INTRAC, A Communication Module for Interactive Programs

Language Manual

Elmqvist, Hilding; Wieslander, Johan

1978

Document Version:
Publisher's PDF, also known as Version of record

CODEN: LUTFD2/(TFRT-3149)/1-060/(1978)

INTRAC

A COMMUNICATION MODULE FOR INTERACTIVE PROGRAMS

LANGUAGE MANUAL

J. WIESLANDER
H. ELMQVIST

Department of Automatic Control
Lund Institute of Technology
August 1978

INTRAC -

A Communication Module for

Interactive Programs,

Language Manual.

Johan Wieslander
Hilding Elmqvist

Dokumentutgivare
Lund Institute of Technology

Handläggare
Karl-Johan Åström

Författare
Johan Wieslander
Hilding Elmqvist

Dokumentnamn
LUTFD2/(TFRT-3149)/1-060/(1978)

Utgivningsdatum
Aug 1978

Dokumentbeteckning

Ärendebeteckning

10T4

Dokumenttitel och undertitel
INTRAC - A Communication Module for Interactive Programs, Language Manual

Referat (sammandrag)
A general purpose communication module for use in interactive programs is described. It implements primarily a command dialogue, but through the use of stored sequences of interactions, here called macros, other forms of interaction is easily implemented, e.g. question & answer dialogue. INTRAC contains features allowing inclusion of normal program elements such as loops, branches, I/O statements, variables and procedures. It thus can serve as a tool in generating problem-oriented interactive languages.

Referat skrivet av
Author

Förslag till ytterligare nyckelord

Klassifikationssystem och -klass(er)

Indextermer (ange källa)

Omfång
60 pages

Övriga bibliografiska uppgifter

Språk
English

Sekretessuppgifter

ISSN

ISBN

Dokumentet kan erhållas från
Department of Automatic Control
Lund Institute of Technology, P O Box 725, S-220 07 Lund, Sweden

Mottagarens uppgifter

Pris

DOKUMENTDATABLAD enligt SIS 62 10 12

SIS-DB 1

UPPGIFTER I DOKUMENTDATABLADET

Nedanstående uppgifter motsvaras av ledtext i blanketten. Beteckningarna består av radgångsnummer och tabuleringsläge (T-läge). De anger startläge, dvs var textrad normalt skall börja.

04T0
DOKUMENTUTGIVARE
ISSUING ORGANIZATION

Fullständigt namn för den organisation (motsv) som utgivit dokumentet (uppgiften kan vara förtryckt).

04T4, 04T6
DOKUMENTNAMN OCH -BETECKNING
DOCUMENT NAME AND REF.NO.

Dokumentnamn anger dokumentets typ och ändamål och då dokumentet har ett dokumentnamn utsätts detta (med versaler). Exempel: PM, RAPPORT, KOMPENDIUM, BESKRIVNING, INSTRUKTION, UTREDNING, ANBUD, STANDARD. Dokumentnamnet följs av en dokumentbeteckning som anger dokumentets plats i en serie, t ex rapportnummer, protokollnummer. Är dokumentet i första hand framställt för spridning utanför den egna organisationen bör beteckningen utformas så att utgivande institution kan identifieras med uppgift om enbart dokumentbeteckningen.

Exempel härpå är rapportnumrering enligt amerikansk nationell standard (ANSI Z39-23) och det av ISO tillämpade beteckningssystemet för dokument i olika utvecklingsskeden.

06T0
HANDLÄGGARE

Person som sakbearbetar ärendet hos dokumentutgivaren.

06T4
UTGIVNINGSDATUM
DATE OF ISSUE

Dokumentbladet förses med dokumentets utgivningsdatum. Datum skall anges enligt SIS 01 02 11, dvs numeriskt i ordningen år, månad, dag (exempel 1975-07-03) eller alfanumeriskt med de tre första bokstäverna av månadens namn (exempel Aug 1975).

06T6
ÄRENDEBETECKNING
PROJECT NO. ETC

Exempel på ärendebeteckning är diarienummer, projektnummer, kontonummer, kort sakord eller annan beteckning som hänför sig till det i dokumentet behandlade ärendet ("sak").

08T0
FÖRFATTARE
AUTHOR(S)/CORPORATE AUTHORS

I första hand anges dokumentets författare. Vid kollektivt författarskap (t ex kommitté) anges för dokumentet ansvarigt organ. Även projektledare, redaktör eller annan "handläggare" kan anges. Namn bör anges i följden (1) hela förnamn (2) efternamn. Titel erfordras ej.

10T0
(RESERV)

Fältet (motsv v2 i standardiserad brevblankett) kan utnyttjas då flera författarnamn förekommer. Fältet kan även användas för namn och adress på mottagare av dokumentdatabladet, t ex dokumentationscentral.

10T4
ANSLAGSGIVARE
SPONSORING ORGANIZATION

Fältet (motsv h2 i standardiserad brevblankett) bör reserveras för namn och postadress på anslagsgivare eller annan institution som i egenskap av finansierande organ (för ett projekt) åberopas i dokumentet.

18T0
DOKUMENTTITEL OCH UNDERTITEL
DOCUMENT TITLE AND SUBTITLE

Fältet används för dokumentets titel och (ev) undertitel. Antalet bilagor anges inom parentes efter den första titeluppgiften. Titeluppgifterna får ej förkortas eller på annat sätt förändras, men de kan kompletteras med exempelvis översättning till engelska.

26T0
REFERAT (SAMMANDRAG)
ABSTRACT

Texten i fältet skall bestå av ett sammandrag av dokumentets innehåll. Nyckelord skall understrykas.

Det bör uppmärksammas att sammandraget bör kunna återges i reproduktion på kort i format A6L eller i förminskning. Detta motiverar att referatet skrivs med början i tabuleringsläget T2. (Vid början i TO iakttas en radlängd av högst 125 mm).

Anm: ISO 214 - Documentation - Abstracts ger ledning vid utformning och redigering av referat.

42T0
REFERAT (SAMMANDRAG)
SKRIVET AV
ABSTRACT WRITTEN BY

Det förutsätts att sammandraget i första hand skrivs av dokumentets författare. Härvid anges "förf" i rutan. Om annan än författaren är ansvarig för referatet (sammandraget) kan initial och efternamn eller enbart initialer anges.

44T0
FÖRSLAG TILL (YTTERLIGARE)
NYCKELORD
KEY WORDS

Med nyckelord avses ord som författaren finner bäst karaktäriserar innehållet i det refererade dokumentet. Antalet nyckelord bör vara minst 5 och högst 15.

55T0
KLASSIFIKATIONSSYSTEM
OCH KLASS(ER)
CLASSIFICATION SYSTEM
AND CLASS(ES)

I fältet anges beteckningar för dokumentets ämnesinnehåll enligt något klassifikationssystem.

52T0
INDEXTERMER
INDEX TERMS

Härmed avses innehållsbeskrivande termer om de hämtats från en kontrollerande vokabulär (tesaurus). Indextermerna kan avvika från nyckelorden (jfr 44T0) som kan fritt väljas. Den tesaurus eller motsv från vilken indextermerna har hämtats skall anges efter sista indexterminen.

56T2
ÖVRIGA BIBLIOGRAFISKA UPPGIFTER

Uppgifterna kan avse impressum (förlagsort, förlag, utgivningstid, tryckeri etc), upplaga, serietillhörighet och liknande.

56T0
OMFÅNG
NUMBER OF PAGES

Uppgiften anges med antal sidor i det refererade dokumentet inkl bilagor, förekommande (särpaginerade) bilagor.

58T0
SPRÅK
LANGUAGE

Dokumentets språk i det fall det är annat än svenska.

60T0
SEKRETESSUPPGIFTER
SECURITY CLASSIFICATION

Uppgift om begränsad tillgänglighet hos dokumentet med hänsyn till sekretess och andra av myndighet (genom lagstiftning eller på annat sätt) eller av dokumentutgivaren gjorda inskränkningar.

60T4
ISSN

International Standard Serial Number. Anges endast för serier åsatt ISSN.

