first element of the input.

Syntax: FIRST <object>

NOTE : <object> can be a word, a list, a number or a variable. Outputing the first element of an empty word or list is always an error.

EXAMPLE:

```
? PRINT FIRST "ABC

A

? PRINT FIRST [U.K. LONDON]

U.K.

? PRINT FIRST [[U.K. LONDON] [JAPAN TOKYO]]

[U.K. LONDON]

? PRINT FIRST 123

1

? MAKE "A [THIS IS A LOVELY WORLD]

? PRINT FIRST :A PRINT BF :A

THIS

IS A LOVELY WORLD
```

```
? TO "QUESS

>PRINT SE [WHAT'S THE CAPITAL OF] FIRST [CANADA OTTAWA]

>MAKE "ANS READC

>IF :ANS = BF [CANADA OTTAWA] [PRINT "RIGHT! STOP]

>PRINT "SORRY!

>END

>^Z
```

SE is a primitive that outputs its inputs in the form of a list. Read 11.4 for details.

11.3 WORD (OPERATION)

Outputs a word formed by its inputs.

Syntax: WORD <word1> <word2>

NOTE : 1. <word1> and <word2> must be words, numbers, or variables. Anything other than these is an error.

2. WORD can read only 2 inputs at a time and the number of characters in total must not exceed 254.

EXAMPLE:

```
? PRINT WORD "LITTER "BUG

LITTERBUG
```

```
? PRINT WORD WORD WORD "O "P "E "C

OPEC

? PRINT WORD WORD WORD "SVI 2000 "C "ROGO

SVI2000CROGO

? MAKE "A "LET

? PRINT WORD "HAM :A

HAMLET

? PRINT WORD "UNITED [STATES OF AMERICA]

SYNTAX ERROR IN WORD
```

11.4 SE (OPERATION)

Outputs the inputs in form of a list.

Syntax: SE <object1> <object2>

NOTE : 1. <object1> and <object2> can be words, lists, numbers, or variables.

2. SE can read only 2 inputs at a time and the total number of characters must not exceed 254; otherwise the STRING TOO LONG IN SE error message will appear. EXAMPLE:

```
? PRINT SE "BIG [BILLY GOAT GRUFF]

BIG BILLY GOAT GRUFF
```

```
? PRINT SE SE "GREAT "BIG [BILLY GOAT GRUFF]

GREAT BIG BILLY GOAT GRUFF

? TO "GREETING

>PRINT [WHAT'S YOUR NAME?]

>PRINT SE SE [HOW ARE YOU,] READL "?

>END

>^Z

? GREETING

WHAT'S YOUR NAME?

NOBODY

HOW ARE YOU, NOBODY ?
```

11.5 LPUT (OPERATION)

Outputs a new list formed by combining the first and second input.

Syntax: LPUT <list> <object>

NOTE : 1. The first input must be a list.

2. The second input can be a word, a list, a number, or a variable.

EXAMPLE:

```
? PRINT LPUT [UNITED STATES OF] "AMERICA

UNITED STATES OF AMERICA
```

The differences between SE and LPUT are demonstrated in the following examples:

EXAMPLE:

```
? PRINT SE [I FORGOT TO REMEMBER.] [I REMEMBERED I FORGOT.]

I FORGOT TO REMEMBER. I REMEMBERED I FORGOT.

? PRINT LPUT [I FORGOT TO REMEMBER.] [I REMEMBERED I FORGOT.]

I FORGOT TO REMEMBER. [I REMEMBERED I FORGOT.]


? PRINT SE SE [TO BE] "OR [NOT TO BE]

TO BE OR NOT TO BE

? PRINT LPUT LPUT [TO BE] "OR [NOT TO BE]

TO BE OR [NOT TO BE]
```

Notice the square brackets. SE forms a new list by linking the elements of both inputs, whereas LPUT forms the list by simply combining the inputs together. So, if the second input of LPUT is a list, the square brackets will not be stripped off. Besides, the first input of LPUT must be a list which is not a must in SE.

11.6 NUMBERP (OPERATION)

Outputs true if the input is a number; otherwise false.

