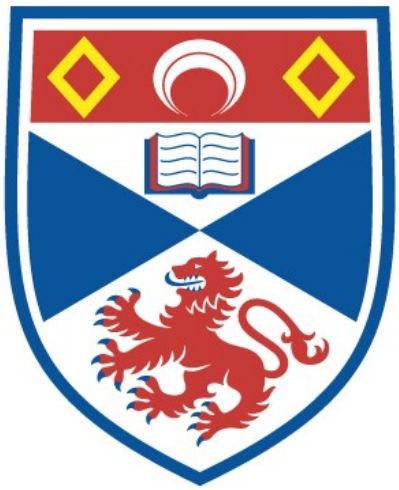
University of St Andrews



Full metadata for this thesis is available in
St Andrews Research Repository
at:
http://research-repository.st-andrews.ac.uk/

This thesis is protected by original copyright

CONVERSATIONAL PROGRAMMING TECHNIQUES

by

Robert M. Campbell B.Sc.

Computing Laboratory

University of St. Andrews

A thesis presented for the degree of Master of Science

September 1968

Th 5583

CONTENTS

INTRODUCTION

Conversational mode programming is a technique of computer programmingbby which a user can communicate to a computing machine and the machine immediately return him computed results or comments. This is usually achieved by a two way dialogue often of a question and answer nature which, in most cases, takes place via an on-line console typewriter.

With the present growth of multi-access systems and the development of moderately priced, medium size computers the topic of man-machine communication using on-line conversational techniques has aroused much interest.

When digital computers were first introduced it was a common practice to sit at the console entering instructions and data via a keyboard, telephone dial or a series of switches and buttons. During such a process the computer spent a great deal of time idle while waiting for the user to enter information. However as the demands on computing time grew it was realised that the only way to get through the increased work load was to ensure that the machine was active all the time and any process which involved the computer waiting for human inputs was discouraged.

The situation was somewhat improved with the use of punched cards and paper tape for the input of both programs of instructions and data. Using such input media a user could formulate his program of instructions and punch up his data block off-line and thus save valuable processing time. The output of computed results was similarly improved through the use of card and paper tape punches and line printers.

Improvements were made not only in the input/output phases of computing but also in programming. To write a program of instructions in a machine code is a time consuming task. However if a high level language, which takes care of many of the detailed operations which require to be fully sequenced in machine code programming, is used then, not only is there a reduction in pre-computer time but also the program is more compact and comprehensible and less likely to contain errors and thus requires only a minimum of computer runs for debugging purposes.

The next step in the drive for efficient computer usage was the development of operating systems. Such systems termed Supervisors, Monitors and Executives relieve the Human operator of many of the routine tasks involved in controlling the sequential processing of a job load.

However, as the calculating speeds of computers increased with improvements in technology it became obvious that even for machines which appeared to be fully occupied there was still a great deal of processing time being wasted while waiting for peripheral devices such as line printers and card readers.

With the introduction of buffered storage areas for input and output processes allowing asynchronous data transfers to be efficiently accomplished a new concept in computing emerged: Multi-programming. Multi-programming allows more than one independent program to be held in core store at the same time. The basic concept of such a system is that though only one program is active at any one instant, if its processing should be held up for a slower device then the computer switches to another program. Multi-programming not only makes for efficient use of a computer but also provides the user with a faster turn-round time than would be achieved from a single program machine.

Multi-access systems have developed out from the basic idea of multi-programming and permit a large computer to have connected to it many communications terminals. Each user gets the impression of having the computer to

himself as the control system switches the internal processing facilities to each user in turn. Such a system, which again allows direct on-line user-machine communication as was experienced in the early days of computing but without the accompanying wastage in processing time, is possible because of the internal speeds of the computer which are so much in excess of the speed with which a user can exchange information with the system. Thus by using a series of buffers for input and output it is possible to service several consoles simultaneously with the minimum of wasted processing time.

The facilities for conversational computing made available through multi-access systems are of use not only to the professional computer programmer for on-line programming but are also of value to the computer user with no formal programming knowledge. Such a user may define his problem in some simple high level language or an alternative approach is to maintain a library of routines in storage for the solution of frequently encountered problems, the user specifying the required routine by a code name.

Though conversational computing is ideally suited to multi-access time shared computers there is still a

role to be played by small, single program machines. Many of the conversational programs now used in a time shared environment were initially developed on single program machines and it is common practice to work on-line with a small computer which calls on the services of a large machine only on an interrupt basis.

This thesis presents the results of a study of several conversational programming techniques.

Chapter 1 outlines the requirements of a conversational programming language and reviews a variety of conversational systems which have been implemented in several different environments.

Chapter 2 describes some of the hardware configurations employed in conversational systems.

The concept of user oriented conversational computing is introduced in Chapter 3 while Chapter 4 is devoted to a description of STAT-CHAT, a user oriented conversational program for on-line data analysis, developed by the writer.

A general discussion on conversational programming is given in Chapter 5.

CHAPTER 1

CONVERSATIONAL PROGRAMMING LANGUAGES

There are many ways in which a program of instructons can be presented to a computer but ultimately all their functions are carried out in patterns of binary digits. It would however be highly inconvenient to actually prepare programs in binary and this is virtually never done.

The most basic form of program is one written for a particular machine in its own machine code for which a translater is provided by the manufacturer to translate the decimal numbers into binary and store them in the desired core store locations.

Programming in a machine code requires great attention to detail and in addition makes the understanding of the program difficult, however if a high level language is used for programming, the computer is made to do much of the routine hard work and the finished program is readable by anyone with only the most basic knowledge of the programming language. A high level language simplifies programming by allowing the programmer to state in a relatively simple language the steps to be carried out for the solution of

a problem and to obtain automatically from the computer, under direction of the processor, an efficient machine language programmfor this process. In general conversational systems employ some form of high level language for man - machine communication.

Raphael (1966) recognises three major components of a programming language:

(i) the elementary program statement,
(ii) mechanisms of linking elementary statements together,
(iii) a means by which programs can bbtain data input.

Elementary program statements are of three types:

(i) command: an imperative statement that commands the action to be taken without saying anything at all about what effect will thereby be achieved;
(ii) requirement: describes the effect to be achieved without saying anything at all about the actions to be taken in achievingiit nor requiràng the programmer to know how the effect will be achieved;
(iii) implicit specification: which is similar to

a requirement but the programmer must know something about what actions will be taken to achieve the desired effect.

One of the main factors influencing the composition and structure of a language is the mode in which it will be used. Batch or job-shop computer operation requires a language to minimise computer running time but on-line conversational operation, whether on a single program machine or a time shared system, requires a language not only to minimise run time but also to present a user / machine interface which is essentially user oriented. To achieve such a result through an on-line language with, in general, overall characteristics similar to those of off-line languages, requires the optimisation of routines for the handling of the input / output, compilation, interpretation and processing of the user's instructions and data.

Though conversational languages are basically of the same structure as off-line languages, their on-line nature allows the user communication with the computer before, during and after program execution and is not restricted to user / computer before program execution and computer / user after program execution as in the off-line languages. In the design of any conversational system much attention must be paid to the provision of

the facility to allow the user direct computer access and to control the execution of program instructions as they proceed.

There are several different types of conversational mode system. An obvious dichotomy is into systems for use with multi-access machines and these implemented on single program machines. Such a classification is however misleading as in many multi-access systems, eg MAC, several of the conversational mode programs have been developed and run successfully on single program machines. A more useful classification is one based upon the class of user for which the systems were designed. Using this criterion three groupings can be made:

(i) systems designed for the formulation, compilation, testing or debugging of programs,

(ii) user oriented systems designed for the non programmer,

(iii) "intelligent" systems for true conversational interaction.

The remainder of this chapter is a review of a number of conversational mode programs which have been implemented on a variety of computer systems.

JOSS

A system falling into the first grouping above is the JOHNIAC Open Shop System (JOSS) designed and implemented by the RAND corporation to facilitate on-line time shared computing (Shaw 1964, 1968).

The purpose of JOSS was to provide users with an easy direct way of solving small numerical problems through a machine independent language. It was to be an experiment with the aim of demonstrating the value of on-line access to a computer via an appropriate language and was intended as a contribution to the long term project of sophisticated information processing.

Several time shared systems provide the user with access to a computing machine and to various languages. These systems can be regarded as open. (Shaw 1964) JOSS however is a closed system man / machine communication being via a powerful integrated high level language which makes it smooth and easy to use for small numerical computations (this is discussed further in Chapter 3). The JOSS system can conveniently be thought of as a computing aide allowing man-machine interaction by means of a simple language (see Example 1).

Example 1: JOSS language and interaction (taken from Shaw (1964))

U: denotes inputs from the JOSS user.

J: denotes outputs from JOSS.

(a)

```
U:  Type 2+2.
J:       2+2=    4
U:  Set  X=3.
    Type X.
J:         X=    3
U:  Type X+2,X-2,2-X,2.X,X/2,X*2.
J:       X+2=    5
         X-2=    1
         2.X=    6
         X/2=    1.5
         X*2=    9
U:  Type ((|X-5|.3+4).2-15).3+10.
J:  ((|X-5|.3+4).2-15).3+10=25
```

(b)

```
U:  Delete all;
    1.1 Do part 2 for b=1(1)a.
    2.1 Setcc=sqrt(a*2+b*2).
    2.2 Type a,b,c in form 1.
```

```
      Form 1:
      a=__ b=__ c=__.__
      Do part 1 for a=1(1),3.
J:    a=1     b=1     c=1.414
      a=2     b=1     c=2.236
      a=2     b=2     c=2.828
      a=3     b=1     c=3.162
      a=3     b=2     c=3.606
      a=3     b=3     c=4.243
U:    Type all steps.
J:    1.1 Do part 2 for b=1(1)a.
      2.1 Set c=sqrt(a*2+b*2)
      2.2 Type a,b,c in form 1.
U:    Type all forms
J:    Form 1:
      a=__ b=__ c=__.__
U:    Type all.
J:    1.1 Do part 2 for b=1(1)a.
      2.1 Set a=sqrt(a*2+b*2).
      2.2 Type a,b,c in form 1.
      Form 1:
      a=__ b=__ c=__.__
      a=3
      b=3
      c=4.24264069
```

U: 2.15 line if fp(c)=0.

2.2 Type a,b,c in form 1 if fp(c)=0.

Type part 2.

J: 2.1 Set c=sqrt(a*2+b*2).

2.5 Line if fp(c)=0.

2.2 Type a,b,c in form 1 if fp(c)=0.

U: Do part 1 for a=1(1)15.

J: a=4 b=3 c=5.000

a=8 b=6 c=10.000

a=12 b=5 c=13.000

a=12 b=9 c=15.000

a=15 b=8 c=17.000

U: Delete part 2.

Type part 2.

J: Error above: No such part.

U: Type all values.

J: a=15

b=15

c=21.2132034

U: Delete all.

Through the JOSS language the user has the option of typing a direct command or an indirect step without having to call for different operating modes. The numeric labels prefixing steps are sufficient to store the steps in sequence. The user must supply labels for all steps in a stored program and these labels determine whether an indirect step is an addition, an insertion or a replacement of another step.

The language includes several logical operators which together with the numeric relations and and or enable powerful constraints to be set on any step.

As JOSS is based on an interpretive technique the user is allowed to edit his stored program freely and quickly. The language provides "tags" at various levels of aggregation so that the user is not forced to do his own editing at one particular level. Steps are aggregated into parts and parts into units. Higher aggregates may be addressed by using operands such as all steps, all parts, all forms, all values and all.

When in use the current status of the conversational system is indicated by various typed messages distinguishing the states of interrupt, stop and error from the successful

completion of a task. The error messages report violations in the language or syntactical errors and permit the user to correct them as they are diagnosed.

A record of the frequency of occurrence of language violations is maintained from which the systems programmers and designers responsible for JOSS can obtain indications of possible weak points in the system and make modifications accordingly. This has proved to be an extremely useful inclusion in the system.

List processing is the corner stone of the JOSS system. List processing routines are used for storage and operation of step parameters and also to handle the numerical representation of vector elements. A vector is represented by a short list of elements each with its index.

Quite elaborate list structures are necessary for JOSS to record the current step number at each level in a hierarchical operation and this is made more elaborate if a step carries a conditional clause requiring the storage of additional information to control the iterative process.

One of the difficulties experienced in designing the JOSS system was to ensure that a user's block was not

irreversibly altered before it was certain that the execution of an operation would proceed to a satisfactory conclusion without error, thus allowing the user to correct the error and proceed with the originally intended operation.

JOSS was originally implemented on the JOHNIAC computer serving ten remote consoles. The JOHNIAC computer has a 40 bit word and 4096 words of core store with additional storage for 12288 words on a slow access drum. The JOSS program runs to about 6000 words a large part of which resides in permanent store but the low frequency portions are stored on the drum and overlay each other in core store when called for execution. Because of the deeply hierarchical nature of the program and the lack of an adequate subroutine linkage operation it was a considerable challenge for the designers to compress the program sufficiently to leave room in store for the processing of a user's program block.

To date the system has over 100 users and the project is regarded as having been successful. The advantages offered by the JOSS system of conversational computer operation lies in the syntax semantics and power of the communication language which permits the user to communicate with the computer on a high level and in an easily under-

standable form involving no technical jargon;

International Computers and Tabulators Limited have used the JOSS language as a model for JEAN (ICT 1967) a conversational system for the 1900 series of computers but this, unlike JOSS itself, is not readily understandable due to the excessive use of jargon and, in the writer's opinion, must be regarded as a retrograde step from JOSS and not a development.