60T6
ISBN

International Standard Book Number. Anges endast för dokument med åsatt ISBN.

62T0
DOKUMENTET KAN ERHÅLLAS FRÅN
DISTRIBUTION BY

Namn och (om möjligt) postadress på distributör av det refererade dokumentet.

62T4
MOTTAGARENS UPPGIFTER
RECIPIENT'S NOTES

Fält som kan disponeras av mottagaren.

66T0
PRIS

Priset anges i svenska kronor såvida ej dokumentets spridning motiverar annan valuta.

För att möjliggöra maskinell läsning av text i dokumentdatabladet har teckensnitt ORC-B använts vid ifyllning av förlageblanketten. Teckensnittet behandlas i SIS 66 22 42 (under arbete 1976).

Contents

1. INTRODUCTION

The introduction of the computer has meant a dramatic change not only in business management and society in general but also in the scientific field. A numeric solution has become not only acceptable but as important as an analytic one. In other words, what should be considered as a feasible method has changed, which in turn has greatly influenced the development of the theoretical advances in e.g. automatic control. Some new problem areas have in the same time emerged; one is how the computer should be applied in problem solving, which is the scope of this report.

The task of using a computer to solve a problem consists of two parts: one is to supply the computer with relevant data, to specify constants, parameters, and maybe structural details on the problem, the other being to do the same for the solution method.

Describing the solution method could for instance mean to write a program to do the job. A first step in facilitating this operation would be to provide a subroutine library from which the problem solver can take some ready-made routines as "primitives" to build his own program. This approach will give the user great freedom to design his own way to solve a specific problem but leaves him with the heavy burden of implementing a main program with control of the logic flow and I/O of problem-related data.

Ready-made programs

The second step would be to provide a ready-made program containing facilities applicable to a certain problem field. The user (problem-solver) would then have to interact with the program in some way so as to achieve a solution to his

specific problem. The user might have different requirements on the interaction viz.

- The batch user
- The experienced user
- The one-time user
- The beginner
- The assistant

The batch user can (and must) select in advance a sequence of actions that the program is going to follow with a preselected set of input data.

The experienced user, on the other hand, might be trying to solve a new and complicated problem exercising his combined prior knowledge, skill, intuition, and common sense. In this situation he wants a maximum of freedom in the choice of solution steps, and it is of great importance to be able to communicate directly with the computer. It is then possible to view the result as they become available and to direct the future steps accordingly.

The one-time user could be a student solving a laboratory exercise, e.g. in a course on automatic control. The task would then be to use a program with a fairly rigid structure solving a well-defined problem. Being a one-time user, he would not be interested in anything but the elementary facilities needed for his task and simplicity is a main consideration.

The beginner is initially in the same situation as the one-time user; simplicity is of importance to be able to get started. The beginner has a desire to become an advanced user some day, though. What he wants then is a facility for rapid help and instruction.

The assistant finally is someone who either does not know the fine details or is using them routinely as a part of an investigation designed by the experienced user.

Naturally, the ideal type of interaction is quite different for these users, but it is most important to realize that the same program may have to meet these varying demands. For instance, the student during his exercise may get stuck and call the help of his supervisor. The supervisor, being an experienced user, would then prefer a more concise means of interaction than would the student.

It is also worth noting that the experienced user, after a few months disuse of a program may have forgotten some details of its use, and thus for a while may be regarded as a fast-learning beginner. Also, the experienced user may be his own assistent, that is he does the routine work himself.

Programming philosophies

A natural and often used way of interaction is the question and answer method. Here the program puts questions regarding parameter values and other information on the problem or its solution, which are answered by the user. He has also the possibility to answer questions regarding the future operation of the program, i.e. choose from a list of available program modules which one should be run next.

This method allows good guidance to the one-time user together with a possibility of good security as the answers may be tested in their context. On the other hand, the experienced user may feel himself locked into a winding trail although he knows a more direct way to the goal.

Another way of interaction is by means of commands. The user condenses his wishes into a command line, containing keywords, flags, and values to indicate a specific action to be taken. This is a much more concise method and leaves the experienced user with all possible freedom. The one-time user, however, is left on his own.

Intrac

Intrac is a subroutine package designed so that it can be easily combined with application modules to form an interactive program. Intrac itself contains no application dependent features so it can be used in any application field.

An interactive program built around Intrac will in principle have the structure shown in Fig.1.

There is a main program module which calls upon a number of subroutines, here called action routines, and upon Intrac. When Intrac is called it will (normally) read a command line from the user's terminal and analyse it. The number of the recieved command is computed from a command table sent to Intrac from the main module and the rest of the command line is processed and stored in memory. If the command recieved is an application command, Intrac returns to the main module which in its turn uses the command number to select the proper action routine to call. Thus each application command generally corresponds to a specific action routine. It is possible for the selected action routine to call Intrac too, in order to recieve commands to further specify the required action. Such subcommands have to be fully implemented within the corresponding action routine.

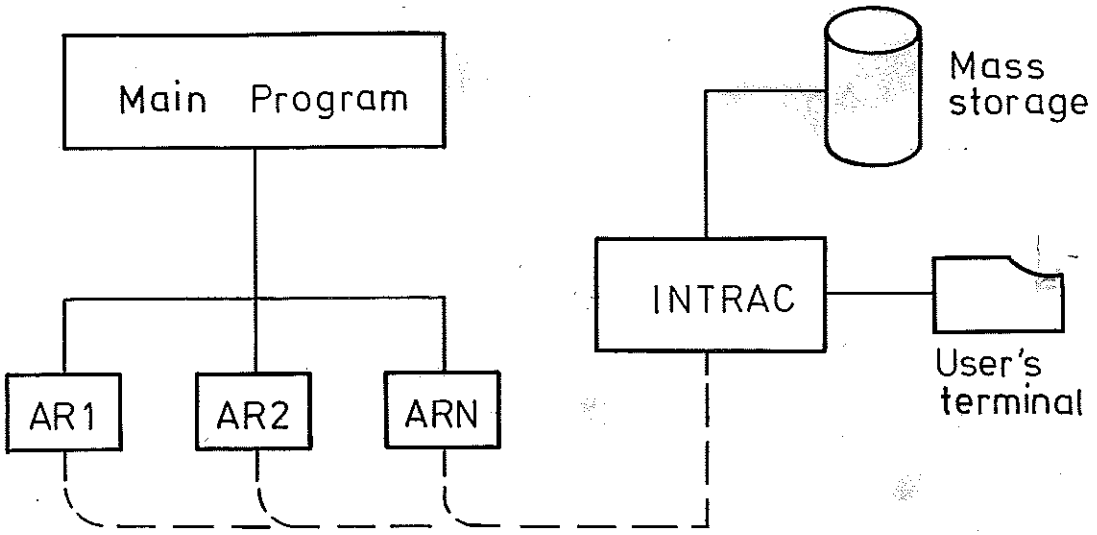

Fig. 1

Intrac has the possibility to read the commands off mass memory, this is the case when a macro is to be executed. Then a file of stored commands are read and acted upon, to the user looking as if a single command was executed. Also Intrac will recognize a number of general purpose commands and process them within itself, thus they are never seen by the main program module nor by application routines in the subcommand case.

Intrac is basically command oriented but contains features, viz. the macro facility and the general purpose commands, that allow e.g. help functions, batch operation, question and answer dialogue etc. to be easily implemented. This

will be further explored and exemplified in later chapters. As we shall see, it may be fruitful to look at Intrac as a means of implementing a problem-solving language, geared by the set of action routines towards a specific application area. Chapter 2 will give an introduction to this idea using a fictitious application; linear algebra. Chapter 3 will then give a description of the elements of the Intrac language while Chapter 4 contains specifications on the available general purpose commands. Chapter 5 finally will give more extensive examples showing the various forms the interaction will take, depending on which use is made of the available facilities. A summary of notation, syntax and facilities is given in the appendix.