Syntax: NUMBERP <input>

NUMBERP, WORDP, LISTP, and EMPTYP are four operations that work very much alike. They are useful in controlling the flow of procedures. To give you an idea how they work, we have included them in an example in 11.9. But for the time being, let's look at the function of each of these operations.

EXAMPLE:

```
? PRINT NUMBERP 123

TRUE

? PRINT NUMBERP "HELLO

FALSE
```

11.7 WORDP (OPERATION)

Outputs true if the input is a word or number; otherwise false.

Syntax: WORDP <input>

EXAMPLE:

```
? PRINT WORDP 123

TRUE
```

```
? PRINT WORDP "HELLO

TRUE

? PRINT WORDP [A WICKED OLD BEAR]

FALSE
```

11.8 LISTP (OPERATION)

Outputs true if the input is a list; otherwise false.

Syntax: LISTP <input>

EXAMPLE:

```
? PRINT LISTP "HELLO

FALSE

? PRINT LISTP [A WICKED OLD WOLF]

TRUE

? PRINT LISTP []

TRUE
```

11.9 EMPTYP (OPERATION)

Outputs true if the input is an empty word or list; otherwise false.

Syntax: EMPTYP <input>

EXAMPLE:

```
? PRINT EMPTYP BF "A

TRUE

? PRINT EMPTYP []

TRUE

? PRINT EMPTYP 123

FALSE

? PRINT EMPTYP "HELLO

FALSE

? PRINT EMPTYP [A SLY OLD FOX]

FALSE
```

As mentioned in 11.6, NUMBERP, WORDP, LISTP, and EMPTYP are very useful in controlling the flow of procedures. Below is an example that reveals the capabilities of these operations while helping you introduce the basic concept of Rogo list, word, and number to your friends.

EXAMPLE:

```
? TO "TEACH
>PRINT [ENTER SOMETHING.]
>MAKE "ENTRY FIRST READL
>IF EMPTYP :ENTRY [PRINT [YOU DIDN'T ENTER ANYTHING.]]
>IF NUMBERP :ENTRY [PRINT [THE FIRST ELEMENT IS A NUMBER.]]
>IF WORDP :ENTRY [PRINT [THE FIRST ELEMENT IS A WORD.]]
>IF AND NUMBERP :ENTRY WORDP :ENTRY [PRINT [THE FIRST ELEMENT
 IS BOTH A NUMBER AND A WORD.]]
>IF LISTP :ENTRY [PRINT [THE FIRST ELEMENT IS A LIST.]]
>ASK
>END
>ΔZ

? TO "ASK
>PRINT [TRY AGAIN? Y/N.]
>TEST READC = "Y
>IFTRUE [TEACH]
>IFFALSE [PRINT [BYE FOR NOW.] STOP]
>END
>ΔZ
```

11.10 CHAPTER SUMMARY

PRIMITIVE	SYNTAX	DESCRIPTION
BF	BF <object>	Outputs all but the first element of the input.
FIRST	FIRST <object>	Outputs the first element of the input.
WORD	WORD <word1> <word2>	Outputs a new word formed by its inputs.
SE	SE <object1> <object2>	Outputs the inputs in the form of a list.
LPUT	LPUT <list> <object>	Outputs a new list formed by combining the inputs.
NUMBERP	NUMBERP <input>	Outputs true if the input is a number; otherwise false.
WORDP	WORDP <input>	Outputs true if the input is a word or a number; otherwise false.
LISTP	LISTP <input>	Outputs true if the input is a list; otherwise false.
EMPTYP	EMPTYP <input>	Outputs true if the input is an empty word or empty list; otherwise false.

CHAPTER 12
SCREEN COMMANDS

There are 5 Screen Commands in Rogo: CLS, SHOWARM, HIDEARM, SHOWTEXT, and HIDETEXT. CLS clears the Text Screen, SHOWARM invokes the Graphic Screen, HIDEARM exits the Graphic Screen, whereas SHOWTEXT and HIDETEXT toggle the display of text in the Graphic Screen. Below is an example to illustrate how to preview a movement before it actually takes place.