The merits of JOSS are best expressed in the following quote from a senior electronics engineer arguing for a replacement of the JOHNIAC computer.

"JOSS is becoming essential to our output like paper, or coffee. It speeds up calculations; makes it easy to try experiments. It is the greatest, when it works. I have heard it compared favorably to beer and referred to as RAND's most important fringe benefit. People adjust their lives to fit around JOSS.....No use coming to RAND before 10:00 am. When JOSS arrives; in fact noon or after 5:00 pm. is abbetter time, JOSS is less busy. When JOSS starts typing answers, the titillating pleasure is equalled only by the ensuing anguish when JOSS breaks off into jibberish or goes away commending your program to oblivion.

We can hardly live with JOSS, but we can't live without it. We're hooked" (Shaw 1968)

THE ENGINEERING AND SCIENTIFIC INTERPRETER

Waks (1967) describes a conversational system - the Engineering and Scientific Interpreter (E.S.I.) written for implementation on single program machines such as the PDP-5, PDP-8, PDP-8/S and the LINC-8. The E.S.I. is a self contained system which simplifies the programming and debugging of modest scale programs and is highly interactive with the user allowing an easy transition from design, to coding and debugging, to program execution. The Interpreter provides, for a single user, most of the facilities of JOSS and, as in JOSS, the command statements take the form of English like imperative sentences (see Example 2).

Example 2: Conversation with the Engineering and Scientific Interpreter (adapted from Waks (1967))

```
Delete all.
Type 2+2.
     2+2 =      4
Set  X=5.
Type X.
       X =      5
Type X*X3,X/7,SGN(X),SQRT(X).
```

```
X**3 = 125
X/7 =    .7142857
SGN(X) =  1
SQRT(X) =  2.236068
Type X*Y.
Error above: Undefined
Set Y = -7.
Type X*Y
    X*Y = -35
```

Each statement contains a main clause which starts with an imperative verb and ends with a period. Any clause not involved in the transfer of control can be preceded with one or more iterative clauses which begin with the word for and end with a comma. Any statement may be preceded with a conditional clause.

The E.S.I. provides a powerful tool for the solution of modest size numerical computations and offers a reasonable alternative to the cost of time sharing.

BASIC

BASIC originally appeared as the principal computing language of the Dartmouth time sharing system but is rapidly gaining acceptance as a general purpose on-line computer language (Kemeny and Kuntz 1967).

In many respects BASIC resembles other computer programming languages but it is primarily designed to be simpler and easier to use. It is possible to write a variety of programs using only a few different types of BASIC statement.

The elementary BASIC statements are listed below:

READ	Reads data from data block
DATA	Storage area for data
PRINT	Types numbers and labels
LET	Computes and assigns value
GO TO	Transfers control
IF	Conditional transfer
FOR	Sets up and operates a loop
NEXT	Closes loop
END	Final statement in program

There are several features of BASIC that take advantage of the ability of the conversational user to interact with his program during computation.

One such feature is the INPUT command which is used exactly like a READ statement except that the numerical information is typed in by the user from the console rather than having to be put in a DATA statement. This allows the user to supply data after seeing some partial results. As the data to be inputted may be a coded signal to the program the user is permitted to make policy decisions during the course of a computation.

When an INPUT statement is encountered at run time a question mark is typed and the computer halts until the user enters the necessary data, after which the computation proceeds from where it was interrupted.

Real time input is used in the program example DEGREE given below which converts temperatures from degrees Fahrenheit to degrees Centigrade.

```
LIST

DEGREE

100   PRINT "FAHRENHEIT";
110   INPUT F
120   LET C = (F-32)*5/9
130   PRINT "CENTIGRADE:"C
140   PRINT
150   GO TO 100
160   END

RUN

DEGREE

FAHRENHEIT? 32
CENTIGRADE:  0

FAHRENHEIT? 212
CENTIGRADE: 100

FAHRENHEIT? STOP
STOP.
READY.
```

Line numbers serve a dual role in BASIC.

(i) They serve as labels so that the order of computation may be changed by a GO TO or IF-THEN statement.

(ii) They serve to identify lines for the purpose of updating, correcting and expanding programs.

To change an incorrect line it is merely retyped with the same line number. To delete the line the line number alone is typed. To insert a line or add a line before or after a program the new line with an appropriate line number is entered. The system automatically rearranges lines in the indicated order.

As it is both simple and sophisticated BASIC holds great promise as a language for use in on-line time sharing environments. Successful implementations of BASIC have been demonstrated by several computer manufacturers.

DOCUS

DOCUS (Display Oriented Computer Usage System) described by Corbin and Frank (1966) was developed primarily to investigate software techniques for on-line information processing and man / machine communication using visual displays. The system has been implemented on a CDC 1904 B computer with a Bunker Ramo 85 display console, but the techniques used are however not restricted to this hardware.

The DOCUS system was designed both for the compilation of programs and for the execution of pre-programmed routines by non programmer users and this falls into both categories (i) and (ii) of the previously outlined classification structure.

Man machine communication is achieved by a series of related user actions such as pressing a function key, entering data via a keyboard or pointing a light pen. The computer responds by presenting him either with a display or changing the state of indicator lights. The user examines the computer's message and performs another action. The process continues until the user is satisfied or the computer is unable to respond to the user in a satisfactory manner upon which an error condition occurs.

MAN/MACHINE INTERACTION THROUGH DOCUS

USER	COMPUTER/CONSOLE
Press Function key	
	Identifies functions and presents display.
Examines display; enters data or selects item with light pen.	
	Processes data and notifies user of results with a display or status light.
Observes results; continues another action.	

(adapted from Corbin and Frank 1966)

27

Two types of language are used in DOCUS to handle common functions related to the general operation of the system and the common recurring functions. The executive languages are oriented toward the particular type of user.

GOL (General Operating Language) provides the basic framework in which a non programmer user may solve his problem by application of functions from his own library or the system library.

PIL (Procedure Implementation Language) provides the basic framework in which the programmer may solve his problem allowing him to create and modify functions and displays.

The languages available to the DOCUS programmer are FORTRAN, CODAP, DOL (Display Oriented Language) and Debugging Language.

The keys of the FORTRAN key group represent FORTRAN statements. When a function key is pressed the particular FORTRAN statement with appropriate blanks to be filled in is placed on the display screen. The operator then enters via the keyboard any variables necessary to complete the statement. Using such a system eradicates syntactical

errors such as:

DO 10, I = 1, N

as when a user presses the DO function key he receives a display of the form:

DO __ = _ , _

and he is required merely to type in the appropriate statement number, an index and the range of that index through which the loop is to be executed. An additional advantage is that less machine checking of the input need be performed.

CODAP is designed as a coding tutor and provides an on-line reference manual and a comprehensive source for utility routines.

DOL is used to describe, create and manipulate displays and develop conversational mode procedures. The comprehensive form of DOL permits symbolic reference to information strings, areas, lines, pages and displays. Full displays can be assembled from partial displays and static and dynamic displays can be mixed. Data entered into a format display can be extracted for use by a key program.

The Debugging Language is available to assist in the checkout of a program. Control and criteria data are entered via a format display and the output from the debugging operation is either displayed or written on the on-line printer.

Languages available to the non programming user under the GOL executive are theoretically unlimited though only a few have been actually implemented being mainly to test ideas on console usage. These languages are a simplified File Management Language, Text Manipulation Language, a Computation Language and a Graphics Language.

STATPAC

STATPAC, a user oriented display system, was developed by Goodenough (1965) for on-line data analysis. The system was designed initially for use by psychologists at Decision Service Laboratory, Hanscom Field, Bedford, Mass..

The aims of STATPAC were:

(i) to have an operating procedure which was instructive and capable of being learned quickly by scientists who were not programmers.

(ii) To permit a user to return and reuse the system without difficulty after long periods of disuse.

(iii) To perform small jobs quickly in terms of user time.

To reach the required degree of man-computer interaction to achieve these aims a light pen and computer driven scope are used for communication of user commands and machine computed results.

Communication from user to computer is by a light pen which the user points at a display of operations.

This displayed English like "menu" of operations frees the user from the necessity of learning and remembering the capabilities of the system as no matter what he points at with his light pen he cannot confuse the system. There is also no need for the user to know which operations are monadic and which dyadic. A simple set of rewriting rules in the program ensure that only syntactically correct commands are given to the computer (see Example 3).

Example 3: STATPAC (adapted from Goodenough 1965)

(a) display after reading in data

```
YOU HAVE REQUESTED
EXECUTE     TITLES     OPERATIONS     RESET

DATA IN STORAGE
20      TEST 1
20      TEST 2
20      RATINGS
```

(b) display after pointing at Ratings

```
YOU HAVE REQUESTED
(RATINGS)
```

EXECUTE TITLES OPERATIONS RESET

DISPLAY
TYPE
PUNCH

CORR BETWEEN (X) AND (Y)
STD DEV OF
MEAN OF
REGRESSION (X) VS (Y)
SUM OF SQUARES OF

EXPUNGE
EXP
DIVIDE
LOG
MINUS
MULTIPLY
SUM
PLUS

(c) display after pointing at "Corr between (X) and (Y)"

YOU HAVE REQUESTED
CORR BETWEEN (RATINGS) AND (Y)

EXECUTE TITLES OPERATIONS RESET

DATA IN STORAGE
20 TEST 1
20 TEST 2
20 RATINGS

(The user now points at one of the data blocks - Test 1

and then at "Execute" and the value of the correlation coefficient between Ratings and Test 1 is typed by the on-line typewriter).

THE LINCOLN RECKONER

The Lincoln Laboratory at M.I.T. has developed the Lincoln Reckoner (Stowe et al 1966), a time-shared on-line conversational system for use in scientific and engineering research.

The Reckoner is primarily a facility for making use of routines and not for writing them. To execute a routine the user merely types its name on the on-line typewriter, the operands on which it is to operate and the name under which the results are to be filed.

The library of routines concentrates on numerical computations on arrays of data and are intended for use in feeling one's way through the reduction of data from a laboratory experiment or for trying out small theoretical calculations.

The Reckoner belongs to a class of systems which includes MAP (Kaplow et al 1966, Schwartz 1966) and the Culler-Fried System (Culler and Fried 1965, Orr et al 1968) which have the following features:

(i) the automatic application of routines - requiring the

user only to indicate the operation he wants to apply and the data to which he wants it applied.

(ii) The automatic retention of results allowing them to be used as operands for data routines.

(iii) Facilities for the concatenation of routines which permit the user to define a sequence of operations and then to use the sequence as he would a single operation. The new operation can also form part of another sequence.

The Reckoner runs on the Laboratory TX-2 experimental computer under a time shared system called APEX. APEX provides three important services:

(i) input buffer
(ii) directory of names
(iii) stack of "maps"

The input stacker stores input characters in a buffer area. The Executive notifies the user's program when each statement is complete but continues stacking to the buffer capacity whether or not the program has processed the previous statements.

The Executive maintains a private directory for each user and also a public directory of library routines. The directories are ring structured dictionaries of names and definitions. The user's directory can contain a name defined by a pointer to a definition in a public directory though it is permissible for names to be undefined.

User's programs are not allowed to write in the directories but they can manipulate the user's private directory and inspect both directories through calls to the Executive.

APEX provides each user with an "apparent computer" whose core is divided into 16 segments. A segment can contain only one file and a "map" is a list specifying the name of the file, if any, that goes into each segment. A connector file common to all maps enables information to be passed between routines on different maps.

The basic translator receives the user's input from the buffer area, cleans it up and initiates an APEX lookup for the name.

APEX looks first in the user's file and then in the public directory and if the name is not found it is inserted in the user's private directory as undefined. In either

case APEX replies with a pointer to the location of the name in one of the directories.

The translator types an error message if it is unable to interpret an input character or if the directory will not accept a name.

If there are no errors the translator puts into the connector a calling sequence that is a term for term translation of the statement the user typed. Numbers appear in binary form and names as pointers to the directory. A "go up" call to the routine to be executed is initiated.

When a routine has been entered by "going up" to it it receives instructions from the calling sequence in the connector. If the operation required will manipulate data files pointers in the calling sequence mark these enabling the space required for working storage and for results to be computed and newfilesoof the appropriate size to be created.

The routine proceeds to do the computations which are its main task, places the result in the desired file and "peels" back to whatever called it.

THE COLIN ON-LINE ACCESS SYSTEM

Colin (1966, 1967) has developed a conversational program for on-line data analysis using the I.B.M. 1620 computer.

This is a simple system written in 1620 Symbolic Assembly Code and using the on-line console typewriter for input of user commands and output of computed results.

Communicationsis kept to a very simple level information being suppliedby theuser in uniits of one word or number at a time and takes the form of a question and answer dialogue (see Example 4).