How to construct an interactive program package based upon Intrac is discussed in detail in Schonthal (1977) *). The following problems are treated:

- Designing application routines and interfacing them to Intrac.
- Calling and initializing Intrac.
- How to use Intrac's auxiliary routines for argument decoding and data exchange between Intrac and application routines.
- Display handling.

*) Tomas Schönthal: INTRAC - Programmer's Guide, Department of Automatic Control, Lund Institute of Technology, Lund 1977, TFRT-7128.

2. INTRODUCTION TO THE INTRAC LANGUAGE

In an application program, the programmer has built in a (hopefully) natural structure in the available command line formats. Intrac itself implements some commands of an application independent nature, see Chapter 4. Together, these form statements in a problem solving language, designed for a specific application field.

In the normal or command mode of operation, actual statements, or commands, are entered from the keyboard of the users terminal. Intrac also provides the possibility of stringing a sequence of commands together to form a macro command. The command string is stored on mass memory.

This macro facility corresponds to what in ordinary programming languages is called subroutines or procedures. It is by such procedures in the command language the various dialogue types mentioned in Chapter 1 can be constructed. This is discussed to some length in Chapter 5.

The Intrac language will be introduced by means of an example. For that reason a completely fictitious application program is defined. The treatment will be concise for the benefit of the computer language accustomed reader. The beginner will find notations and ideas explained in later chapters.

The application is a program to solve problems in linear algebra. The following commands are available:

MATRIX <matrix identifier> (<dimension 1><dimension 2>)

Declares a matrix. The elements are assigned values via subcommands:

ROW (<row index>) {<number>}*

Undefined rows are zeroed. (Note that { }* means repetition one or more times).

EXIT

Returns to the main command level.

LETME <matrix identifier> (<index 1><index 2>) = <value>

Assigns a new value to a matrix element.

PRINT <matrix identifier>

Outputs a matrix to the terminal.

MATOP <matrix identifier> = <matrix expression>

Evaluates matrix expressions. The operands can be scalars, vectors, matrices. The operands are + - * **. Parantheses can be used in the usual way. The inverse of a matrix is e.g. written A**-1.

EIGEN <matrix of eigenvalues><matrix of eigenvectors> = <matrix identifier>

Computes eigenvalues and eigenvectors of a square matrix. The matrix of eigenvalues is an nx2 matrix where the first column contains the real parts and the second column contains the imaginary parts. The matrix of eigenvectors is an nxn matrix.

Allowing comments following the character " we have the following commented example:

```
>MATRIX A (2 2)          "Define a 2x2 matrix A.
  >ROW (1) -6 -5         "Use subcommands
  >ROW (2)  1  Ø         "to input elements.
  >EXIT                  "Return to main level.
>EIGEN L X = A           "Eigenvalues of A into L.
>PRINT L                 "Print eigenvalues.
  -5  Ø
  -1  Ø
```

Dialogue 1

Now assume that we want to examine the change in eigenvalues when we change elements in A. This could be done by issuing the command sequence LETME, EIGEN, and PRINT a number of times.

If this is to be done frequently, it may pay off to define a macro-command as shown in Dialogue 2. Note the use of formal arguments

```
>MACRO EIGALTER MAT I J V "Definition of a macro.
  >LETME MAT(I J) = V     "Change element.
  >EIGEN L X = MAT
  >PRINT L
  >END
>EIGALTER A 1 2 -9        "Use the macro.
  -3   Ø
  -3   Ø
>EIGALTER A 1 2 -13       "Use again.
  -3   2
  -3  -2
```

Dialogue 2

As already has been hinted at, Intrac contains commands to control, among other things, the execution of a macro. Dialogue 3 shows a macro with a FOR-NEXT loop automating the regular change in a matrix element. Note that this macro calls the macro EIGALTER from Dialogue 2. Note also the variable length argument list; if the formal argument PVEC is given the value EIGVEC the eigenvectors are printed as well as the eigenvalues.

```
>MACRO EIGITER KMIN KMAX KSTEP ; PVEC
  >FOR K=KMIN TO KMAX STEP KSTEP
  >LET A12 = -5-K
  >WRITE 'Parameter K= ' K
  >WRITE 'Eigenvalues:'
  >EIGALTER A 1 2 A12
  >IF PVEC NE EIGVEC GOTO L
  >WRITE 'Eigenvectors:'
  >PRINT X
  >LABEL L
  >NEXT K
  >END
>
```

Dialogue 3

The result of using this macro is shown in Dialogue 4. To further illustrate the execution of a macro the switches ECHO and TRACE have been used which cause a printout of the commands as they are executed.

```
>EIGITER Ø 8 4
  Parameter K=Ø
  Eigenvalues:
  -5    Ø
  -1    Ø
  Parameter K=4
  Eigenvalues:
  -3    Ø
  -3    Ø
  Parameter K=8
  Eigenvalues:
  -3    2
  -3   -2
>
>SWITCH ECHO ON              " Now, ECHO ON
>EIGITER Ø 3 3 EIGVEC
  Parameter K=Ø
  Eigenvalues:
  <LETME A(1 2) = -5
  <EIGEN L X = A
  <PRINT L
  -5   Ø
  -1   Ø
  Eigenvectors:
  <PRINT X
   0.98058  -0.70711
  -0.19612   0.70711
  Parameter K=3
  Eigenvalues:
  <LETME A(1 2) = -8
  <EIGEN L X = A
  <PRINT L
  -4    0
  -2    0
```

```
  Eigenvectors:
  <PRINT X
  -0.97014    0.89443
   0.24254   -0.44721
>SWITCH TRACE ON            " Now, TRACE ON
>EIGITER Ø 1 2
  <MACRO EIGITER KMIN KMAX KSTEP ; PVEC
  <FOR K=KMIN TO KMAX STEP KSTEP
  <LET A12 = -5-K
  <WRITE 'Parameter K= '  K
  Parameter K=Ø
  <WRITE 'Eigenvalues:'
  Eigenvalues:
  <EIGALTER A 1 2 -5
    <MACRO EIGALTER MAT I J V
    <LETME A(1 2) = -5
    <EIGEN L X = A
    <PRINT L
  -5   Ø
  -1   Ø
    <END
  <IF PVEC NE EIGVEC GOTO L
  <LABEL L
  <NEXT K
  <END
>
```

Dialogue 4

3. CONCEPTS OF THE INTRAC LANGUAGE

The user of an interactive program based on Intrac interacts with the program via a terminal and expresses wishes concerning the solution of his problem in the form of commands or answers to questions.

The commands can be divided into different categories. Some general purpose commands (or Intrac statements) are handled by Intrac itself, while others, application commands, are analyzed by Intrac and then passed on to the main program module which selects the appropriate action routine to handle them. In some cases the action routines may need or offer further interaction in order to carry out the desired actions. This is then accomplished by means of subcommands, i.e. commands received through Intrac but with a different command table, depending on the specific action routine.

Another facility supported by Intrac is the use of macro commands, i.e. calls to previously defined command sequences on mass memory. Technically, when Intrac recognizes a reference to such a command sequence, it starts reading commands from a mass memory file, rather than from the user's terminal. A macro corresponds to subroutines or procedures in ordinary programming languages.

Actually, a program built around Intrac may be regarded as an interpreter for an interactive problem solving language with the same type of facilities as found in many other languages for interactive programming. An important difference is, however, that here we are aiming towards a specific problem area, by the inclusion of special problem oriented action routines / commands. Macros (subroutines / procedures) in this problem oriented language can be used to implement common subproblem solutions, give user guidance or implement question / answer dialogue. In this way many of

the demands mentioned in Chapter 1 can be met.

A detailed discussion on the elements of the Intrac language will now follow. Appendix 1 describes the syntax notation used.

3.1 Commands

A command has the generic form:

<command identifier><argument list>

There are three types of commands: Intrac-statements, application commands and macro calls. The legality of commands depends on the context in which they are called, as described later.

Following the <command identifier> are the arguments of the command (if any), which convey information on the problem or its solution.

<argument list>::=[<argument>]*

where

<argument>::=<integer>/<real>/<identifier>/<delimiter>/<variable>

Spaces are not allowed within an <argument>. One or more spaces must separate <argument>s if ambiguity otherwise would arise.