EXAMPLE:

```
? TO "MOVE

>BA 200 BC 200 LU 200 LD 200 FU 200 FD 200 WA 200 WC 200 FO 200
 FC 200

>END

>^Z

? TO "PREVIEW

>SHOWARM

>MOVE

>HIDEARM

>END

>^Z

? TO "START

>PREVIEW

>MOVE

>END

>^Z
```

The above example consists of 3 modules: MOVE, PREVIEW, and START. If you execute START, PREVIEW is called and MOVE is simulated on the Graphic Screen. When the simulation is completed, the actual Robotarm moves as specified in MOVE.

12.1 CLS (COMMAND)

Clears the Graphic Screen.

Syntax: CLS

Whenever the Text Screen gets too messy, too full, or you simply want to clear it for another purpose, such as in the case of locating the position to print, enter CLS. The Text Screen will then be cleared and the cursor will return to the top left-hand corner (home position).

12.2 SHOWARM (COMMAND)

Invokes the Graphic Screen.

Syntax: SHOWARM

Entering this command will replace the Text Screen with the Graphic Screen where the simulated movement of the Robotarm and the angles of movement of the respective axes are displayed.

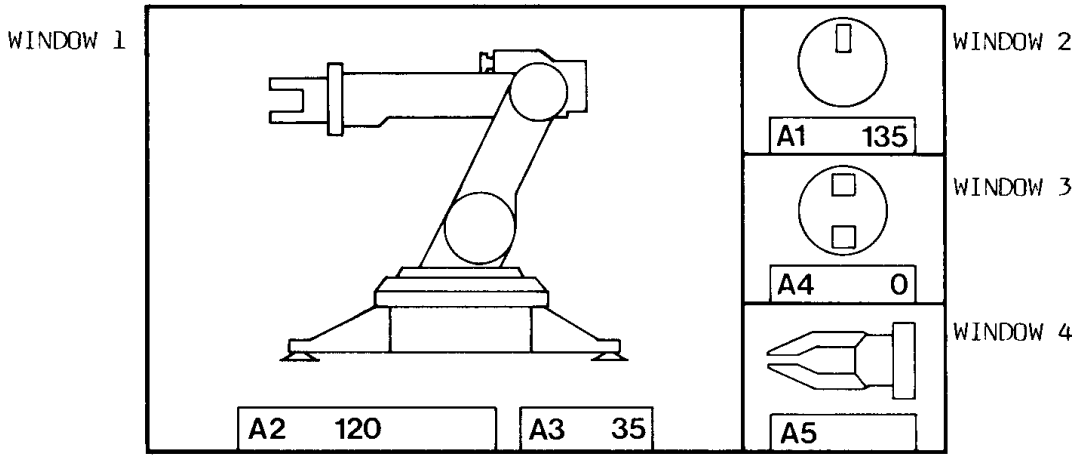


Fig. 12.1 The Graphic Screen

As you have noticed in the above illustration, the Graphic Screen is made up of 4 windows:

Window 1 - It displays the outline of the Robotarm and simulates the movement of axis 2 and 3 (read Fig. 5.1 for details).

Window 2 - It is a top view of the Robotarm and shows the rotation of axis 1.

Window 3 - It displays a front view of the forceps and reveals the rotation of axis 4.

Window 4 - It is a close up of the front of the forceps.

12.3 HIDEARM (COMMAND)

Exits the Graphic Screen and returns to the Text Screen.

Syntax: HIDEARM

Entering this command will exit the Graphic Screen and you will be brought back to the Text Screen.

12.4 SHOWTEXT (COMMAND)

Displays text in the Graphic Screen.

Syntax: SHOWTEXT

You can enter commands as usual in the Graphic Screen. Both the actual Robotarm and the simulated one will move accordingly. What you type will not be displayed by default. If you feel more comfortable to read as you type, enter SHOWTEXT. What you type will then be displayed in 2 command lines on top of the Graphic Screen.

12.5 HIDETEXT (COMMAND)

Hides text in the Graphic Screen.

Syntax: HIDETEXT

It hides text in the Graphic Screen. It is the default choice.