Example 4: (adapted from Colin 1966)

HELLO. WHENEVER YOU TYPE A MESSAGE ON THIS TYPEWRITER PLEASE TERMINATE IT WITH THE R.S KEY TO BE FOUND AT THE TOP LEFT-HAND CORNER OF THE KEYBOARD. IF YOU MAKE A MISTAKE, THEN TYPE @ JUST BEFORE R.S AND WHAT YOU HAVE JUST TYPED WILL BE IGNORED. HAVE YOU EVER USED THIS PROGRAM BEFORE. (PLEASE TYPE YES OR NO)

NO R.S

PLEASE GIVE YOUR NAME

A. J. T. COLIN R.S

A FILE HAS BEEN OPENED UNDER YOUR NAME
THIS IS A STATISTICAL PROGRAM, AND YOUR FIRST STEPS MUST BE TO SUPPLY THE MACHINE WITH DATA TO BE ANALYSED. YOUR DATA PRESUMABLY CONSISTS OF A SET OF RECORDS EACH OF WHICH CONTAINS NUMERICAL VALUES FOR ONE OR MORE VARIABLES. WOULD YOU LIKE AN EXAMPLE (TYPE YES OR NO)

NO THANKS R.S

NO. YOUR CHOICE IS LIMITED TO THE FOLLOWING
YES
NO
NOW TRY AGAIN.

NO R.S

YOUR DATA MUST BE PUNCHED ON TO CARDS, ONE CARD TO EACH RECORD. THE VALUES OF THE OBSERVATIONS MAY BE ARRANGED IN ANY WAY YOU LIKE, PROVIDED THAT FRACTIONAL VALUES ALWAYS HAVE A DECIMAL POINT IN THE RIGHT PLACE, AND THAT THE ARRANGEMENT IS THE SAME FOR ALL YOUR CARDS. YOU MUST

DECIDE ON NAMES (SUCH AS *AGE* OR *IQ* FOR ALL YOUR VARIABLES.

HOW MANY VARIABLES ARE THERE IN EACH RECORD

4 R.S

GOOD. YOU MUST NOW DESCRIBE THE WAY YOUR VARIABLES
ARE ARRANGED, AND GIVE THEM NAMES. WE SHALL DEAL WITH
THE VARIABLES ONE AT A TIME

VARIABLE 1

WHAT NAME DO YOU GIVE TO THIS VARIABLE

CLOUD COVER R.S

WHAT IS THE FIRST CARD COLUMN WHERE VALUES OF THIS VARIABLE
ARE PUNCHED

SEVEN R.S

THIS IS NOT A VALID NUMBER

NOW TRY AGAIN

7 R.S

WHAT IS THE LAST COLUMN

7 R.S

SLOUD COVER IS PUNCHED IN 1 COLUMNS STARTING AT COLUMN 7

IS THIS RIGHT

YES R.S

(the program continues in this manner for all 4 variables)

THE VARIABLES YOU HAVE DEFINED ARE AS FOLLOWS

FIRST COLUMN	LAST COLUMN	NAME
7	7	CLOUD COVER
13	16	TEMPERATURE
18	21	HUMIDITY
23	26	PRESSURE

ARE YOU SATISFIED THAT YOUR DEFINITIONS ARE CORRECT

YES R.S

LOAD YOUR CASE CARDS INTO THE 1620 READER, FACE DOWN,

WITH THE *NINES* EDGE TOWARDS THE MACHINE

PUT THREE BLANK CARDS ON TOP AND START THE READER (GREEN

BUTTON)

VARIABLE VALUES ON THE FIRST CARD ARE

CLOUD COVER = +2.0000000

TEMPERATURE = +47.800000

HUMIDITY = +65.000000

PRESSURE = +1006.0000

IS THIS RIGHT

YES R.S

NUMBER OF CASES READ = 109

THE HIGHEST AND LOWEST VALUES OF YOUR VARIABLES ARE

CLOUD COVER	0.0000000,	+8.0000000
TEMPERATURE	+47.100000,	+68.100000
HUMIDITY	+45.000000,	+99.000000
PRESSURE	+991.00000,	+1029.0000

THE COMPUTER IS NOW READY TO COMPUTE STATISTICS. WOULD YOU LIKE TO SEE A BIST OF THOSE IT CAN WORK OUT

NO R.S

WHICH STATISTIC DO YOU WANT

AVERAGE R.S

FOR WHICH VARIABLE

TEMPERATURE R.S

AVERAGE VALUE OF TEMPERATURE = +56.729357

(the program continues in this manner)

Each step to be followed by the user is clearly typed by the machine and every item of information required is elicited by a question.

The program allows the user to call for any of a repertoire of statistical processes to be applied to his data.

An important point is that the program detects inconsistencies and nonsense and this cannot be broken by mistakes.

The simple method chosen for communication has advantages over more formal styles of problem definition in that there is virtually no syntax to be learned by the user and that mistakes can be corrected immediately without the necessity of retyping entire lines or paragraphs.

The conversational input is handled by a hierarchy of subroutines which, after checking for errors, pass control to another set of subroutines to carry out the required operations.

The program was written for the completely unskilled user and was intended as an experiment in man/machine communication. Though the program has proved to be of much use both as a teaching aid and as a practical tool, in an environment of non programmer computer users, it has a number of defects:

(i) the method of supplying units of information one at a time is less compact than more formal styles.

(ii) No distinction is made between spelling errors and misunderstandings of the system.

(iii) No adaptation is made for the user who had had some experience of the program.

(iv) Commands are not stored and thus the advantages of having a stored program are lost necessitating in a long analysis the typing of the same statement several times.

The above disadvantages of the pilot program are recognised by Colin and in a second version of the program for the I.C.T. 1909 it is intended that many of these will be eliminated.

ELIZA

A number of conversational programs fall within the grouping of "intelligent".

ELIZA is a program operating at M.I.T. under the time shared system of project MAC which makes possible certain kinds of natural language conversation between man and computer (Weizenbaum 1966, 1967).

When in conversation with ELIZA the user types in some statement or set of statements in natural language using normal punctuations and sentence structures. ELIZA analyses the user's statement and generatess some response which it types out giving control again to the user. The overall procedure is that the text is read and inspected for the presence of a keyword. If such a word is found the sentence is transformed according to a rule associated with the keyword, if not, a content free remark or, under certain conditions, an warlier transformation is retrieved. The text, so computed or retrieved is printed out as a computer response (see Example 5).

Example 5: ELIZA - a typical conversation (from Weizenbaum 1966)

Men are all alike.

IN WHAT WAY

They're always bugging us about somethingoor other.

CAN YOU THINK OF A SPECIFIC EXAMPLE

Well, my boyfriend made me come here.

YOU BOYFRIEND MADE YOU COME HERE

He says I'm depressed much of the time.

I AM SORRY TO HEAR YOU ARE DEPRESSED

It's true. I am unhappy.

DO YOU THINK COMING HERE WILL HELP YOU NOT TO BE UNHAPPY

I need some help, that much seems certain.

WHAT WOULD IT MEAN TO YOU IF YOU GOT SOME HEKP

Perhaps I could learn to get along with my mother.

TELL ME MORE ABOUT YOUR FAMILY

My mother takes care of me.

WHO ELSE IN YOUR FAMILY TAKES CARE OF YOU

My father.

YOUR FATHER

Youaare like my father in some ways.

WHAT RESEMBLANCE DO YOU SEE

You are not very aggressive but I think you don't want me to notice that.

WHAT MAKES YOU THINK I AM NOT VERY AGGRESSIVE

You don't argue with me

WHY DO YOU THINK I DON'T ARGUE WITH YOU

You are afraid of me.

DOES IT PLEASE YOU TO BELIEVE I AM AFRAID OF YOU

My father is afraid of everybody.

WHAT ELSE COMES TO MIND WHEN YOU THINK OF YOUR FATHER

Bullies.

DOES THAT HAVE ANYTHING TO DO WITH THE FACT THAT YOUR BOYFRIEND MADE YOU COME HERE

In the above example machine responses are shown in capitals.

Key-words can have a rank or precedence number. The procedure for keyword identification is sensitive to rankings in that it will abandon a keyword already found in a left to right scan of the text in favour of one having a higher rank. Commas and periods are recognised as delimiters. Whenever such a delimiter is recognised and a keyword has been previously found all subsequent text is deleted from the input message. If no keywords have been found the phrase or sentence to the left of the delimiter is deleted. As a result only single phrases or sentences are ever transformed.

The ELIZA program is written in MAD SLIP and formulated in such a way as to cope with the following problems:

(i) The identification of the "most important keyword" (by making use of the rankings mentioned above).

(ii) The identification of some minimal context within which the keyword appears eg. if the keyword is "you" is it followed by the word "are"? (in which case an assertion is probably being made).

(iii) The choice and execution of a transformation.

(iv) Generation of "intelligent" responses in the absence of a keyword.

The transformation procedures are achieved by a number of SLIP functions which serve to:

(i) decompose a data string according to a certain criterion, hence to test the string as to whether it satisfies these criteria or not.

(ii) To reassemble a decomposed string according to certain assembly specifications.

Consider the sentence "I am very unhappy these days". Now if a foreigner with but a limited knowledge of English

heard that sentence understanding only the first words "I am" he may respond with "How long have you been very unhappy these days?". What he has done is applied a type of template to the sentence to match the words "I am" isolating the remainder and then used a reassembly rule that specifies that any sentence of the form "I am -----" can be transformed into "How long have you -----" ie. the input has been transformed according to a rule associated with the keywords "I am". ELIZA functions in a similar way.

The sentence "It seems that you hate me" can be decomposed with keywords "you" and "me" into:

(i) It seems that

(ii) you

(iii) hate

(iv) me

The reassembly may be "What makes you think I hate you" ie. the first component is thrown away, "you" is transformed into "I", "me" to "you" and the stock phrase "What makes you think" is tacked on to the front of the reconstruction.

A formal notation of the template is:

(0 you 0 me)

and the reassembly rule can be stated as:

(what makes you think 3 you)

The "0" in the decompositionrrule stands for an indefinite number of words while the "3" in the reassembly rule indicates that the third component of the subject decomposition is to be inserted in its place.

The scan of the input text is left to right. Each word is looked up in a dictionary of keywords. If a word is identified as a keyword then only decomposition rules containing that word need be tried. The trial sequence can be part ordered as for example the rule used in the example above (0 you 0) will always be successful and thus should be the last one tried with that particular keyword (you).

The two problems which arise with the scanning of the input text are:

(i) almost none of the input words are contained in the keyword dictionary.

(ii) The association of both decomposition and reassembly rules with keywords.

To solve both these problems the keywords and associated rules are placed on a list of the following format:

$$(K((D_1)(R_{1,1})(R_{1,2}) - - - - - (R_1,m_1))$$

$$((D_2)(R_{21})(R_{2,2}) - - - - - - (R_2,m_2))$$

$$((D_n)(R_{n,1})(R_{n,2}) - - - - - - (R_n\ m_n))$$

Where K is the keyword, D_i the i^{th} decomposition rule associated with K and $R_{i,j}$ the j^{th} reassembly rule associated with the i^{th} decomposition rule. Since list structures of this type have no predefined dimensionality limitations any number of decomposition rules can be associatedwith a given keyword and any number of reassembly rules with a given decomposition rule. SLIP is rich in functions that sequence over structures of this type efficiently and hence many of the programming problems are minimised.

By using the list sequencing functions of SLIP the identification of a keyword results not only in a pointer to the list of decomposition and reassembly rules associated with that keyword but also yields pointers to all the information associated with that word.

Transformations of the input text may be made independently of any contextual considerations. First person pronouns may be exchanged for those of the second person and vice versa as in the first example given above, or such changes as "yourself" to "myself" may be made. To facilitate such substitutions any word in a key dictionary may be followed by an equals sign and the word tobbe used for its substitution. Transformation rules may but need not follow.

The only serious ELIZA scripts on record to date are some which cause the computer to respond roughly as would a psychotherapist (see Example 5). Any lack of understanding of the text is concealed to encourage the user to offer more input from which remedial information can be selected. However some workers (ELIZA's writer among them) postulate that a swithh to a revaluation of misunderstandings could lead to making an ELIZA like program the basis of an effective natural language man-machine communication system.

Raphael (Weizenbaum 1967) has developed a program (SIR) which is capable of inferential data acquisition in a way analagous to that displayed in some of the ELIZA programs while Quillian (Weizenbaum 1967) has directed work toward establishing data structures capable of searching semantic dictionaries and deciding upon different meanings for the same word used in different contexts.

STUDENT

Bobrow (Minsky 1966, Weizenbaum 1967) utilising some of the experience of other workers has developed STUDENT, a program capable of solving algebraic word problems of the standard set in secondary school.

Bobrow's concern was to provide the computer with the ability to read the informal verbal statement of a problem and to derive from that the equations required to solve the problem (see Example 6).

Using STUDENT the machine reads the statement of the problem and attempts to convert each simple sentence into an equation, finally trying to solve the set of equations and present the required answer converted back to a simple English sentence.

An extension of the LISP programming language incorporating techniques developed by Yngve forms the basis of STUDENT. Eachsstep in the problem solving process is aided by the memory library which includes a dictionary, a variety of factual statements and several special purpose routines for solving particular kinds of problems.

The strength of the program lies in the ability to recognise and resolve ambiguities either by inferentially acquired information or sometimes by asking the user. This ability rests in the filing system which is loosely arranged yet readily accessible and cuts across the linguists formal distinction between syntax and semantics thus avoiding the problems which tend to hinder most language studies.