The character " is interpreted as a line terminator for Intrac. Thus the characters following a " are never scanned and can thus be used as comments. Empty lines and hence

lines only containing a " followed by a comment are legal.

<integer> and <real> arguments need no further comments, they pass numbers to the action routines. <delimiter>s are used to give structure to the command line. This can be used to separate an input side from an output side in an argument list, to enclose flags within parentheses etc. The following line illustrates this:

```
>ML (SC) MODEL = DATA 2
```

To summarize, <delimiter>s are used to make the argument list easy to decode unambiguously, to allow arguments to be optional, to allow precise error diagnostics, and to make the command format natural and easy to memorize, i.e. as "syntactic sugar".

An <identifier> may be used in several ways, one is to signify a <variable>. This is treated in section 3.4. The other is as a literal, i.e. it represents itself as a character string. This string can be used as a file name, the name of an internal data area, as a flag etc., depending on the application routine.

In the example above, MODEL and DATA are file names, while SC is a flag indicating that the user wants to use subcommands.

3.2 Macros

A macro consists of a sequence of Intrac-statements, macro calls and application commands. They are stored on a text file on mass storage.

The first line in the macro should be a MACRO-statement which has the following form.

MACRO <macro identifier> [<formal argument>/<delimiter>/<termination marker>]*

The <macro identifier> has to be the same as the file name. The statement declares the formal arguments of the macro. The use of <delimiter> and the <termination marker> is explained later.

The list of formal arguments can be extended by FORMAL-statements. They must follow immediately after the MACRO-statement and have the form:

FORMAL {<formal argument>/<delimiter>/<termination marker>}*

After the MACRO-statement and optional FORMAL-statements follows a sequence of Intrac-statements, macro calls and application commands. The last line in the macro should contain an END-statement:

END

Example

Three examples of macros are given below.

```
MACRO MAC1 ALPHA BETA
...
...
END
```

```
MACRO MAC2 A1, A2, A3
FORMAL / A4, A5
FORMAL (A6)
...
...
END


MACRO MAC3 A1=A2 ; B1=B2 ; C1=C2
...
...
END
```

The MACRO-, FORMAL- and END-statements are not only used in the macros. They are also used from the terminal to generate new macros. This is described in Section 4.1.

Macro call

A macro is called by giving its name followed by actual arguments in the same way as a command.

If the <termination marker> is not used then the number of actual arguments should be equal to the total number of formal arguments in the MACRO- and FORMAL-statements. The delimiters appearing among the formal arguments should be given at corresponding positions in the call.

The <termination marker> is used when a variable number of actual arguments should be allowed.

<termination marker> ;:=/;

It indicates that the formal arguments and delimiters appearing following the symbol need not have corresponding actual arguments. If the <termination marker> is used

several times in the macro then it gives alternative places where the call can be terminated. The formal arguments which have no corresponding actual arguments will be 'unassigned', see Section 3.4.

Example

Alternative calls to the macros in the previous example are given below.

```
>MAC1 10.5 2
>MAC1 DATA1 DATA2

>MAC2 1,2,3 /4,5 (6)

>MAC3 P1=5 P2=7 P3=9
>MAC3 INDATA=FILE1
```

3.3 Command modes

Intrac reads commands either from the keyboard on the user's terminal or from a macro, i.e. a file. When a command is awaited from the terminal a prompting character (e.g. >) is output. Three special cases of terminal input can be recognized. The program starts in direct mode. When the MACRO-statement is entered the program goes to generation mode. The subsequently entered commands will be stored on a macro file as described in section 4.1. The execution of a macro is temporarily interrupted, i.e. suspended, e.g. if an erroneous command or the SUSPEND statement is encountered. In such a case Intrac enters suspended mode and accepts commands from the terminal. These commands are interpreted in the environment of the suspended macro. The execution of the macro is resumed by the RESUME-statement or by a GOTO to some label in the macro.

The application commands may be arranged hierarchically. Some commands may require a more detailed specification than the one given in the command line. This may be so because the command line otherwise would be impractically long, or that some arguments are needed only in special cases or that the proper setting of parameters is apparent only after an initial examination.

In such cases, a subcommand sequence is entered. This is indicated by the prompting character appearing indented. In a subcommand sequence, Intrac commands are still available. Available application commands, however, are now a function of the "parent command" within which they are implemented. There is always a subcommand to leave the subcommand mode.

Table 1 shows how the validity of the Intrac statements, the macro calls and the application commands depend on the command mode.

Table 1. Command modes and validity.

	direct	macro	generation	suspended
MACRO	X	X		X
FORMAL		X	X	
END		X	X	X
LET	X	X	X	X
DEFAULT	X	X	X	X
LABEL		X	X[1)]	
GOTO		X	X[1)]	X
IF		X	X[1)]	X
FOR		X	X	
NEXT		X	X[1)]	
WRITE	X	X	X	X
READ	X	X	X	X
SUSPEND		X	X[1)]	
RESUME				X
SWITCH	X	X	X	X
FREE	X	X	X	X
STOP	X			
Macro call	X	X	X[1)]	X
Application command	X	X	X	X

1) Legal but not executed

3.4 Variables

A variable can be of three different types:

<variable>::=<formal argument>/<local variable>/
<global variable>

where

<formal argument>::=<identifier>
<local variable>::=<identifier>
<global variable>::=<identifier>.[<identifier>]

The value of a variable can be either an integer number, a real number, an identifier or a delimiter. A variable can also be unassigned.

During the processing of the argument list of a command, Intrac will for every occurance of an <identifier> check if it represents a <variable>, by looking through its internal tables of the three <variable> types. If not found it is treated as a literal.

If, on the other hand, Intrac finds it to be a <variable> of any of the types above, the <identifier> in the argument list is substituted by Intrac to its actual value according to the internal tables maintained by Intrac.

The substitution rule does not always apply to Intrac-statements. The items which can be substituted are underlined in the syntax for the Intrac-statements.

Formal arguments

Formal arguments exist when a macro is being executed and are those arguments listed in the definition of the macro, cf. the commands MACRO and FORMAL.

When the macro is called, a corresponding list of actual argument values should be specified. The substitution rule is then applied so that every occurence of a formal argument in the macro is replaced by its value before the command arguments are passed on to the proper routine.

The value of a formal argument may be altered inside the macro. Note though, that no value can be returned from the macro to the caller using formal arguments. This corresponds to call by value in modern programming languages.

Local variables

A local variable has the same form as a formal argument and it is in fact treated in very much the same fashion. It is local to the macro level and is defined when it is first given a value in a READ, FOR, LET, or DEFAULT statement.

Formal arguments and local variables can only be referenced in the macro where they are declared. When a macro is left via the END statement then all its formal arguments and local variables become unaccessible.

Formal arguments and local variables can have the same identifiers in different macros.

The user has to be careful when choosing identifiers for local variables and formal arguments so that they do not coincide with identifiers which are values in commands.

Global variables

A global variable is distinguished by a dot following the identifier.

A <global variable> is always accessible and may pass information between macros. An important use is to define a set of problem dependent parameters stored as <global variables> that can be referenced in several different but

related application commands or macros.

The value of <global variable>s may also be used and returned directly by application command routines. In fact, <global variable>s are the only means by which results may be returned from application commands, within the framework of Intrac. Other possible ways, files, data areas, etc. are not administered by Intrac.

Finally, it should be mentioned, that the implementor of an interactive program package has the possibility to initialize the table of <global variable>s so that there always will be a set of "reserved" <global variables> for special use. Nonreserved global variables can be deallocated by the statement FREE.

Unassigned variables

Under certain circumstances, a global or local variable, or a formal argument may have been defined without having been assigned a value. This may be the result of using the <termination marker> in a macro definition or a READ-command, or when suspending a macro in a READ-command.

The type 'unassigned' may be transferred in a LET-command. The action on 'unassigned' values by IF ... GOTO commands is defined, and most important, the DEFAULT command is specifically designed to handle them.