12.6 CHAPTER SUMMARY

PRIMITIVE	SYNTAX	DESCRIPTION
CLS	CLS	Clears the Text Screen.
SHOWARM	SHOWARM	Invokes the Graphic Screen.
HIDEARM	HIDEARM	Exits the Graphic Screen.
SHOWTEXT	SHOWTEXT	Shows text in the Graphic Screen.
HIDETEXT	HIDETEXT	Hides text in the Graphic Screen.

APPENDIX A
TROUBLE-SHOOTING

Whenever Rogo encounters an error, it displays an error message:

ERROR MESSAGE	DESCRIPTION
SYNTAX ERROR IN ...	The syntax of a certain command is incorrect. Check thoroughly the commands you have entered to see if they are in the correct syntax.
I DON'T KNOW HOW TO ...	You have entered something that Rogo doesn't understand. Check your spelling or type COMMAND to see if the specified command has been built-in.
NOT ENOUGH INPUTS TO ...	The specified command or operation requires the appropriate inputs before it can be executed. Retype the commands and be sure to enter the necessary inputs as well.
OVERFLOW ERROR IN ...	The number resulting from the latest calculation or the number you have entered is too big. Try to use a smaller number.
UNDERFLOW ERROR IN ...	The number resulting from the latest calculation or the number you have entered is too small. Try to use a larger number.

ERROR MESSAGE	DESCRIPTION
SYSTEM BUG ERROR	Something has caused fatal errors to the system. If Rogo is halted, reboot Rogo again.

NOTE: You may have noticed a primitive called SELFTEST when you listed the directory of primitives. Running SELFTEST will check the condition of the Rogo cartridge. The general users may not find this primitive useful as it is meant for the factory users only.

APPENDIX B
QUICK REFERENCE SHEET

ROGO TERMINOLOGY

COMMAND	-	a keyword that causes Rogo to perform a specific function
WORD	-	a word is a set of characters starting with a letter and preceded by a double quotation mark
LIST	-	a list is a set of characters enclosed by a pair of square brackets
INSTRUCTIONLIST	-	an instruction or set of instructions enclosed by a pair of square brackets
VARIABLE	-	a "container" that holds a certain value
OBJECT	-	can be a word, list, the content of a variable, or the result of a Logical Operation

SPECIAL PUNCTUATION

OPERATION	-	a primitive that causes Rogo to output a certain result without specifying what to do with it

SYNTAX NOTATION

< >	-	the content within the angle brackets specifies the nature of an input
"	-	comes immediately before a Rogo word
[ ]	-	encloses a Rogo list
:	-	placed before a word which is to be treated as a variable

(NOTE: Below is the summary of all primitives in Logo. Every operation is marked with an asterisk, *.)

ROBOTARM CONTROL COMMANDS

BC <time period> - rotates the base clockwise
BA <time period> - rotates the base anticlockwise
LU <time period> - raises the lowerarm
LD <time period> - lowers the lowerarm
FU <time period> - raises the forearm
FD <time period> - lowers the forearm
WC <time period> - rotates the wrist clockwise
WA <time period> - rotates the wrist anticlockwise
FO <time period> - opens the forceps
FC <time period> - closes the forceps

<time period> is the length of time you want Rogo to move the specified part of the Robotarm. It can be expressed in a positive real number, a variable, or an Arithmetic Operation.

VARIABLES

MAKE <variable name> <value> - assigns a value to a variable
THING <variable name>* - outputs the value of the named variable

<variable name> can be made up of any combination of alphanumeric characters and symbols as long as it begins with a letter and does not exceed 9 characters.

<value> can be a word, a number, a list, an Arithmetic Operation, or an I/O Operation (J1, J2, READC or READL).

ARITHMETIC OPERATIONS

SUM <real number A> <real number B>* - adds A and B
DIFF <real number A> <real number B>* - subtracts B from A
PROD <real number A> <real number B>* - multiplies A with B
QUOT <real number A> <real number B>* - divides A by B
INT <real number>* - outputs the integer portion
ROUND <real number>* - outputs the integer round off

LOGICAL OPERATIONS

<condition 1> < <condition 2>* - outputs true when <condition 1> is less than <condition 2>

<condition 1> > <condition 2>* - outputs true when <condition 1> is greater than <condition 2>

<condition 1> = <condition 2>* - outputs true when <condition 1> is equal to <condition 2>

<condition 1> <= <condition 2>* - outputs true when <condition 1> is less than or equal to <condition 2>

<condition 1> >= <condition 2>* - outputs true when <condition 1> is greater than or equal to <condition 2>

<condition 1> < > <condition 2> - outputs true when <condition 1> is not equal to <condition 2>

<condition 1> and <condition 2> in <, >, =, <=, >=, and < > can be numbers, Arithmetic Operations, letters, or variables.