Though STUDENT has limited demantic dictionaries and handles only a small part of the grammar of English it probably surpasses the average person in its ability to handle verbally stated algebraic problems though it must be pointed out that its use is somewhat confined to the LISP programmer.

Example 6: "STUDENT" an English-reading program (from Minsky 1966)

(THE PROBLEM TO BE SOLVED IS)
(MARY IS TWICE AS OLD AS ANN WAS WHEN MARY WAS AS OLD AS ANN IS NOW. IF MARY IS 24 YEARS OLD, HOW OLD IS ANN Q)

(WITH MANDITORY SUBSTITUTIONS THE PROBLEM IS)
(MARY IS 2 TIMES AS OLD AS ANN WAS WHEN MARY WAS AS OLD AS ANN IS NOW. IF MARY IS 24 YEARS OLD, WHAT IS ANN Q.)

(WITH WORDS TAGGED BY FUNCTION THE PROBLEM IS)

((MARY/PERSON) IS 2 (TIMES/OP1) AS OLD AS (ANN/PERSON)
WAS WHEN (MARY/PERSON) WAS AS OLD AS (ANN/PERSON) IS NOW
(PERIOD/DLM) IF (MARY/PERSON) IS 24 YEARS OLD, (WHAT/QWORD)
IS (ANN/PERSON)(QMARK/DLM)

(THE SIMPLE SENTENCES ARE)

((MARY/PERSON) S AGE IS 2 (TIMES/OP1) (ANN/PERSON) S AGE
G02521 YEARS AGO (PERIOD/DLM))

(G02521 YEARS AGO (MARY/PERSON) S AGE IS (ANN/PERSON) S
AGE NOW (PERIOD/DLM))

((MARY/PERSON) S AGE IS 24 (PERIOD/DLM))

((WHAT/QWORD) IS (ANN/PERSON) S AGE (QMARK/DLM)

(THE EQUATIONS TO BE SOLVED ARE)

(EQUAL G02521 ((ANN/PERSON) S AGE))

(EQUAL ((MARY/PERSON) S AGE) 24)

(EQUAL (PLUS ((MARY/PERSON) S AGE)(MINUS (G02521))) ((ANN/
PERSON) S AGE))

(EQUAL((MARY/PERSON) S AGE)(TIMES 2 (PLUS((ANN/PERSON)
S AGE)(MINUS (G02521)))))

(ANN S AGE IS 18)

note: the program invented the symbol G02521

POP-2

Popplestone and his colleagues at the Department of Machine Intelligence and Perception of the University of Edinburgh have developed a conversational language (POP-2) (Burstall and Popplestone 1968, Popplestone 1968, Popplestone 1968a) which falls into both categories (i) and (iii) of the previously outlined classification, it being suitable for nonnumeric as well as numeric computation (see Example 7).

Example 7: POP-2

a conversation with a question and answer program

```
VOCAB ((COFFEE CAFFEINE STIMULANT INSOMNIA),(CONTAIN CAUSE));

.RTALK
COFFEE CONTAIN S &&FFEINE;
OK
CAFFEINE IS A STIMULANT;
OK
STIMULANT CAUSE S INSOMNIA;
OK
CAUSE MEANS CONTAIN * CAUSE;
OK
```

```
WHAT DOES COFFEE CAUSE;
INSOMNIA
```

note: In the above example all the user input ends with a semi colon. Any vocabulary (apart from the stock phrases of the language) must be declared as shown in the first line of the example. The construction "cause means contain * cause" "redefines" the meaning of cause so that when the user asks "what does coffee cause" the answer is obtained by first searching for what coffee contains (caffeine which is defined as a stimulant) and then searching for what this causes (insomnia).

In its simplest use POP-2 can be regarded as a powerful calculating tool in which each step of a calculation is carried out immidiately after it is requested while, on the other hand it can be treated as a conventional computing system in which a complete program of instructions representing the whole job is given to the system. In practice a combination of these two modes is used so that the user can choose a size of job step that is convenient to him at the particular time.

The language is powerful and sophisticated for non numerical computation but is not intended for heavy duty

numerical work where high efficiency is required.

Only a few basic concepts went into the design of the system which is based on LISP processing techniques and written to be implemented with the minimum of manpower on real machines.

Functions in POP-2 differ in semantic properties and in the syntactic properties of their names but are all the same in essence as they are handled by the machine as a single class of data structures. Such functions are used to represent procedures, arrays and binary operations.

Two advanced features are incorporated into the POP-2 system. Partial application permits the extraction from a given function of another function with fewer parameters with the missing parameters taking on fixed values. Dynamic lists allow the use of long lists without the utilisation of much store. In a dynamic list some of the values are obtained by calculation rather than by being stored.

POP-2 was originally designed as an on-line applications language for use in the fields of machine intelligence and information retrieval where it holds much promise.

MULTI-POP

MULTI-POP (Pullin 1967, Collins et al 1968) is a conversational multi-access computing system designed for biomedical research workers. The system is usable by non-programmers and programmers alike and is suited to non numerical as well as numerical applications. The basic system has been designed and implemented for the following hardware configuration:

Elliot 4120 central processing unit with 32K (24 bit)
Control typewriter
Paper tape station
Line printer
8 channel teletype controller
20 on-line teletype consoles

The system has been designed so that further peripheral devices can be added and arrangements can also be made to run the system on machines without the full set of peripherals.

The software system includes a library package (POPSTATS) for conversational statistical calculations. This program written in POP-2 uses a question and answer technique

to help the user supply the right information at the right time without him having to have expert knowledge of either the package or POP-2. A special feature of the package made possible by the design of POP-2 itself is that the user can, if he wishes, jump out of the package into the POP-2 compiler environment and thus bridge gaps in the facilities offered by the basic package. He can also modify and extend existing facilities by direct console programming in POP-2 before again returning to the package and continuing his analysis.

The basic MULTI-POP system is now being used as a springboard for MULTI-POP/70 which will be a disk based system for the Elliot 4130. The applications envisaged for this new system are:

(i) on the commercial side, general purpose computing.
(ii) on the research side, the incorporation into the software system of tree searching, rote learning, and inductive and deductive problem solving features.

Given below is a table of conversational systems arranged into the three categories as regognised by the writer. Several systems not here described have also been included.

(i) Systems for the formation, compilation, testing or debugging of programs

JOSS

E.S.I.

JEAN

DOCUS

POP-2

BASIC

(ii) User oriented systems designed for the non-programmer

DOCUS

STATPAC

Lincoln Reckoner

MAP

Culler-Fried System

Colin on-line access system

Klerer-May System (Klerer and May 1964)

MULTI-POP

(iii) Intelligent systems

ELIZA

SIR

STUDENT

POP-2

DOCTOR

PLATO

DAC-1 (Jacks 1968)

Robot Draftsman (Pfeiffer 1968)

Socratic Instruction (Bolt 1968)

HELP(Pirtle 1968)

Computer Interrogation (Meadow and Waugh 1966)

CHAPTER 2

HARDWARE

By hardware one means the mechanical, magnetic, electrical and electronic devices and components of a computer as distinct from the software or programming system which surrounds it.

As illustrated in the preceding chapter, conversational programs have been implemented on varying configurations of hardware.

One of the ideas behind conversational interactive computer usage is to make computers widely available and effectively useful in solving complex problems. To achieve this through advanced problem oriented languages designed for on-line use by people not skilled in conventional programming and topprovide organised systems of prepared programs utilises much memory space of a computer. Thus the memory capacities of a computer determine to a large extent the size and scope of a conversational system. Small computers are favourable for some special purposes where the problems to be tackled are of a smallssize (eg. the E.S.I. is implemented on a small P.D.P. machine) but

in general, large memory capacities are required to cope, not only with the solution of a problem but also with the administration of the interactive system itself. In the larger conversational systems it is common to have as well as a random access store a further backing store either on disk files, magnetic tapes or drums. This supplements the main core store and though the access time is generally slow provides storage for parts of the system which are used only infrequently. (eg. a magnetic drum is used in the JOSS system)

Perhaps the most important hardware component of a conversational system is the device via which man/machine communication takes place. Most conversational systems provide an electric typewriter for user input. Output is generally via the same device though some systems have the capacity to output to other devices such as line printers or card punches. "Conversations" though are generally via the typewriter which for most purposes is quite adequate.

For some special purposes the standard keyboard of a typewriter or teletype is not suited. Klerer and May (1964) have described a modification to a Flexowriter. This modification, mainly to provide a facility for subscripting, was necessary to allow the input of mathematical

expressions as they are normally represented into a conversational system for mathematical analysis.

The Culler-Fried System (Culler and Fried 1965) provides a communication medium in addition to a typewriter - a series of function buttons which have a one to one correspondence with prewritten routines. These buttons are arranged on a keyboard and are "concept oriented" eg. in the arithmetic mode pressing the SIN button would compute the numeric value of the sine of a particular operand but in say the real vector mode the same button would yield the sine function of the operand over its entire range. Such devices are known as controls.

Much attention has been focused on the provision of graphical methods for communication between man and machine. The basic hardware for graphical output is the cathode ray tube (CRT). A display system usually consists of a CRT and some electronic devices that enable a computer to control it. When given a set of co-ordinates by the computer program a simple CRT display will flash the corresponding spot on its face. In this way a complete pattern can be made up.

With suitable attachments a CRT display can also be

used as an input device. A stylus photo cell arrangedent (light pen) can be used to make the CRT serve for the manual input of sketches and diagrams. For this purpose a photo cell isplaced in a small hand held tube - since the photo cell responds only when light from some point of the CRT drawing falls within its limited field of view it can indicate to the computer which part of the drawing its user is pointing at. With an appropriate feedback program a light pen can also be used toenter positional information into the computer. Other stylus input devices that detect the position of the stylus through electric and magnetic field effects serve a similar function.

Though as yet no system equal to a pencil and doodle pad or chalk and blackboard, as used for man/man communication of ideas, has been developed for man/computer interaction several systems such as the RAND tablet (Licklider 1965) enable the computer to interpret human sketches and diagrams without the time consuming manual task of reducing them to a set of numeric co-ordinates.

There is continuing interest in the idea of talking with computing machines. If computing machines are ever to be used for direct top-level interaction then it may be worthwhile, even at great expense, to provide communication

through this natural medium. If the equipment was already developed, reliable and available then it would be used.

Speech production poses less technical problems than the automatic recognition of the spoken text, successive generations of automatic talkers having been developed. Work at the Haskins Laboratory (Licklider 1965) has led to the development of a digital code suitable for use by computing machines that makes an automatic voice utter intelligible speech.

Clark (1968) describes an automatic hypnotising machine in which a logic unit controls a tape recorder that sends out a series of verbal messages to a hypnotic subject. The logic unit receives signals from a button which the subject presses. The system, modelled on the same principles as a teaching machine, simulates the automatic part of the behaviour of a human hypnotist.

The magnetic tape contains the output side of the hypnotic procedures in the form of a human voice. At intervals the subject is requested to press his button. By doing so he provides the input side of the hypnotic procedures.

The subject may for example receive the following

messages:

"Your eyes are very heavy . . . . .

They are so heavy you want to close them . . . . .

They are so heavy they are closing . . . . .

Closing by themselves . . . . .

They are closing . . . . .

If your eyes are closed press the button!

They are closing . . . . .

They are so heavy they are closing . . . . .

Closing . . . . .

They are closing . . . . .

If your eyes are closed press the button!

(and so on)

In this way the subject may press the button early on or only after many such requests. The time taken before doing so is some indication of the "depth" of the subject.

At each request to press the button there is a single pulse (S.P.) marker on the tape. If the button is pressed the tape recorder jumps at "fast - forward" speed from that S.P. to the next double pulse (D.P.). Then it plays the next section of the tape.

The feasibility of automatic speech recognition depends heavily upon the size of the vocabulary of words to be recognised and upon the diversity of the accent with which they are to be spoken. Ninety-eight per cent correct recognition of naturally spoken decimal digits has been demonstrated (Licklider 1965) but as yet much further advance has not been made. For real time interaction a vocabulary of about 2,000 words would probably be necessary. This constitutes a challenging problem.

CHAPTER 3

CONVERSATIONAL MODE PROGRAMS AS A SOLUTION TO THE PROBLEM OF BRINGING THE COMPUTER TO THE CASUAL USER

A common problem in a scientific computing laboratory is that of bringing the computer to the casual user. Many people have computational problems which could be solved by digital computers yet which are still tackled by pencil and paper or desk calculator techniques. The person with the problem to solve frequently has neither the desire nor the time to master a programming language sufficiently to write his own programs and the problems are often of a scale for which time spent in explaining them to a programmer and the subsequent delay while the program is being written compares unfavourably with the time required to solve them by hand. Further, when manipulation of the raw data is required prior to statistical or other analysis several programming sessions may be required before satisfactory combinations are found followed by further sessions to add the statistics suggested by earlier results. The total time is lengthened by the number of computer runs, each of which must wait its turn in the job queue.

One approach to the problem is the provision of programming packages. Provided the specifications for the use of the constituents of such a package is not written in the jargon of the professional programmer, a temptation it seems difficult to overcome, and provided that the corresponding supervisor control statements are either standardised and prepunched or else simplified, a provision which usually requires a major modification to the supervisor system, this approach obviated the need for a personal programmer but however still leaves the problem of possibly numerous computer runs. The conversational mode solution has become more attractive with the provision or expectation of the provision of multi-access facilities and the development of several low cost medium size computers.