If an unassigned variable appears as an argument to an application command, it will be totally invisible to the corresponding routine.

Example

Assume that the variables A and B have the following values.

A = 2.5, B = 'unassigned'

The command

```
>COM A B 5.5
```

will then be equivalent to

```
>COM 2.5 5.5
```

Unassigned variables together with the use of the <termination marker> is important when constructing macros for simplified interaction, cf. Chapter 5.

4. THE STATEMENTS OF INTRAC

This chapter contains the detailed description of the statements in Intrac.

4.1 Generation of macros (MACRO, FORMAL, END)

There are some different ways to generate a macro. Since a macro is implemented as a text file it is possible to generate and modify a macro using a text editor.

A macro can also be generated by entering the MACRO-statement from the terminal. In generation mode all correct commands entered from the terminal are stored on a file. This continues until generation mode is left by the END-statement. Whether the commands in the macro should be executed during generation or not is determined by the switch EXEC (see statement SWITCH). If EXEC is OFF then the commands are only checked for formal errors and if correct stored on the file. If EXEC is ON the commands will also be executed. There are some exceptions to this, such as the GOTO- and IF-statements (see Table 1). These statements are only checked and stored. If EXEC is ON, actual values for the formal arguments of the macro are requested, with the value promter (#), when the MACRO- and FORMAL-statements are encountered. The rules on delimiters and the <termination marker> used for macro calls also applies in this case.

The FORMAL-statement can be used to extend the list of formal arguments anywhere in the macro. It is placed after the MACRO-statement automatically when the generation is finished.

Example

```
>SWITCH EXEC OFF
>MACRO MAC A B
  >...
  >FORMAL C
  >...
  >END
>
>SWITCH EXEC ON
>MACRO MAC A B
  #5 7.9
  >...
  >FORMAL C
  #ALPHA
  >...
  >END
>
```

4.2 Assigment of variables (LET, DEFAULT)

Formal arguments are allocated and possibly assigned when a macro is entered. Their values can be changed with the LET-, DEFAULT-, FOR-, and READ-statements.

Global variables and local variables are allocated and assigned (or changed) using the LET-, DEFAULT-, FOR-, and READ-statements.

The LET-statement has the following forms:

```
LET {<variable>=}*{<number>[{+/-/*//}<number>]
LET {<variable>=}*{+/-}<number>
LET {<variable>=}*<identifier>[+<integer>]
```

LET {<variable>=}*<delimiter>
LET {<variable>=}*<unassigned variable>

The two first forms is the usual arithmetic assignment. Only one or two operands can be given.

The third form contains a mixed mode operator + which denotes string concatenation. The second operand, the integer, is converted to the corresponding string of characters and appended to the first operand giving a new identifier.

Examples:

statement:	result:
LET A = B = Ø	A = B = Ø (integer)
LET P = 3*5.5	P = 16.5
LET Gl. = 2+P	Gl. = 18.5
LET DATA = FILE1	DATA = FILE1 (identifier) if FILE1 is not a variable
LET NUMBER.DATA = 1ØØ	NUMBER.DATA = 1ØØ
LET I = 5	I = 5
LET ID = FILE+I	ID = FILE5 (identifier) if FILE is not a variable

The DEFAULT-statement is a conditional assignment statement. Its form is:

DEFAULT {<variable>=}* <argument>

The assignment is performed only if either
- the named variable is 'unassigned'
- the named variable does not exist.

In the last case a new variable is allocated.

The DEFAULT statement is useful in connection with the <termination marker> as it gives rise to 'unassigned' variables.

Examples:

Assume the following variables are accessible.

```
A = 5
G. = DATA
B = 'unassigned'
```

statement:	effect:
DEFAULT A = Ø	none
DEFAULT B = INFILE	B = INFILE
DEFAULT C = 7	C = 7 (new variable)
DEFAULT G. = 1	none

4.3 Branching (LABEL, GOTO, IF)

To make macros flexible it is necessary to have a way to change the sequence of commands executed. This is possible by using simple unconditional and conditional branch statements.

The labels used in branch statements are declared at the position which they indicate using the LABEL-statement:

```
LABEL <label identifier>

<label identifier>::=<identifier>/<integer>
```

The label identifier is locally defined in a macro. Note that there is no check that a label is uniquely defined in a macro. If a label is multiply defined then a GOTO-statement will use the first label found when searching the file from the top.

Examples:

```
LABEL SKIP
LABEL 3
```

The unconditional GOTO-statement is:

GOTO <label identifier>

The next command executed will be the command after the corresponding LABEL-statement. If no LABEL-statement with the same label identifier is found then the command after the GOTO-statement will be the next to execute.

It is allowed to jump out of a FOR-NEXT loop but not into one.

Since the argument in the GOTO-statement could be a variable whose value is a label identifier it is possible to use the statement as the assigned GOTO of FORTRAN.

Examples:

Assume LAB = SKIP, L. = NOLAB, I = 3

statement:	effect:
GOTO SKIP	GOTO SKIP

```
GOTO LAB          GOTO SKIP
GOTO L.           GOTO NOLAB
                     execute next statement if
                     NOLAB is not a label
GOTO I            GOTO 3
```

The conditional GOTO statement has the form:

IF <argument> {EQ/NE/GE/LE/GT/LT} <argument>
GOTO <label identifier>

The effect of this statement is the same as for the GOTO-statement if the relation is true. If it is false the next command in sequence is executed.

The relational operators EQ, NE, GE, LE, GT, and LT means equal, not equal, greater or equal, less or equal, greater than, and less than.

Mixed mode relations are defined as follows. If one value is real and the other is integer the test is done using real arithmetic. If one of the operands is an unassigned variable then the EQ-relation (NE-relation) will be true if and only if the other operand is (is not) an unassigned variable. Mixed mode between numeric and non-numeric variables is illegal.

Example:

Assume A = 5 B. = DATA C = 8.5 D = unassigned
UNASS. = unassigned.

statement:	effect:
IF A GT 2.5 GOTO 1	GOTO 1
IF B. EQ FILE GOTO 2	none
IF A NE UNASS. GOTO 4	GOTO 4
IF D EQ UNASS. GOTO 5	GOTO 5

The GOTO and IF statements are legal in suspended mode. If the label exists (and the condition is true) the macro is resumed and the execution is continued after the label.

4.4 Looping (FOR, NEXT)

It is possible to introduce loops among the commands in a macro. This is done with the FOR- and NEXT-statements. The FOR-statement begins the loop and has the following form:

FOR <variable> = <number> TO <number> [STEP <number>]

The NEXT-statement ends the loop and has the form:

NEXT <variable>

The exact behaviour of the loop-statements are properly described using the branch statements. This is done below:

The statements

```
FOR V = BEGIN TO FINISH STEP INCR

commands

NEXT V
```

are equivalent to

```
LET V = BEGIN
LABEL TEST
IF INCR EQ Ø GOTO OUT
IF INCR LT Ø GOTO NEG
IF V GT FINISH GOTO OUT
GOTO RUN
LABEL NEG
IF V LT FINISH GOTO OUT
LABEL RUN

commands

LET V = V+INCR
GOTO TEST
LABEL OUT
```

If no increment is specified, i.e. STEP <number> is omitted, the increment one (integer) is assumed.

If the variable is previously defined it must be of type integer or real. If the variable is of integer type while all numbers in the FOR-statement are real, then real arithmetic is used followed by an integer assignment.

If the variable is not previously defined then a new variable is allocated with type integer if all numbers in the FOR-statement are integers, otherwise the type will be real.

Loops may be nested to a maximum of five levels. It is allowed to jump (GOTO, IF) out of a loop but not into a loop. A loop entered in generation mode with EXEC ON will execute once with the loop variable equal to BEGIN regardless of the value of FINISH.

Examples:

```
FOR I = 1 TO 1Ø
...
NEXT I

FOR P = Ø TO PEND STEP Ø.1
...
NEXT P

FOR G. = 5 TO -1.5 STEP -1.5
...
NEXT G.
```

4.5 Output and input (WRITE, READ)

The macro facility can be used to implement question and answer interactive programs.