AND <condition 1> <condition 2>* - outputs true when both <condition 1> and <condition 2> are true

NOT <condition>* - outputs true when the condition is not satisfied

OR <condition 1> <condition 2>* - outputs true when either or both <condition 1> and <condition 2> are true

XOR <condition 1> <condition 2>* - outputs true when either <condition 1> or <condition 2> is true

<condition 1> and <condition 2> in AND, NOT, OR, and XOR must be operations composed of <, >, =, <=, >= , < >, or I/O operations involving true or false.

CONTROL FLOW COMMANDS

IF <condition> <instructionlist> - executes the instructionlist if the condition is true

<condition> can be a Logical Operation or an I/O Operation involving J1 or J2.

TEST <condition>* - remembers if the condition is true or false

<condition> can be a Logical Operation or an I/O Operation concerning true or false (e.g. J1).

IFTRUE <instructionlist> - executes the instructionlist if the output of the latest TEST operation is true

IFFALSE <instructionlist> - executes the instructionlist if the output of the latest TEST operation is false

<instructionlist> in IFTRUE and IFFALSE can be made up of user-defined commands or primitives

REPEAT <number> <instructionlist>- repeats the instructionlist for a specified number of times.

<number> must be a positive whole number.

STOP - stops a running procedure and returns control to the top level

WAIT <time period> - waits for a specified time period

<time period> must be a positive whole number or an Arithmetic Operation.

OP <object> - makes a specified Rogo object an output of a procedure

RUN <instructionlist> - runs an instructionlist

<instructionlist> in RUN can be a user-defined command, a primitive, or an operation.

INPUT/OUTPUT COMMANDS

PRINT <input> - prints the input on the Text Screen

<input> can be a word, a list, a number, an Arithmetic Operation, or a variable.

LOCATE <column><row>* - locates the character position for PRINT

READC* - reads and outputs the first word typed

READL* - reads the first line entered and outputs the line as a list

J1 <switch number>* - outputs true if the specified switch of Joystick Port 1 of the computer is closed

J2 <switch number>* - outputs true if the specified switch of Joystick Port 2 of the computer is closed

SAVE "CAS - saves the workspace onto a tape

LOAD "CAS - loads the workspace into the RAM from a tape

WORDS/LISTS PROCESSING OPERATIONS

BF <object>* - outputs all but the first element of the object

FIRST <object>* - outputs the first element of the object

<object> can be a word, a list, a number, or a variable.

WORD <word 1> <word 2>* - outputs a word formed by <word 1> and <word 2>

<word 1> and <word 2> must be words, numbers, or variables.

SE <objectl><object 2>* - outputs <object 1> and <object 2> in form of a list

<object 1> and <object 2> can be words, lists, numbers or variables.

LPUT <list> <object>* - outputs a new list formed by <list> and <object>

<object> can be a word, a list, a number, or a variable.

NUMBERP <input>* - outputs true if the input is a number; otherwise false

WORDP <input>* - outputs true if the input is a word; otherwise false

LISTP <input>* - outputs true if the input is a list; otherwise false

EMPTYP <input>* - outputs true if the input is an empty word or list; otherwise false

SCREEN COMMANDS

CLS - clears the Text Screen
SHOWARM - invokes the Graphic Screen
HIDEARM - exits the Graphic Screen
SHOWTEXT - displays text in Graphic Screen
HIDETEXT - hides text in Graphic Screen

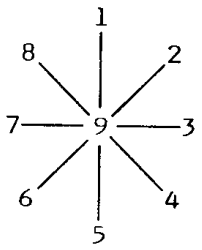
OTHERS

TO - defines a new command
EDIT - edits a previously defined command
DIR - displays the directory of all defined procedures
COMMAND - displays the directory of Rogo primitives
SELFTEST - checks the condition of Rogo