Any system designed to create an interface between user and computer must have certain fundamental properties:

(i) It must be easily approachable by the inexperienced user.

(ii) There should be only a minimum of conventions to learn.

(iii) Communication must be kept at a simple level.

(iv) The user must be protected from his own errors.[2]

Several of the existing user oriented conversational programs comply in some reppects to the above requirements, however few do in all. Some systems (eg. MULTI-POP) require lengthy manuals to be read and complicated syntax to be learned, others, in an attempt to be simple, involve the user in boring dialogue (eg. Colin's on-line access system) and such as DOCUS, which is a large complex system, is by no means easily approachable by the inexperienced computer user.

STAT-CHAT is a user oriented conversational program developed by the writer for on-line data analysis which complies with the properties of a user oriented computing system as given above, and in addition, possesses several further merits.

The pre-machine time required to learn the system is minimised by the provision of a simple yet comprehensive manual. The inexperienced user need only be familiar with the few pages of this manual concerned with the preparation of data and operation of the program. The bulk of the manual (see Appendix) is taken up with sophistications of the system designed for the more experienced user.

The use of STAT-CHAT as opposed to conventional batch

processing for data analysis results in a reduction in the number of computer runs necessary to obtain acceptable results. Even without multiaaccess facilities error elimination, which usually requires several computer runs, can be carried out easily and efficiently on-line. It is similarly simple to transform and manipulate a data structure and to process several sets of data during the one computer run.

Two special features incorporated into STAT-CHAT permit the experienced computer user to:

(i) bypass the conversational phase and to treat the system merely as a program package;
(ii) obtain on punched cards a source program to execute the procedures requested in a "conversation".

By virtue of the fact that the bulk of the system is written in FORTRAN it is relatively easy to make changes and modifications and to expand the system if the need arises.

STAT-CHAT issfully discussed in Chapter 4.

CHAPTER 4

STAT-CHAT

STAT-CHAT has been implemented on an I.B.M. 1620 model II computer with 60K decimal digits storage, line printer, card reader/punch, disk file, digital plotter and on-line console typewriter. No multi-access consoles were available and the project was initially regarded as an exercise in preparation for the later provision of this facility. In practice the system is useful and not excessively time wasting when taking into consideration the compilation and rerun times taken by inexperienced computer users.

The general description of the STAT-CHAT system given below is followed by a more detailed account, with accompanying flowcharts, of the techniques employed to handle the conversational phase.

For operation of the system the user is required to be conversant with a manual (see Appendix) in which are contained details of data preparation, a list of the available processing routines and the valid responses in the conversational dialogue. Thus the necessity for long type outs of instructions as in Colin's program (Example 4)

is obviated and the basic conversation kept at a simple level. As the instructions for data preparation are given in the manual the user only requires one computer run arriving at the console with his data deck ready punched whereas in some programs (such as Colin's) the instructions for data presentation are given in the actual conversational phase requiring in the case of the new user a dummy run before any processing can be attempted.

Though the manual is quite large it is not necessary for the beginner to read it all as much of its bulk is taken up with instructions for alternatives to the basic approach. When approaching the system for the first time it is necessary merely to read the section on data preparation and to be aware of the nature of the dialogue.

Three types of program go to make up the STAT-CHAT system (see Figure 1):

(i) a main-line program;
(ii) a conversational control subprogram;
(iii) a series of data processing and statistical subprograms.

All but the conversational program are written in FORTRAN II. The object of this is twofold; firstly to

enable further statistical routines to be easily added to the system and secondly to enable FORTRAN source decks for subsequent execution on the 1620 or some other computer to be punched on request. This latter facility will be discussed again later.

The drawback to several conversational systems, eg. STATPAC, is that they are written in their entirety in an autocode for a specific machine and are "sealed systems" making the addition of new routines difficult and probably requiring the reprogramming of a great bulk of the conversational system. The configuration of the programs and subprograms of the STAT-CHAT system and the use of FORTRAN II as the main programming language enables the addition of new routines to the system library to be made relatively easily. It is similarly easy to delete or replace existing routines.

In order to make economic use of the core store and maximum use of the existing operating system all the statistical subprograms are declared to be local and are placed by the supervisor in the working area of the disks ready to be loaded if and when they are required. On execution the main line program immediately enters the conversational subprogram which for reasonable efficiency is written in

1620 Symbolic Assembly Language. Communication is on a simple level involving no jargon. Requests can be made for particular statistics to be computed and the user is asked to supply the relevant parameters. For example, following a request for polynomial regression the user is asked to supply the order of the polynomial and the variables between which it is to be calculated. An index is set and control is returned to the main line program which calls and executes the requested subprogram by a computed go to statement and then re-enters the conversational subprogram allowing the user to make another request. The user may request the execution of another statistic or the same statistic if different parameters are required or, to terminatedthe program, he can type "NONE".

Several of the subprograms calculate a statistic over all data vectors - the system can handle up to 9 separate vectors of numbers each of equal length and provided that the product of M, the number of vectors, and N, the number of elements in each, does not exceed 1,000. Other statistics however require the user to select pairs of vectors for their execution. A number of syntactical forms are provided for this allowing at a single user request the sequential execution of the subprogram with different data vectors as parameters thus lifting the

necessity for repeated requests for the same subprogram though this can be achieved simply by requesting "SAME". Similarly if the same set of parameterd is to be passed to two subprograms called sequentially the request for parameters can be answered with "SAME" (see Example 8)

The user may, before selecting a statistic, obtain a list of any new routines added to the system and which may not be included in his manual. He is also given the opportunity to assign names to the data vectors. These names along with a title specified with his data block reference the output. After the execution of any procedure which changes the data matrix the user may rename the vectors, similarly, if he requests new data to be read in he may assign names to the vectors in this new matrix.

An important advantage of the system is that it cannot be broken by users' mistakes. Errors and inconsistences areedetected and the user given a warning message and the opportunity to make a correction. A count of the user's errors is kept and if this becomes excessive then the program comes to a premature end. It was felt necessary to include this error trap as, if the user has understood the operating manual then few mistakes should occur.

Two types of error are recognised. Misunderstandings of the system and misspellings of operator names are not treated with equal severity. When the user is asked to request an operation after preliminary checks for "NONE" and "SAME" the input is compared with the names in the library of valid operators. If the name is not found in the library a "hash count" routine is entered. A hash count is made of the input and this compared with the hash totals of the library of valid names. If the input falls within one letter of a valid name the user is asked if he wanted that particular statistic to which he may reply "YES" or "NO". As several of the operators have similar hash totals the hash count routine scan is based on the resemblance of the first letter or group of letters of the input with the library names. This prevents wrong "guesses" by the computer. Spelling slips recognised in the routine are not included in the error count. To the user the machine appears to be endowed withssome form of intelligense being able to recognise spelling slips and make correct guesses at the intended input.

Error messages, when given, are short and to the point and allow the user to correct his mistake. Curt error messages and the separation of spelling errors from logical errors are two advances of STAT-CHAT over several

other programs which give lengthy type outs for errors, usually giving all the correct forms of user input, whether the user has recognised his mistake or not. In STAT-CHAT if a user is uncertain of how to correct his mistake after receiving an error warning he need only type "HELP" to obtain a more lengthy explanatinn of how to make the correction;

As mentioned previously the user may assign names to all the data vectors. These names are however restricted to 8 characters. When choosing a data vector to be passed as a parameter to a subroutine the number of the particular vector in the data matrix is used and not the assigned name as the user may decline to assign names. The names only reference the output which is printed on the line printer with each sheet fully labelled with the user's reference as specified in his data block. Where names have not been assigned to the data vectors results are referenced in the uutput by numbers.

The printer was chosennfor data output to enable larger and more comprehensive printouts to be accomplished. It is hoped however to have an optnon of printer, typewriter or card output in a future version of the program.

The syntax for the sequential execution of the same

subprogram with different data vectors as parameters is of a simple yet comprehensive nature.

The forms of the parameter string are:

(i) 1,2. causes the subprogram to be executed once with vectors 1 and 2 as the X and Y parameters.

(ii) 1,2,3,4. causes the subprogram to be executed three times with vectors 1 and 2, 1 and 3 and 1 and 4 as the X and Y parameters.

(iii) 1,2-4. causes the same sequence of calls as above.

(iv) 1,2+1,3+1,4. causes the same sequence of calls as above.

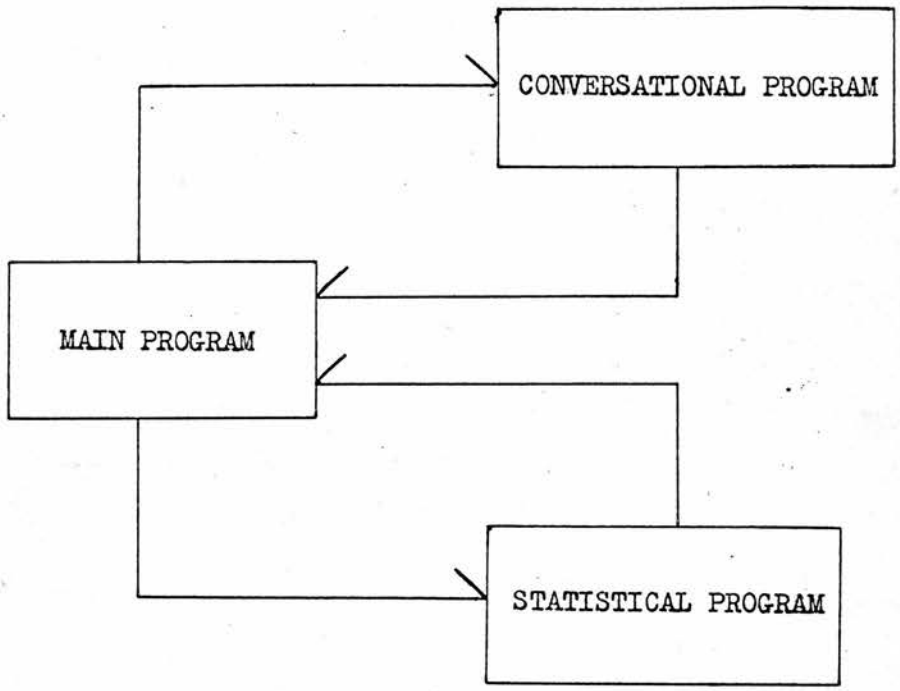
All the above forms can be extended and combined to any extent and to any length not exceeding 30 characters.

Two routines are used to accomplish the sequential execution of a subroutine. The routine which the user's input string first enters checks the string's syntax and validity while the second passes control back to the link program and thence to the subroutine. Indices are set at each call enabling the return to the appropriate place in the conversational program to be made. Parameters are passed via a common area of core store. As the conversational

program is kept in the core store, the subroutine involved in the string execution need only be read down once from the local areas of the disks. Before each return to the link program the contents of any index registers set by the conversational program must be stored away and upon returning back replaced. This storage is necessary as the registers are utilised by several of the FORTRAN routines as well as the conversational subprogram.

As STAT-CHAT was written for data analysis two routines are included for altering the data vectors. These are "COMBINE" which combines two vectors into one by a selected process, and "CHANGE" which transforms a single vector by a selected process. Other options enable the entire matrix to be transformed (see STAT-CHAT manual in the Appendix). It was felt necessary to include such facilities as in data analysis frequently the log or square or such like transform of a variable gives a more useful result than the raw data itself.

Figure 1


CONVERSATIONAL PROGRAM
MAIN PROGRAM
STATISTICAL PROGRAM

Example 8: STAT-CHAT

```
HAVE YOU READ THE MANUAL, IF NOT TYPE NO AND PRESS R/S KEY
     YES
DO YOU WISH LIST OF NEW STATISTICS
     NO
DO YOU WISH TO NAME THE VARIABLES
     YES
VARIABLE 1 =  A
VARIABLE 2 =  B
WHICH STATISTIC DO YOU REQUIRE
     DATA PRINT
WHICH STATISTIC DO YOU REQUIRE
     STANDARD DEVIATION
WHICH STATISTIC DO YOU REQUIRE
CO   CORELATION
DID YOU WANT CORRELATION
     YES
WHICH STATISTIC DO YOU REQUIRE
     REGRESSION
LINEAR OR POLYNOMIAL
     POLYNOMIAL
TO WHAT DEGREE
     3
FOR WHICH VARIABLES
```

```
     A8-, B
INVALID TRY AGAIN
FOR WHICH VARIABLES
     HELP
ACCEPTABLE FORMS ARE
1,2,3,4.
1,2-4.
1,2+1,4-6.
FOR WHICH VARIABLES
     1,2.
WHICH STATISTIC DO YOU REQUIRE
     SCATTER DIAGRAM
NO MORE STATISTICS CAN BE REQUESTED AFTER SCATTER DIAGRAMS
HAVE BEEN DRAWN DO YOU STILL WANT SCATTER DIAGRAMS DRAWN
NOW
     YES
FOR WHICH VARIABLE
     1,2.
END OF JOB
```

Note: for ease of reading user replies have been inset.

A further example of a STAT-CHAT dialogue appears in the Appendix.

DATA INPUT

"A program written to process data can only be expected to produce useful results if the data presented to it are correct in all respects." (Cooper and Whiteside 1963) - for this reason much consideration went into the design of the input routine to make it simple enough for the non-programmer to understand the requirements and yet allowing its modification by the more experienced computer user.