Questions are written on the terminal with the WRITE-statement and the answers are read using the READ-statement.

The WRITE-statement is used to write variables and text strings or to display all available variables. Its form is

WRITE [([DIS/TP/LP] [FF/LF])] [<variable>/<string>]*

When no variable or string is given in the WRITE-statement, all currently accessible variables are displayed on the output device (see example below). This is particularly useful when debugging macros.

A <string> is any string of characters except ' enclosed by two '.

The output device is specified as

DIS - display
TP - terminal printer
LP - lineprinter

The default output device is display. The actual output device is of course installation dependent.

The FF (form feed) and LF (line feed) option specifies if the output should be made on a new page or just on the next line. If this option is omitted and the command does not contain any variables or strings (see below) a form feed is made before the output else the output will be made on the next line.

Examples:

Assume that the following WRITE-statements are given in a macro MAC at macro level 2 or from the terminal with this macro suspended. Also assume that the variables shown below have been defined. The WRITE-statements and the corresponding printouts are shown.

```
>WRITE

        RESERVED GLOBAL VARIABLES
A    .      = 5        PAR  .SYST   = 5.5E-3

        GLOBAL VARIABLES
G1   .      = DATA1  DATA .SIM    = FILE1
G2   .      = 7
```

```
        LOCAL VARIABLES IN MAC AT LEVEL 2
A1    = 13.5   INDATA = FILE2  I      = 1

>WRITE 'What do you want to do?'

What do you want to do

>WRITE (LP) 'The input data was ' INDATA

The input data was FILE2

>WRITE 'G1. = ' G1. ' A1 = ' A1

G1. = DATA1 A1 = 13.5
```

The READ-statement reads values from the terminal and assigns variables. Its form is:

```
READ { {<variable> {INT/REAL/NUM/NAME/DELIM/YESNO}} /
     <termination marker>}*
```

After each variable a type specification for the expected value is given:

```
INT     - integer number
REAL    - real number
NUM     - integer or real number
NAME    - identifier
DELIM   - delimiter
YESNO   - identifier YES or NO
```

When the READ-statement is executed a prompting # is written on the terminal.

The <termination marker> has the same function as in the MACRO-statement. It gives alternative places where the answer could be cut off. The variables that are not given any value become 'unassigned'.

There are two means of escape from the READ statement, resulting in the suspending of the macro.

- If the answer is just a > the READ-statement will have no effect and the macro is suspended. If the macro is resumed by the statement RESUME the READ-statement will be re-executed.

- If an acceptable answer is given followed by a > the variables will be properly assigned and the macro suspended. If the macro is resumed with RESUME, the command following the READ will be executed.

Examples:

statement:	answer:	effect:
READ A NAME GL. NUM	#P1 17	A=P1, GL.=17
READ DATA1 NAME; DATA2 NAME	#FILE1	DATA1=FILE1, DATA2=unassigned
READ INDEX INT	#>	INDEX=unassigned macro suspended
READ INDEX INT	#3>	INDEX=3 macro suspended

4.6 Suspending a macro (SUSPEND, RESUME)

There are cases when the freedom to have formal arguments in a macro is not enough. Perhaps at some point in a macro it is not known at generation which commands that would be

appropriate. It is then possible to switch to command input from the terminal (i.e. suspend the macro). When the command input from the terminal is finished the macro is resumed. This facility is handled by the statements SUSPEND and RESUME as below:

```
Macro                Terminal

MACRO MAC
...
SUSPEND
                     >...
                     >...
                     >RESUME
...
END
```

The commands entered from the terminal are interpreted in the environment of the suspended macro. The variables in the suspended macro are thus accessible.

The execution of the macro can also be resumed by entering the GOTO or IF statements from the terminal. The user also has the possibility to deactivate all macros by giving the statement END from the terminal.

A macro is automatically suspended in some cases.

- When an error is detected during the execution of a macro then an error message is printed and the macro is suspended. The user can then e.g. enter a correct form of the erroneous command and then RESUME the macro.

- When the READ-command has been executed in a macro, the user has to input the requested values from the terminal, or he can enter a special escape character (>) which

causes the macro to be suspended.

- Depending on the implementation, there should be a way to externally interrupt the execution of a macro forcing it into suspended mode.

4.7 Switches in Intrac (SWITCH)

Intrac has some switches which are manipulated by the following command.

SWITCH {EXEC/ECHO/LOG/TRACE} {ON/OFF}

The switches have the following meaning:

EXEC Determines whether the commands entered in generation mode should be executed or not. See section 4.1.

ECHO If ECHO is ON, the commands in a macro are echoed on the terminal as they are executed. Echoed commands are preceded by a 'reversed promter', viz. <.

LOG Determines whether the executed commands should be logged on the line printer or not.

TRACE Affects the echoing and logging features. If TRACE is OFF only application commands are echoed and logged. Also Macro calls and Intrac statements are output if TRACE is ON.

All switches have the default value OFF.

4.8 Deallocation of global variables (FREE)

Nonreserved global variables can be deallocated by the command

FREE { {<global variable>}* / *.* }

The form

FREE *.*

is used to delete all nonreserved global variables.

4.9 Stopping the program (STOP)

The command

STOP

will stop the execution of the program containing Intrac.

5. SOME DIFFERENT FORMS OF INTERACTION

In the context of Chapter 1, the Intrac language is primarily suited for the needs of the experienced user, giving access with few restrictions to all available commands, in any order. As indicated there, this complete freedom may not always be desirable, so other forms of interaction should be provided. This is done by means of the macro facility, and this chapter will demonstrate the main ideas in how to obtain the desired result.

5.1 Commonly used command sequences

The experienced user will often find that a command sequence is frequently executed with only minor changes. By defining that sequence as a macro effectively generates a new special purpose command, suitable for a specific problem. This case is illustrated in Dialogues 2 & 3 in Chapter 2.

This type of macros may serve as a short hand facility for the experienced user, and as simple-to-use primitives for his assistant. In the examples, the macros were generated in the mode EXEC OFF. The mode EXEC ON is suitable when, during the solving of a problem, it is apparent that the following actions will be used more than once. By starting the definition of a macro during the first time through, the command sequence is then immediately available for repeated use.

5.2 Interaction for the infrequent user

A question and answer dialogue, giving good guidance for the one-time or infrequent user, might be realized in the way shown in the example on pages 46 - 49.

The READ and WRITE general purpose commands of Intrac are used to communicate with the user, who is taken in an orderly fashion from point to point so as to solve his problem. Some details are worth noting.

a) After the first run, the user is given a choice of possible next steps. This is done by SUSPENDing the macro. The user can then GOTO a previous point and restart from there.
b) At any READ point, the user can escape from the programmed sequence by using the 'escape symbol' (>). He will then enter command mode and can GOTO any point as in a).
c) By virtue of the <termination marker> in the READ command in the macro ENTER, the user need to type only the number of values consistent with the dimension of the matrix. 'Unassigned' arguments will be invisible to the ROW command.
d) If the user enters an empty line in answer to the READ in the macro ENTER, the argument M1 will be 'unassigned'. The test in the IF command against the global variable UNASS. then effects an exit from the subcommand sequence, leaving the rest of the matrix (the entire matrix, if used on the first row) unaltered. UNASS. is assumed to be a reserved global variable with the value 'unassigned'.
e) The position is continuously recorded in the global variable HELP. . At any command point as in a) or b), the user may call the macro HELP, which will use the

position information to output a message with hints at future actions.