EDITING KEYS IN THE COMMAND EDITOR

F1 / ^T - beginning of the procedure
F2 / ^B - bottom of the procedure
F3 / ^A - left one word
F4 / ^F - right one word
F5 / ^Z - exits the Command Editor
F6 / ^P - up screen 10 lines
F7 / ^N - down screen 10 lines
F8 / ^D - deletes line
F9 / ^I - inserts line
F10 / ^Q - aborts without saving
DEL - deletes character
INS ON/OFF - inserts character in INS ON and overwrites when OFF
→ - right character
← - left character
↑ - up one line
↓ - down one line

JOYSTICK SWITCH NUMBERS



APPENDIX C
USEFUL SAMPLE PROCEDURES

Below is a set of sample procedures which you can apply as building blocks in your own procedures. Note that many of the procedures are interrelated and therefore cannot operate on their own.

EXAMPLE:

```
? TO "ABS :VALUE

>IF :VALUE < 0 [OP DIFF 0 :X]

>OP :VALUE

>END

>^Z

? PRINT ABS -1

1

? PRINT ABS 1

1
```

ABS is a procedure to output the ABSOLUTE VALUE of a real number. If the input is a negative number, OP will output the difference between 0 and the input and then exit the procedure automatically. Remember that ABS is an operation. You have to PRINT ABS in order to read the result on the screen.

EXAMPLE:

```
? TO "REMAIN :A :B

>OP ROUND PROD ABS :B DIFF QUOT :A :B INT QUOT :A :B

>END

>^Z
```

APPENDIX C

```
? PRINT REMAIN 5 2

1

? PRINT REMAIN 10 4

2
```

REMAIN is an operation to output the REMAINDER of the division of X by Y. Make sure you have defined ABS before executing REMAIN since the former becomes one of the commands in the latter.

EXAMPLE:

```
? TO "ODDP :NUMBER

>IF REMAIN :NUMBER 2 = 1 [PRINT "ODD] STOP

>PRINT "EVEN

>END

>^Z

? ODDP 5

ODD

? ODDP 10

EVEN
```

ODDP is a procedure to check whether the input is ODD or EVEN. It does it by making use of the REMAIN operation to output the remainder of the division between the input and 2.

EXAMPLE:

```
? TO "SIN :X
```

```
>MAKE "N1 1 MAKE "N2 -1 MAKE "X1 0

>MAKE "X2 1 MAKE "X3 PROD :X 0.017453

>MAKE "N 1

>REPEAT 15 [MAKE "N1 PROD :N1 :N MAKE "X2 PROD :X2 :X3 IF ODDP
 :N [MAKE "N2 PROD -1 :N2 MAKE "X1 SUM :X1 QUOT PROD :X2 :N2
 :N1] MAKE "N SUM :N 1]

>OP :X1

>END

>ΔZ

? PRINT SIN 45

. 70709747319768
```

SIN is a rather complex operation to output the SINE value of the input. The procedure is quite mathematics-oriented, which may be a bit puzzling for those who are not that strong in trigonometry.

EXAMPLE:

```
? TO "FACT :X

>IF :X = 0 [OP 1]

>OP PROD :X FACT DIFF :X 1

>END

>ΛZ

? PRINT FACT 4

24
```

APPENDIX C

```
? PRINT FACT 5

120
```

FACT is an operation to output the FACTORIAL of the input. For instance, if you enter FACT 4, FACT will output the result of 4 X 3 X 2 X 1, which equals 24.

FACT is different from the rest in that it is recursive. Note the FACT DIFF :X 1 command. It decreases the input by 1 and sends it back to the beginning of the procedure. The IF command acts as a control to the recursive command. Whenever the value of the input equals 1, it outputs 1 and exits the loop automatically.