Input is card oriented. Data is normally read by a "read" subroutine which has a fixed format to which the standard user must comply (see STAT-CHAT manual in the Appendix). Several options are available to the more experienced user:

(i) he may, by indicating on a header card which precedes his data, choose a "free format" option. By choosing this option the main "read" subroutine calls another subordinate routine (written in 1620 Symbolic Assembly Language) which reads in the data. This free format routine however requires each variable to be recorddd as a decimal number, ie. with the decimal point punched, and the individual vector elements to be separated by a sign or, at the least, one blank. As in the

main "read" routine the equivalent elements of each vector must be recorded on the same cards.

(ii) He may, again by indicating on the header card, choose the "format card" option. This option calls a sub-routine (again written in 1620 Symbolic Assembly Language) which changes the format specified in the "read" routine to a new format specified by a card preceding the data deck. The data is read according to this new format.

(iii) The experienced user may write his own read routine though with option (ii) above this should never be necessary unless input is to be from a source other than cards.

PUNCHING A PROGRAM DECK FOR FUTURE USE

Provision is made for the user who wished to execute at some future date the same operations as in his "conversation" with the computer. This takes the form of a punched out source deck to call the required subprograms and can be run as a normal batch processing job. As all the subprograms are written in FORTRAN II such a source program is not only compatible with the machine from which it was obtained but also with larger machines (with FORTRAN II compilers) enabling fast execution times for large data sets. To make use of this provision an index must be set in the header card which precedes the data.

When this option is called for as each routine is requested a corresponding code and set of parameters are stored away and when the user terminates his conversation these codes and parameters are passed to a punching subprogram (written in 1620 Symbolic Programming Language). This subprogram first notes the routine to be called then calculates the length of the source program made up of these routines testing if it will fit in core store. If the program will fit in core it is immediately punched out but if not then the number of local records is established and then the source deck and local control cards

are punched. The user has merely to place his data block behind the punch out and then his program is ready to run as a normal batch processing execution. (this feature is discussed in more detail elsewhere)

BYPASSING THE CONVERSATIONAL PHASE

It is possible to bypass all of the conversational mode part of the system and execute subprograms as a normal batch process job. This method of execution treats the library of statistical subprograms as a program package. To make use of this facility the user must specify in a number of parameter cards, which precede the data, codes for the operations he requires and also any additional parameters which these operations require.

The program to carry out this function is, as is the conversational mode program, based on a computed go to statement. The index for this statement is read in from the parameter cards.

To ensure maximum operating efficiency the subprtggams, as in the conversational mode program, are stored in the local areas of the disks and only read down into core if and when required.

This provision is intended for the user who has become familiar with the system in the conversational mode and who wishes merely to do routine executions of one or more of the system subprograms.

SUBPROGRAMS WITH SUBROUTINES

Due to the restrictions of the operating system of the 1620 on programs held in the local areas of the disks certain difficulties arise when subprograms are included in the STAT-CHAT library which themselves require to call upon other subprograms. The operating system does not allow local subprograms to call each other and a subprogram called by any local subprogram must be loaded into the main core store. However if several of the local subprograms call upon different subroutines, all of which must be loaded in store, then the core store,(only 60,000 characters) is soon utilised to the extent that only very small main line subprograms can be loaded. A balance must be kept between the amount of core store utilised by subroutines and the size of the main line subprogram which has to be loaded. As the number of subroutines increases, the allowable size of a subprogram decreases. To conserve the maximum amount of core store it was necessary to make all of the subprograms in the system (with the exception of the "read" routine) selfcontained and only calling on 1620 library routines (such as square root, log, etc.). Some difficulty arose however over the subprogram designed to plot scatter diagrams on to the digital plotter. This routine called upon a 1620 plotting routine, two 1620 character drawing

routines and a sorting routine. It was found impractical to arrange these routines in any efficient way which was economical of storage. To allow the inclusion of the scatter diagram in the STAT-CHAT library the call link feature of FORTRAN II was utilised. This however does not allow any further subprogram executions to be accomplished after a scatter diagram has been drawn. When scatter diagrams are required a call link call is made to a dummy program which stores parameters for up to 10 scatter diagrams and loads into the core store, to replace the calling program, the subprogram to draw the scatter diagrams and its associated subroutines. When all the required scatter diagrams have been drawn the STAT-CHAT program comes to an end.

A similar arrangement and set of constraints must be made for other subprograms of a like nature.

STATISTICAL SUBPROGRAMS

The statistical subprograms are written in FORTRAN II single precision arithmetic. Several are modifications of programs from the Goddard Science Institute Biostatistical Programming Package (Shannon and Henschke 1967) the theory of which are based on Ostle (1963). The statistical theory of the others is from the same source.

Each subprogram has its own output routine and is thus a self contained unit. Much thought went into the design of the output format for each routine to enable a full set of results to be presented in an easily comprehensible form. If however results fall outwith the range of the format specified by a subprogram then the operating system will indicate errors. The corrected results will still be given but the format of the output layout will be disrupted. If the data is handled correctly then these errors should not occur. Where necessary, subprograms have built in routines to test the validity of the data which they are given to compute and give indications if such data is unsuitable.

Though STAT-CHAT is primarily a facility for the automatic application of routines rather than for their

writing and compilation, it can still be regarded as providing a form of programming language, though of a very restricted nature and of a very high level allowing the user with one command to execute a complete program of instructions. The user's "program" is made up of combinations of calls to various STAT-CHAT subprograms.

The three components of a programming language as recognised by Raphael (1966) are present in the vocabulary of STAT-CHAT. The elementary program statement is a requirement, the user knowing the result of his actions but not how this is achieved. The language statements are linked in a loop, control always returning to the user after the execution of his requirement. Data input is catered for as previously described by an independent subprogram which the user can call upon in the same way as he does for any of the other available subprograms.

The system, as it stands, operates under the 1620 monitor system and is handled as a normal job. At each execution the subprograms are loaded from the permanent areas of the disk files into the local working areas of the disks and the core store. As the number of subprograms which require to be loaded is quite large the operation takes several minutes to complete. To speed up the process

a separate disk pack has been utilised to hold all the STAT-CHAT programs. At execution time the subprograms are copied en bloc into their respective locations in core or local working areas, thus cutting down on loading time. Under this method the computer operated as a one disk system instead of the normal three. The one disk operating system is provided as an alternative to the Monitor system. Using this alternative load time is reduced by a factor of 30/1.

THE CONVERSATIONAL SUBPROGRAM

The flow of information through the conversational subprogram is shown in Figure 2. This subprogram as previously mentioned is written in 1620 Symbolic Assembly Language.

Operations are of three types:

(i) declarative;

(ii) imperative;

(iii) control.

Declarative operations are used to assign core storage areas for input, output and working. Imperative operations specify the instructions to be executed and, except for certain macro-instructions, a one for one correspondence exists between a symbolic code and a machine language instruction. Control operations merely provide the programmer with control over portions of the assembly process.

A number of constants and variables are declared in the FORTRAN calling program to be in common storage and, in order to maintain consistency, the appropriate storage locations in the conversational subprogram must be assigned. Held in common are the data matrix, the number of rows

and columns of the data matrix, an array holding the names of the data vectors, the index for the computed go to in the main program and a series of constants to hold parameters to be passed to the FORTRAN subprograms.

The dictionary of valid user replies and the series of "canned" messages are stored by declaring alphameric constants. Collecting these constants under symbolic addresses enables loops to be easily formulated when more than one declarative is to be utilised in an operation.

To permit subroutines to be readily used within the conversational subprogram the linkage between the main program and the conversational subprogram does not utilise the BB (Branch Back)instruction available in 1620 Symbolic Programming Language. On entry to the conversational subprogram the return address in the main program is stored away and on exit an indirect branch is made to the storage address which results in a branch back to themmain program and the execution of the next instruction in that program. Subroutines internal to the conversational subprogram use the normal 1620 Symbolic Programming Language subroutine linkage instructions.

On each entry to the subprogram a number of tests

must be performed to ensure that the conversation will continue in a logical manner. When a FORTRAN subprogram is being executed with a "string" of parameters an index is kept toiindicate this and also as a pointer to the next set of parameters to be passed to the subprogram. This index must be tested at each call of the conversational subprogram as must the indices associated with the "change" and "combine" procedures. The index (I) of the computed go to statement (initially zero) is also examined as is another index set when data is read in. This latter index has different values depending on whether or not a data block has been previously read (see Figure 3).

A user reply to a STAT-CHAT question is checked by comparing it with the appropriate words in the dictionary of valid user inputs. This is achieved through the compare instruction available in the 1620 Symbolic Programming Language. At each reply point (except the very first) the user can ask for help from the system by typing "HELP". Each user input is checked for "HELP" before the dictionary is scanned. If the word "HELP" is detected an index, the value of which is dependent on the point in the program where "HELP" is detected, is passed to an internal subroutine which, based on the value of the index, types out appropriate instructions for the user. The program continues with a

repeat of the question to which the user previously answered "HELP".

Where the user's reply is invalid a subroutine, functioning in a similar manner to that which handles "HELP", is entered which gives appropriate error warnings. This routine keeps a count of the errors and the severity of the warning messages increases with the count.

If the user wishes to assign names to his data vectors then control passes to a routine which will read in 8 letter names and store them in a common area of core store so that they may be accessed by all the FORTRAN programs. The floating point word length of the FORTRAN programs is 8 so that an 8 letter alphameric word requires two FORTRAN floating point words. Names must be stored in the form:

X̄XXXXXXX̄0̄0X̄XXXXXXX̄0̄0

eg. if the name to be stored is ABCDEFGH then the following sequence of digits will be placed in core store:

4̄14243444̄0̄04̄5464748̄0̄0

where the letters A-H are represented by the numbers 41-48.

This method of storage ensures compatibility with FORTRAN A type format.

If no names are assigned by the user then the numbers 1 - M (where M is the number of vectors in the data matrix) are stored.

An index register is utilised to cope with the allocation of addresses for the names (see Figure 4).

The schematic flow chart shown in Figure 5 outlines the portion of the conversational subprogram which deals with a user's request for a particular operation to be performed. After the question "WHICH STATISTIC DO YOU REQUIRE" has been put by the program an input area is cleared by transmitting into it a field of alphameric blanks and then, into this field, the user's input is read.

After a test for "HELP" a scan is initiated which searches through the library of available statistics. Before the scan is begun the value 1 is placed in I (the index of the computed go to in the main program) and then, as the scan proceeds this value is incremented so that when a match is found the value of the index to call the

appropriate subprogram is already set up. Before I is reset to 1 the input is checked for "SAME". If a match is found then a branch is made to a check routine which compares the value in I (which is the index of the subprogram executed immediately previous to the present call of the conversational subprogram) with a list of values for which the request "SAME" is valid. If no match occurs then an error is indicated. A match results in a branch to the routine which obtains parameters to be passed to the subprogram. When a match is found in the scan routine a branch is made either to a return routine which returns control to the main program to allow the requested subprogram to be called and executed or else, if the requested subprogram requires additional information, a branch is made to a routine to obtain parameter values.

If no match of the user's input can be made with a name in the library of operations then, before an error is indicated the "hash count" routine is entered. The hash count routine sums the numeric representation of the characters in the input area (which was filled up with blanks before the user entered his input) and then attempts to match the total with one of the hash totals of a valid operation. These latter totals were pre-calculated and entered into the program as constants. As several

hash totals are similar the match of hash counts is not attempted until the first letter, or in some cases group of letters, of the user's input matches those of a valid name. When such a match is found then the hash total is compared with that of the "match" name and if the totals are within 50 of each other (ie. there must only be about one letter of difference in the two names) then that name is transmitted into a preformed output format so that the message,

"DID YOU WANT XXXXXX"

(where XXXXXX is the matched name) can be typed. The appropriate value of I is also set up in the common area so that if the user replies "YES" to the above question the program can continue as though a match had been found in the initial scan. If no match can be found in either of the above routines then control passes to the error routine and thence back to ask for another request. Similarly if "NO" is the reply to the question,

"DID YOU WANT XXXXXX"

the user is asked to make another request.

If either "COMBINE" or "CHANGE" are requested by the user then a branch is made to the appropriate routine which asks which of the procedures of these routines is required. The checking routines for these libraries operates in

exactly the same way as do those described above which check the main statistic library. The response "NONE" to the question,

"WHICH COMBINE ROUTINE DO YOU REQUIRE"

and "WHICH CHANGE ROUTINE DO YOU REQUIRE"

causes a branch to allow the user to rename the data vectors. This is achieved by using the naming routine as previously described.