f) There is a label with the name RESTART in both the main macro SOLVELIN and in the macro ENTER. Thus a command GOTO RESTART always makes good sense and will take the user to the restart point of the current macro, regardless of the macro nesting.

```
MACRO SOLVELIN
LET HELP. = HINIT
HELP
LABEL ORDER
WRITE 'ORDER: Number of equations:'
LET HELP. = HORDER
READ N INT
IF N LE 5 GOTO AMATRIX
WRITE 'Too many equations'
GOTO ORDER

LABEL AMATRIX
LET HELP. = HAMAT
WRITE 'AMATRIX: Enter matrix A'
ENTER A N N

LABEL BMATRIX
LET HELP. = HBMAT
WRITE 'BMATRIX: Enter matrix B'
ENTER B N 1

LABEL SOLVE
WRITE 'SOLVE: Solution of equations'
MATOP X = A**-1*B
PRINT X

LABEL RESTART
LET HELP. = HREST
WRITE 'You can go to ORDER, AMATRIX, BMATRIX, SOLVE,'
WRITE 'ALTER or OUT or write commands'
SUSPEND

LABEL ALTER
LET HELP. = HALTER
WRITE 'ALTER: Give matrix name, index 1, index 2 and value'
READ MAT NAME I1 INT I2 INT V NUM
LETME MAT (I1 I2) = V
GOTO RESTART

LABEL OUT
END
```

```
MACRO ENTER MAT N M
MATRIX MAT(N M)
FOR I = 1 TO N
LABEL RESTART
WRITE 'ROW' I
READ ;M1 NUM; M2 NUM; M3 NUM; M4 NUM; M5 NUM
IF M1 EQ UNASS. GOTO DONE
ROW (I) M1 M2 M3 M4 M5
NEXT I
LABEL DONE
EXIT
END


MACRO HELP
GOTO HELP.
LABEL HINIT
WRITE 'The program solves linear equations of the form'
WRITE '      A * X = B'
WRITE 'The matrices A and B are asked for and the solution'
WRITE 'X is printed. It is possible to change the matrices'
WRITE 'and recompute the solution. Help information can be'
WRITE 'obtained by calling the macro HELP.'
GOTO END

LABEL HORDER
WRITE 'Enter the dimension of the square matrix A and'
WRITE 'the vector B'
GOTO END

LABEL HAMAT
LABEL HBMAT
WRITE 'The matrix is entered as one row at a time'
GOTO END

LABEL HREST
WRITE ' '
WRITE 'Use the GOTO-statement for the commands'
WRITE 'MATRIX, LETME, PRINT of MATOP'
GOTO END

LABEL HALTER
WRITE 'You can change an element in A or B'
GOTO END

LABEL END
LET HELP. = HINIT
END
```

```
>SOLVELIN
The program solves linear equations of the form
    A * X = B
The matrices A and B are asked for and the solution
X is printed. It is possible to change the matrices
and recompute the solution. Help information can be
obtained by calling the macro HELP.
ORDER: Number of equations:
#3
AMATRIX: Enter matrix A
Row 1
#>
>HELP
The matrix is entered as one row at a time
>RESUME
#1 3 Ø
Row 2
#5 7 -2
Row 3
#Ø -1 2
BMATRIX: Enter matrix B
Row 1
#2
Row 2
#-3
Row 3
#5
SOLVE: Solution of equations
-Ø.666667
 Ø.888889
 2.944444

You can go to ORDER, AMATRIX, BMATRIX, SOLVE,
ALTER or OUT or write commands
>HELP
Use the GOTO-statement or the commands
MATRIX, LETME, PRINT and MATOP
>GOTO ALTER
ALTER: Give matrix name, index1, index2 and value
#A 2 2 Ø

You can go to ORDER, AMATRIX, BMATRIX, SOLVE,
ALTER or OUT or write commands
>PRINT A
 1       3       Ø
 5       Ø      -2
 Ø      -1       2
>GOTO SOLVE
SOLVE: Solution of equations
 Ø.5
 Ø.5
 2.75

You can go to ORDER, AMATRIX, BMATRIX, SOLVE,
ALTER or OUT or write commands
```

```
>GOTO AMATRIX
AMATRIX: Enter matrix A
Row 1
#-5 Ø
Too few columns
>GOTO RESTART
Row 1
#-5 Ø 5
Row 2
#-2 1 Ø
Row 3
#Ø 2 Ø
BMATRIX: Enter matrix B
Row 1
#
SOLVE: Solution of equations
 2.75
 2.5
 3.15

You can go to ORDER, AMATRIX, BMATRIX, SOLVE,
ALTER or OUT or write commands
>GOTO OUT
>
```

Example 1

5.3 Simplified command forms

Example 2 shows a method of implementing commands with two possible call formats. One form allows a single line call with arguments, while the other form consists of only the command name. The necessary arguments are then asked for, one by one. Finally, the proper action routine is called. This example demonstrates a possible implementation of the command EIGEN of Chapter 2. The actual computations are implemented in the action routine called by the command QREIG (eigenvalues by the Q-R method).

a) Note the use of the <termination marker> and the use of the 'unassigned' global variable, as in Example 1.
b) Note the use of a <delimiter> in the list of formal arguments. The rules state that the same delimiter must appear in the same position among the actual arguments.

```
MACRO EIGEN ; EVAL EVEC = A
IF EVAL NE UNASS. GOTO XCT
WRITE 'Name of eigenvalues?'
READ EVAL NAME
WRITE 'Name of eigenvectors?'
READ EVEC NAME
WRITE 'Name of matrix?'
READ A NAME
LABEL XCT
QREIG EVAL EVEC A
END
```

Example 2.

The command syntax for EIGEN as implemented by Example 2 thus looks like (cf Chapter 2):

EIGEN [<matrix of eigenvalues> <matrix of eigenvectors> =
<matrix identifier>]

5.4 Macros giving help and information

For the user with ambition to learn the possibilities of an interactive package in order to some day be an experienced user, facilities other than those of Example 1 are needed. Also, the experienced user may need occasional short advice, e.g. on a seldom used facility.

The macros given in Example 3 will offer some assistence in such a case. HELPSYN and HELPINF serve to write some informatory text, chosen through the argument. HELPSYN will give the syntax of the command in question, while HELPINF will give information on the nature and use of the different command arguments. HELPEX will ask questions to execute a command in a way similar to that of Example 2.

The operation of the macro HELP is as follows:

The beginner wanting to get to know the interactive program on his first session at the terminal types HELP as response to the promting character. The presentation and the information on the different modes of the help offered is then output. The mode will initially be 0. The beginner should keep this value the first time and will then get the menu, i.e. a list of all available commands, shown. He then indicates his interest for one of the application commands, and then, being in mode 0, receives information on the chosen command. In this way, a certain familiarity with the program is gained.

After a while, the user will feel ready to execute the commands. Specifying mode 1 in the section 'MODES' will cause HELP to execute the command chosen by the user in a question & answer type mode. Finally, the user will try his wings by writing the complete command with arguments. Mode 2 will still give some help in that the proper command syntax is displayed.

In this way, the beginner will recieve support according to his current state of training. Finally the user won't need the detailed help the HELP function gives. The subfunctions HELPSYN and HELPINF will however still be useful also for the experienced user, and can of course be used separately.

Some parts of the macro HELP are worth further comments:

a) The DEFAULT statement in the beginning of the macro gives the possibility of initialization the first time it is called.
b) The repeated use of WRITE statements to output large amounts of text is somewhat clumsy. In most implementations, there will probably be an application command available to output text files to a terminal. In such a case, presentation text, the menu and similar information would be stored on files separate from the macro HELP.
c) Note the use of global variables to allow the mode and state of the macro to be saved so that at the next call, the desired options are still in effect.

```
MACRO HELP
"
" Demonstration HELP function.
"
DEFAULT HELP.STATE=Ø
IF HELP.STATE EQ  1 GOTO ACTIVE
IF HELP.STATE EQ -1 GOTO MENU1
"
" Initialize
LET HELP.STATE=1
LET HELP.MODE =Ø
"
LABEL PRESENT
WRITE 'PRESENTATION.'
WRITE 'This is a demonstration of some possible help'
WRITE 'function facilities. The application is the earlier'
WRITE 'linear algebra package.'
WRITE 'The help function can work in different modes,'
WRITE 'selectable in the MODES section.'
WRITE 'The help function utilizes functions that also can'
WRITE 'be called upon directly. They are:'
WRITE ' '
WRITE 'HELPSYN CMND  - Displays the command syntax for CMND'
WRITE 'HELPINF CMND  - Displays information on command CMND'
WRITE 'HELPEX  CMND  - Ask questions to help execute CMND'
WRITE ' '
"
LABEL MODES
WRITE 'MODES.'
WRITE 'You may now choose the mode of this help function.'
WRITE ' '
WRITE 'Ø - Obtain information only'
WRITE '1 - Obtain help to execute'
WRITE '2 - Obtain command syntax, execute by yourself'
WRITE '3 - Execute by yourself with no help'
WRITE ' '
LABEL CHOOSE
LET TMP=HELP.MODE
WRITE 'Choose mode by typing the appropriate integer.'
WRITE 'The current value is (' TMP '). You may accept'
WRITE 'this with an empty line.'
READ ; TMP INT
IF TMP LE Ø GOTO WRONG
IF TMP GT 3 GOTO WRONG
LET HELP.MODE=TMP
GOTO MENU
LABEL WRONG
WRITE 'Your answer must be in the range Ø-3'
GOTO CHOOSE
"
LABEL MENU1
LET HELP.STATE=1
LABEL MENU
WRITE (FF) ' '
WRITE 'MENU.'
```

```
WRITE 'The following facilities are available:'
WRITE ' '
WRITE 'MATRIX :'
WRITE 'LETME  :'
WRITE 'PRINT  :  Application commands'
WRITE 'MATOP  :'
WRITE 'EIGEN  :'
WRITE ' '
WRITE 'MODES   - Change help mode'
WRITE 'PRESENT - Obtain the presentation of help'
WRITE 'EXIT    - Exit from the help function'
WRITE ' '
WRITE 'What is your interest?'
READ ANSWER NAME
IF ANSWER EQ MODES   GOTO MODES
IF ANSWER EQ PRESENT GOTO PRESENT
IF ANSWER EQ EXIT    GOTO EXIT
" Must be a request for an application command
" Act according to mode
IF HELP.MODE EQ Ø GOTO INFO
IF HELP.MODE EQ 1 GOTO XCT
IF HELP.MODE EQ 3 GOTO LEAVE
" Must be mode 2
HELPSYN ANSWER
LABEL LEAVE
WRITE 'Now you are in command mode.'
WRITE 'Return to HELP by typing RESUME.'
SUSPEND
GOTO MENU
"
LABEL INFO
HELPINF ANSWER
GOTO MENU
"
LABEL XCT
HELPEX ANSWER
GOTO MENU
"
LABEL ACTIVE
WRITE 'HELP is already active.'
WRITE 'Use RESUME to obtain more help.'
GOTO END
LABEL EXIT
" HELP not active any more.
LET HELP.STATE=-1
LABEL END
END
```

Example 3a

```
MACRO HELPSYN CMND
GOTO CMND
WRITE 'There is no command with name ' CMND
GOTO EXIT
LABEL MATRIX
WRITE 'MATRIX MNAME (D1,D2)'
WRITE '  Subcommands:'
WRITE '  ROW (I) V1 V2 ... VN'
WRITE '  EXIT'
GOTO EXIT
LABEL LETME
WRITE 'LETME MNAME(I1,I2) = V'
GOTO EXIT
LABEL PRINT
WRITE 'PRINT MNAME'
GOTO EXIT
LABEL MATOP
WRITE 'MATOP MNAME = EXPRESSION'
GOTO EXIT
LABEL EIGEN
WRITE 'EIGEN EVAL EVEC = MNAME'
LABEL EXIT
END
```

Example 3b

```
MACRO HELPEX CMND
GOTO CMND
WRITE 'There is no command with name ' CMND
GOTO EXIT
LABEL MATRIX
LABEL LETME
LABEL PRINT
LABEL MATOP
WRITE 'HELPX for ' CMND ' is not implemented'
GOTO EXIT
"
LABEL EIGEN
WRITE 'Name of eigenvalues?'
READ EVAL NAME
WRITE 'Name of eigenvectors?'
READ EVEC NAME
WRITE 'Name of matrix?'
READ A NAME
EIGEN EVAL EVEC = A
LABEL EXIT
END
```

Example 3c

```
MACRO HELPINF CMND
GOTO CMND
WRITE 'There is no command with name ' CMND
GOTO EXIT
"
LABEL MATRIX
WRITE 'Declares a matrix. Values are assigned'
WRITE 'via subcommands.
GOTO EXIT
"
LABEL LETME
WRITE 'Assigns a value to a matrix element'
GOTO EXIT
"
LABEL PRINT
WRITE 'Outputs a matrix to the terminal'
GOTO EXIT
"
LABEL MATOP
WRITE 'Evaluates matrix expressions'
GOTO EXIT
".
LABEL EIGEN.
WRITE 'Computes eigenvalues and eigenvectors of
WRITE 'a square matrix'
GOTO EXIT
"
LABEL EXIT
END
```

Example 3d

6. ACKNOWLEDGEMENTS

Intrac as presented in this report is the fruit of many year's evolution. It has been designed as a common facility to handle interaction in a number of programs developed at our department. Many poeple have been involved in this work.

First of all, the programming of this, as we think, final version has been done by Tommy Essebo and Tomas Schonthal, based on the original FORTRAN version by Staffan Selander.

Secondly, the large group of users of, and contributors to, these interactive programs have formed a realistic environment in wich our ideas have been tested. Naturally they have had to endure errors and malfunctions. To a significant extent, this group consists of our friends and collegues. Not least due to our common head of department, K-J Astrom, they all help to create an inspiring atmosphere for wich we are grateful.

Finally, financial support from ITM (Institute of Applied Mathematics) and STU (Board of Technical Developement) has been recieved for many years, and is gratefully acknowledged.

APPENDIX

1. Syntax notation

The following syntax notation is used.

/ or (separates terms in a list from which one and only one must be chosen)

{ } groups terms together

[ ] groups terms together and denotes that the group is optional

{ }* denotes repetition one or more times

[ ]* denotes repetition none or more times

It should be noted that the syntax is not complete in some respects. It does not contain the definition of basic items like <identifier> and <number>. Trivial production rules such as <macro identifier> ::= <identifier> are omitted. Items are sometimes underlined in the syntax. It is used to indicate that an item could be replaced by a variable with the value 'item'.

Example

<number> could formally be replaced by <number>/<variable>. The notation also gives the information that the value of the variable must be a number.

2. Summary of Intrac-statements

MACRO <macro identifier>[<formal argument>/<delimiter>/
<termination marker>]*

Begins a macro definition and creates a macro.

FORMAL {<formal argument>/<delimiter>/
<termination marker>}*

Declares formal arguments in a macro definition and when creating a macro.

END

Ends a macro and ends macro creation mode.
Deactivates suspended macros.

LET {<variable>=}* {<number>[{+/-/*//}<number>]
/{+/-}<number>
/<identifier>[+<integer>]
/<delimiter>
/<unassigned variable>

Assigns (allocates) variables.

DEFAULT {<variable>=}* <argument>

Assigns a variable if it is unassigned or does not exist previously.

LABEL <label identifier>

Defines a label.

GOTO <label identifier>

Makes unconditional jump.

IF <argument> {EQ/NE/GE/LE/GT/LT} <argument>
GOTO <label identifier>

Makes conditional jump.

FOR <variable> = <number> TO <number> [STEP <number>]

Starts a loop.

NEXT <variable>

Ends a loop.

WRITE [([DIS/TP/LP] [FF/LF])] [<variable>/<string>]*

Writes variables and text strings or displays currently available variables.

READ { {<variable> {INT/REAL/NUM/NAME/DELIM/YESNO}} /
<termination marker> }*

Reads values for variables from the terminal

SUSPEND

Suspends the execution of a macro.

RESUME

Resumes the execution of a macro.

SWITCH {EXEC/ECHO/LOG/TRACE} {ON/OFF}

Modifies switches in Intrac.

FREE {{<global variable>}* / *.*}

Deallocates global variables.

STOP

Stops the execution of the program.