APPENDIX D
RECURSIVE COMMANDS

When a procedure calls itself, it becomes recursive. Let's look at an example first:

EXAMPLE:

```
? TO "COUNTBACK :START

>MAKE "NUMBER :START

>PRINT :NUMBER

>MAKE "NUMBER DIFF :NUMBER 1

>COUNTBACK :NUMBER

>END

>^Z
```

Don't execute the above example yet. COUNTBACK is a recursive procedure that deducts one from the number everytime the number passes through the loop. It starts from the number you enter as the input of COUNTBACK. What makes the procedure recursive is the command COUNTBACK :NUMBER which, whenever executed, brings the procedure back to the beginning again and thus an endless loop is formed.

If you execute COUNTBACK, you will find you are caught in a runaway procedure that will stop only when the Rogo buffer is used up. Once the Rogo buffer is used up, Rogo exits the procedure automatically and you are brought back to the prompt. To give you an idea how far you can go in a recursive command, let's experiment with COUNTBACK. If you are ready, type COUNTBACK 0.

Whenever COUNTBACK is executed once, a number is printed on the screen. You will get a running list of number in descending order. After a while, you will get a NESTING IS TOO DEEP message and the procedure is exited.

Obviously, that's not a very effective way of controlling a recursive command. There are better ways. For instance, you can make use of the Control Flow Commands. Let's change COUNTBACK as follows:

```
? TO "COUNTBACK :NUMBER

>PRINT :NUMBER

>IF INT :NUMBER = 0 [STOP]

>COUNTBACK DIFF :NUMBER 1

>END

>^Z
```

Notice the IF command. It checks whether the number has reached 0. If it has, it will stop the procedure. In fact, it is up to you when or where to exit a recursive command. Now you can execute the procedure and see what we mean. Don't forget to enter a positive real number along with the command COUNTBACK. If you enter a number smaller than 0 for COUNTBACK to start with, the IF command you have inserted in COUNTBACK as a control will have no effect on the recursive procedure.

EXAMPLE:

```
? TO "MOVE

>IF J1 1 [PRINT [LOWERARM UP] LU 200]

>IF J2 1 [PRINT [FOREARM UP] FU 200]

>IF J1 3 [PRINT [BASE CLOCKWISE] BA 200]

>IF J2 3 [PRINT [WRIST CLOCKWISE] WC 200]
```

```
>IF J1 5 [PRINT [LOWERARM DOWN] LD 200]

>IF J2 5 [PRINT [FOREARM DOWN] FD 200]

>IF J1 7 [PRINT [BASE ANTICLOCKWISE] BA 200]

>IF J2 7 [PRINT [WRIST ANTICLOCKWISE] WA 200]

>END

>^Z

? TO "JOYSTICKS

>TEST AND J1 9 J2 9

>IFTRUE [STOP]

>IFFALSE [MOVE]

>JOYSTICKS

>END

>^Z
```

In the above example, the procedure is made up of 2 modules, JOYSTICKS and MOVE. It begins with JOYSTICKS which tests if the firebuttons of both joysticks are pressed. If they are, JOYSTICKS will stop the procedure. Otherwise, it will execute MOVE. You may have noticed that JOYSTICKS is recursive. It will not stop unless you press both firebuttons together, or the buffer is used up.

APPENDIX D

Consider the two examples below:

EXAMPLE 1

```
? TO "COUNT1 :VARIABLE

>PRINT :NUMBER

>COUNT SUM :NUMBER 1

>END

>^Z
```

EXAMPLE 2

```
? TO "COUNT2 :VARIABLE

>MAKE "NUMBER :VARIABLE

>REPEAT 569 [PRINT :NUMBER MAKE "NUMBER SUM :NUMBER 1]

>END

>^Z
```

EXAMPLE 1 is a recursive command which gives you a list of numbers in ascending order. In normal case, it will loop some 569 times before it stops. EXAMPLE 2 is an ordinary procedure. By using the REPEAT command, it will loop 569 times giving exactly the same result as you may have in EXAMPLE 1. Though the procedures work alike, EXAMPLE 2 is preferable since it will not drain the buffer.

Some final words on recursive commands. When you write a recursive command, you must control it. If you let go a recursive procedure, most probably you will drain the buffer. If you are not sure you can keep a recursive command under control, you'd better leave it alone and use some other commands instead.

FIX LABELS ON JOYSTICKS
AS SHOWN IN MANUAL

PRINTED IN HONG KONG IMPRIME A HONG KONG