Several routines handle the input of parameters which a user requires to pass to a particular subprogram. Where only one parameter is to be given (eg. the degree of a polynomial, the number of intervals for a histogram) a series of compare operations check if the user's input is within the allowable range. In order to check the syntax of a parameter string ie. when the same subprogram is to be executed with more than one set of vectors as parameters, a 3 part routine is utilised. Part 1 scans the first character of the input for the sequence:

DIGIT COMMA DIGIT

and then checks the next character for:

(i) a full stop

(ii) a comma

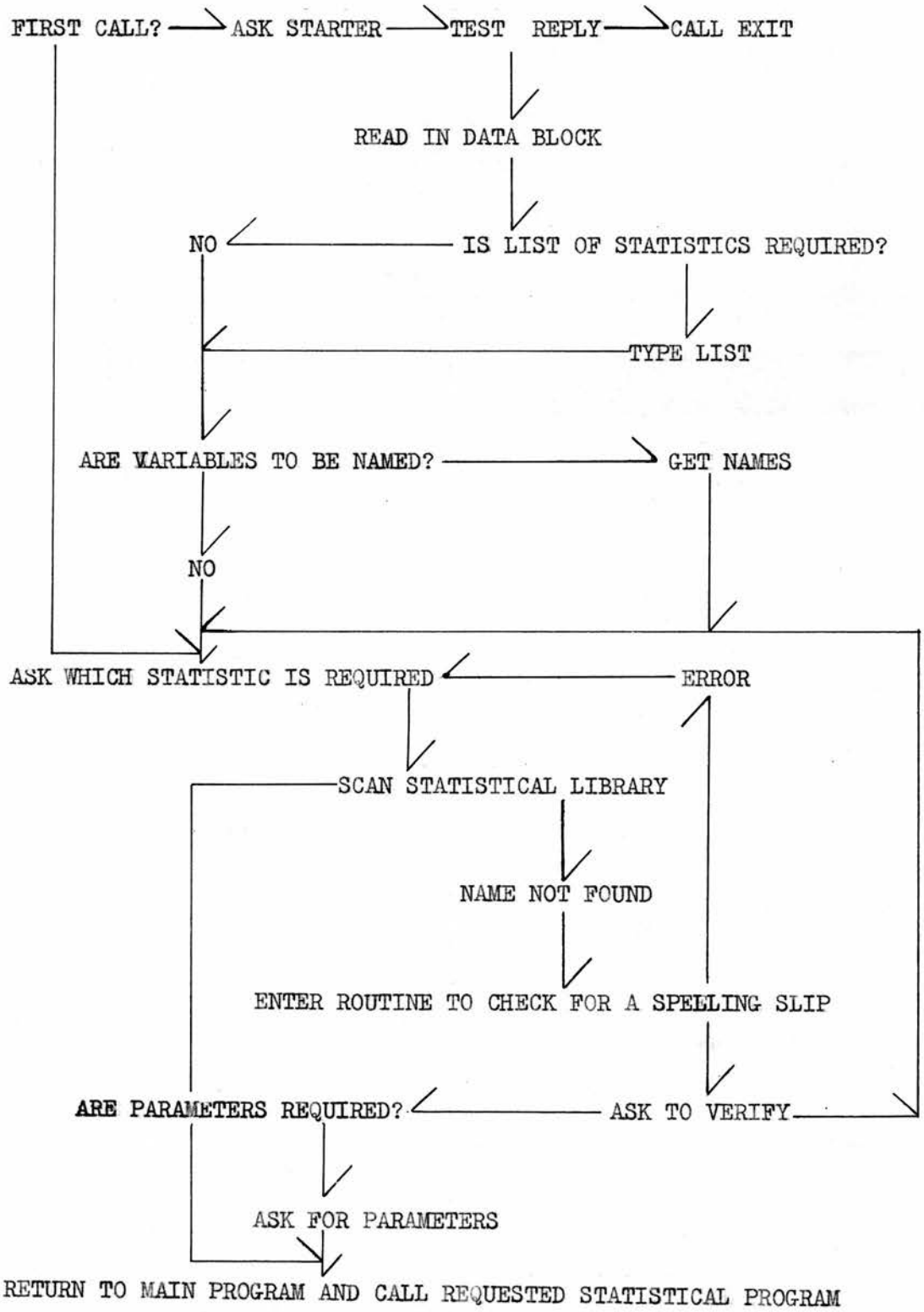
(iii) a dash

(iv) a plus

If (i) or (ii) is found the "comma" routine is entered to cope with this particular type of string. If (iii) is found the "dash" routine is entered and where (iv) is regognised a branch is made back to part 1 of the checking routine. In each routine the input characters are examined individually and tested for validity by comparison either with an operator (ie. a + or -, etc.) or with M (the number of data vectors specified) (see Figure 6). After the string has been checked and found valid then the executinn routine is entered. This routine scans the input string in a similar way to the checking routine but as each pair of parameters is recognised they are transmitted into the common area, a return index set and a branch made to exit from the subprogram and return to the main calling program thus allowing the requested subprogram to be executed. The return index ensures that a return to the correct place in the conversational program will be made (see Figure 3). When the "change" or "combine" procedures are selected additional indices are required to be set to ensure return.

The conversational subprogram hinges on the operation of compare and branch instructions which serve to check the user's inputs and guide him through a series of sub-routines which ensure the successful execution of a chosen operation.

Figure 2


FIRST CALL?
ASK STARTER
TEST REPLY
CALL EXIT
READ IN DATA BLOCK
NO
IS LIST OF STATISTICS REQUIRED?
TYPE LIST
ARE VARIABLES TO BE NAMED?
GET NAMES
NO
ASK WHICH STATISTIC IS REQUIRED
ERROR
SCAN STATISTICAL LIBRARY
NAME NOT FOUND
ENTER ROUTINE TO CHECK FOR A SPELLING SLIP
ARE PARAMETERS REQUIRED?
ASK TO VERIFY
ASK FOR PARAMETERS
RETURN TO MAIN PROGRAM AND CALL REQUESTED STATISTICAL PROGRAM

KEY TO FIGURE 3

ENTRY INTO CONVERSATIONAL SUBPROGRAM

1: Store index registers and switch on band A.

2: Test if "string" index is set.

3: Branch to get next parameters in the string.

4: Test if new data block been read in.

5: Branch to routine to allow data vectors to be named.

6: Test if first data block been read in.

7: Branch to routine to give list of new statistics.

8: Test if "combine" index set.

9: Branch to ask if another "combine" procedure is required.

10: Test if "change" index set.

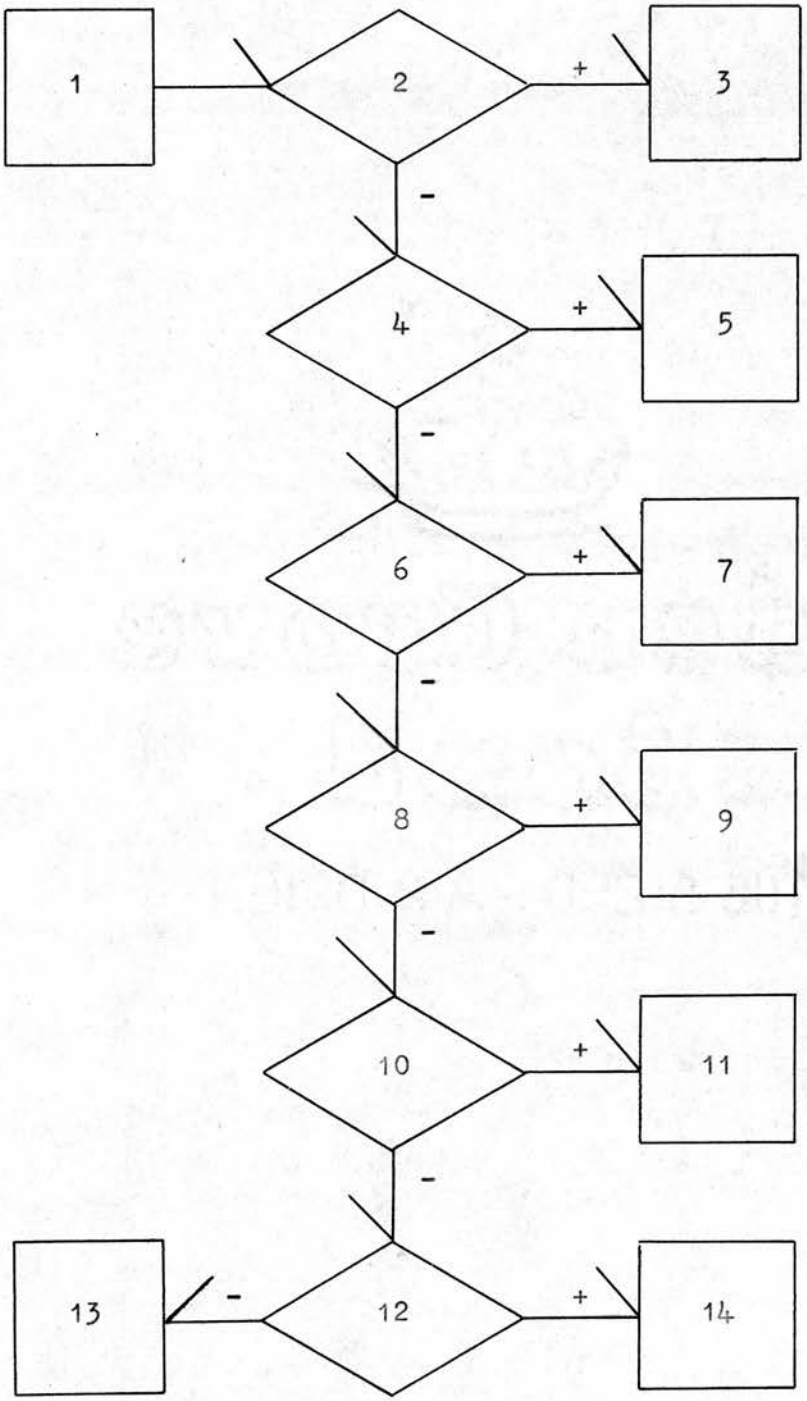
11: Branch to ask if another "change" procedure is required.

12: Test if any subprograms have been executed.

13: Ask starter question.

14: Ask for another selection.

Figure 3



KEY TO FIGURE 4

ROUTINE TO SET UP VECTOR NAMES

1: Load index register.

2: Ask for vector name.

3: Clear input area.

4: Read in name.

5: Transfer first 4 characters to common area.

6: Transfer 00 to common area.

7: Transfer next 4 characters.

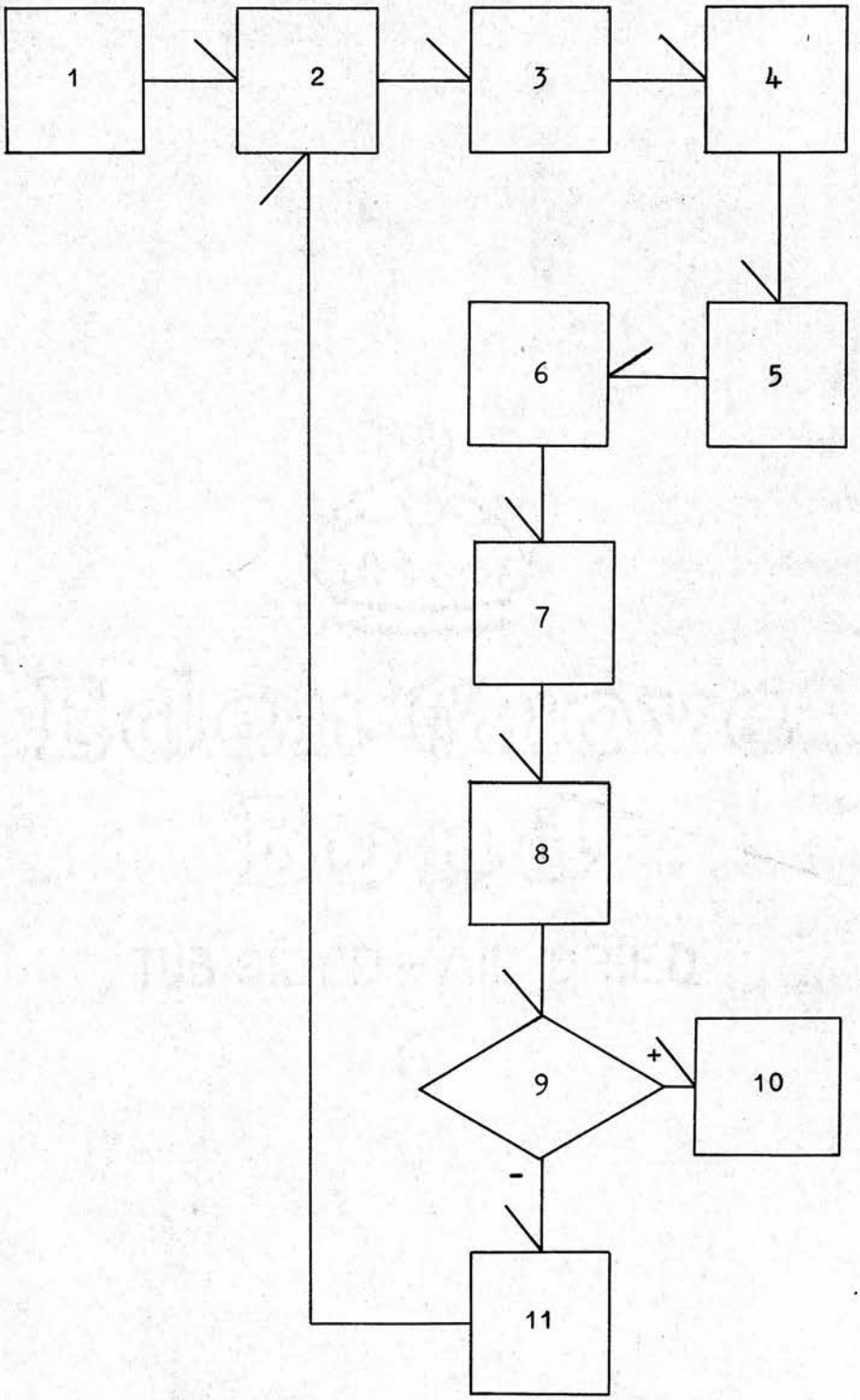
8: Transfer 00.

9: Test if all vectors have been named.

10: Branch to ask which statistic is required.

11: Adjust address register and count if the vectors already named.

Figure 4



KEY TO FIGURE 5

REQUEST ROUTINE

1: Clear input area.

2: Ask which statistic is required.

3: Read reply.

4: Test for "NONE".

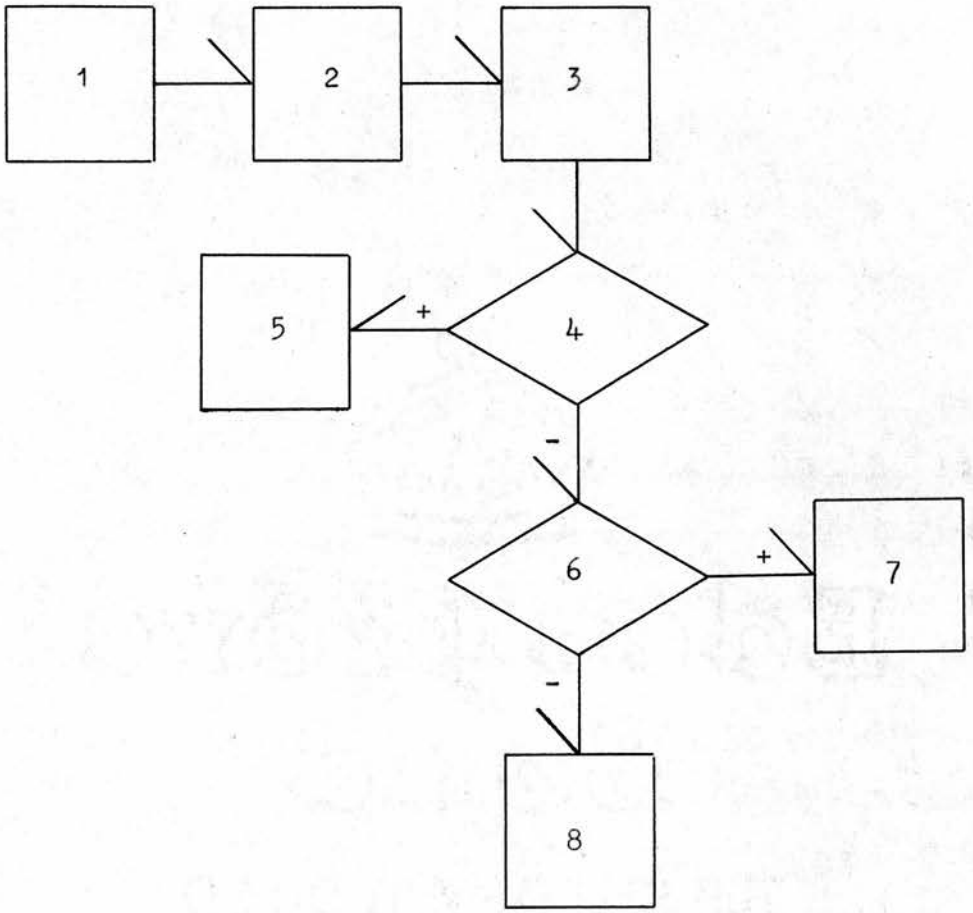
5: Set I=1 and branch to exit routine.

6: Test for "SAME".

7: Branch to routine to test if valid.

8: Branch to routine to check if name is valid.

Figure 5



KEY TO FIGURE 6

SYNTAX CHECKER

1: Load index register which scans input string with 0.

2: Is character a digit not greater than M.

3: Error.

4: Increment register.

5: Is character a comma.

6: Error.

7: Increment register.

8: Is character a digit not greater than M.

9: Error.

10: Increment register.

11: Is character a plus.

12: Increment register.

13: Is character a comma.

14: Branch to "comma routine".

15: Is character a dash.

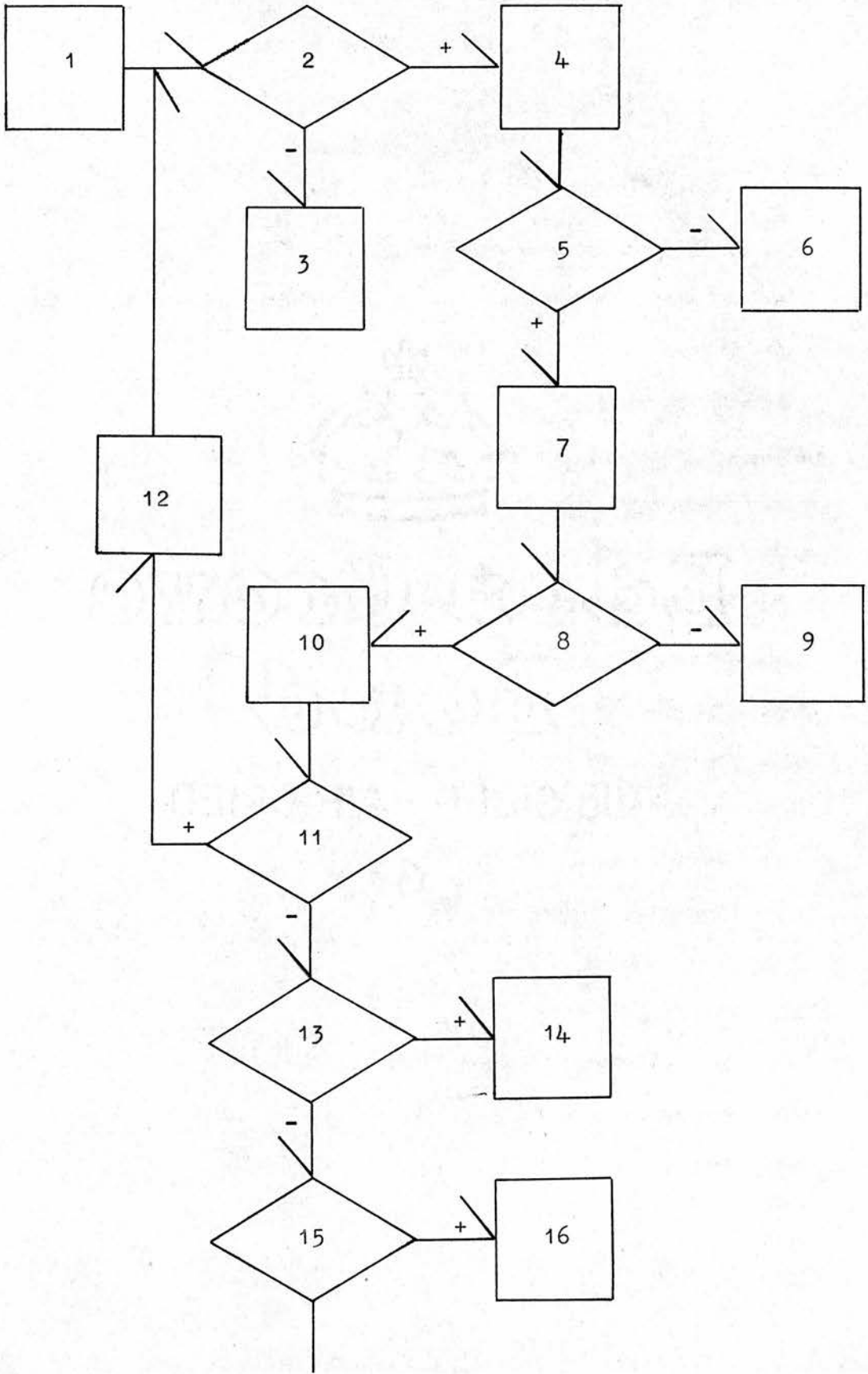
16: Branch to dash routine.

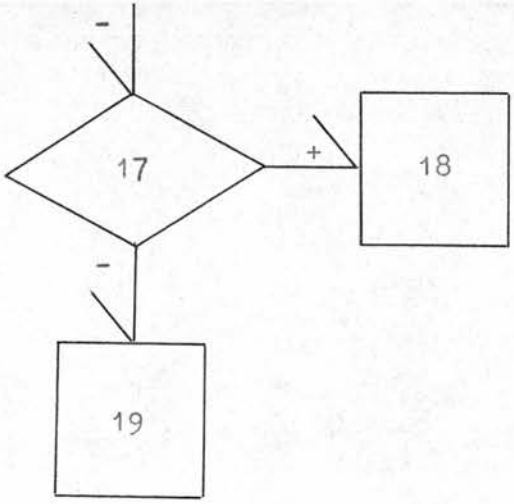
17: Is character a stop.

18: Branch to execute routine.

19: Error.

Figure 6




-
17
+
18
-
19

KEY TO FIGURE 6(A)

COMMA ROUTINE

1: Increment register.

2: Is character a digit no greater than M.

3: Error.

4: Increment register.

5: Is character a comma.

6: Is character a stop.

7: Branch to "execute" routine.

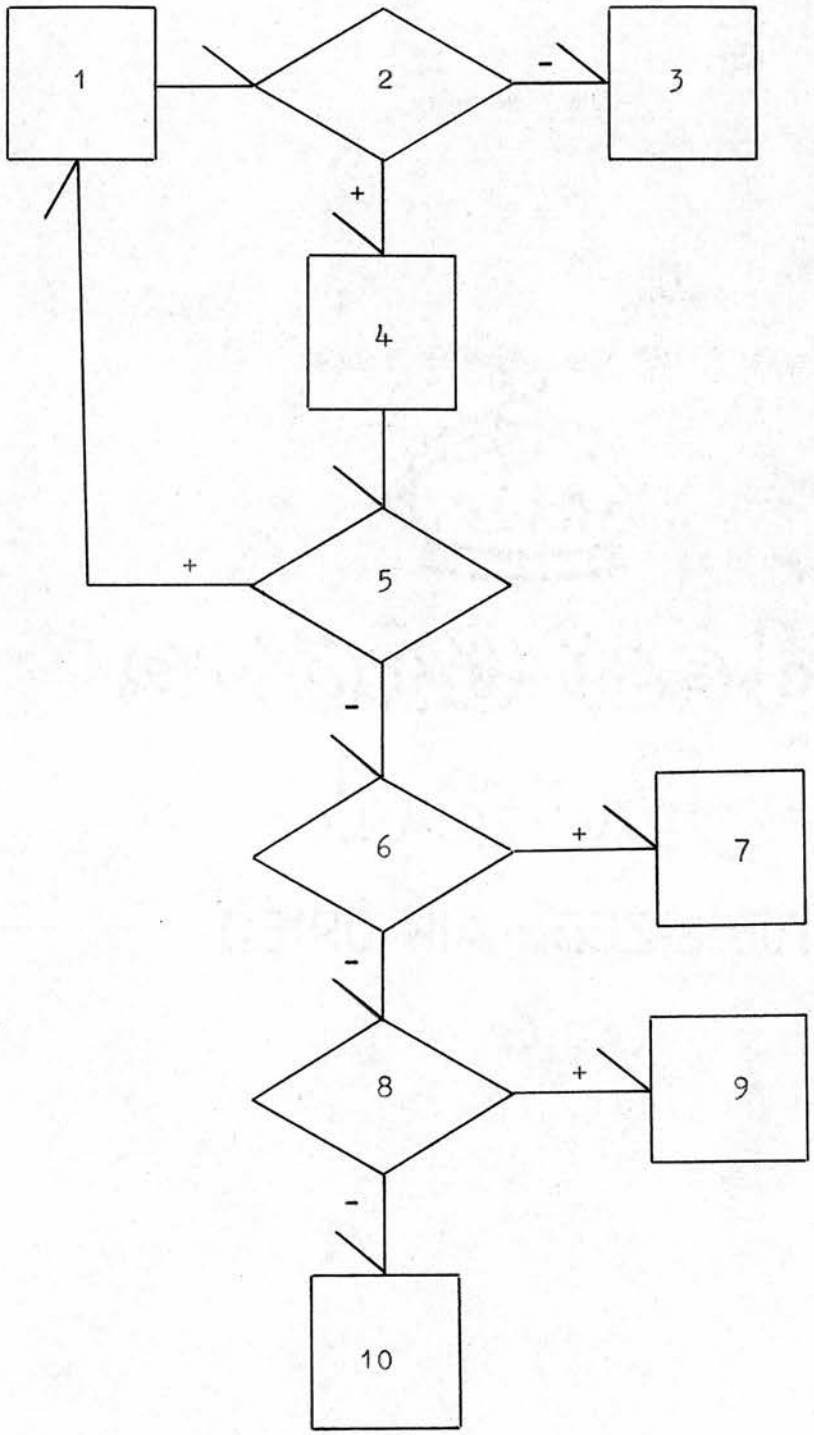
8: Is character a plus.

9: Branch to 12 (Figure 6).

10: Error.

Figure 6(A)

COMMA ROUTINE



KEY TO FIGURE 6(B)

DASH ROUTINE

1: Increment register.

2: Is character a digit greater than the last digit found but not greater than M.

3: Error.

4: Increment register.

5: Is character a stop.

6: Branch to "execute" routine.

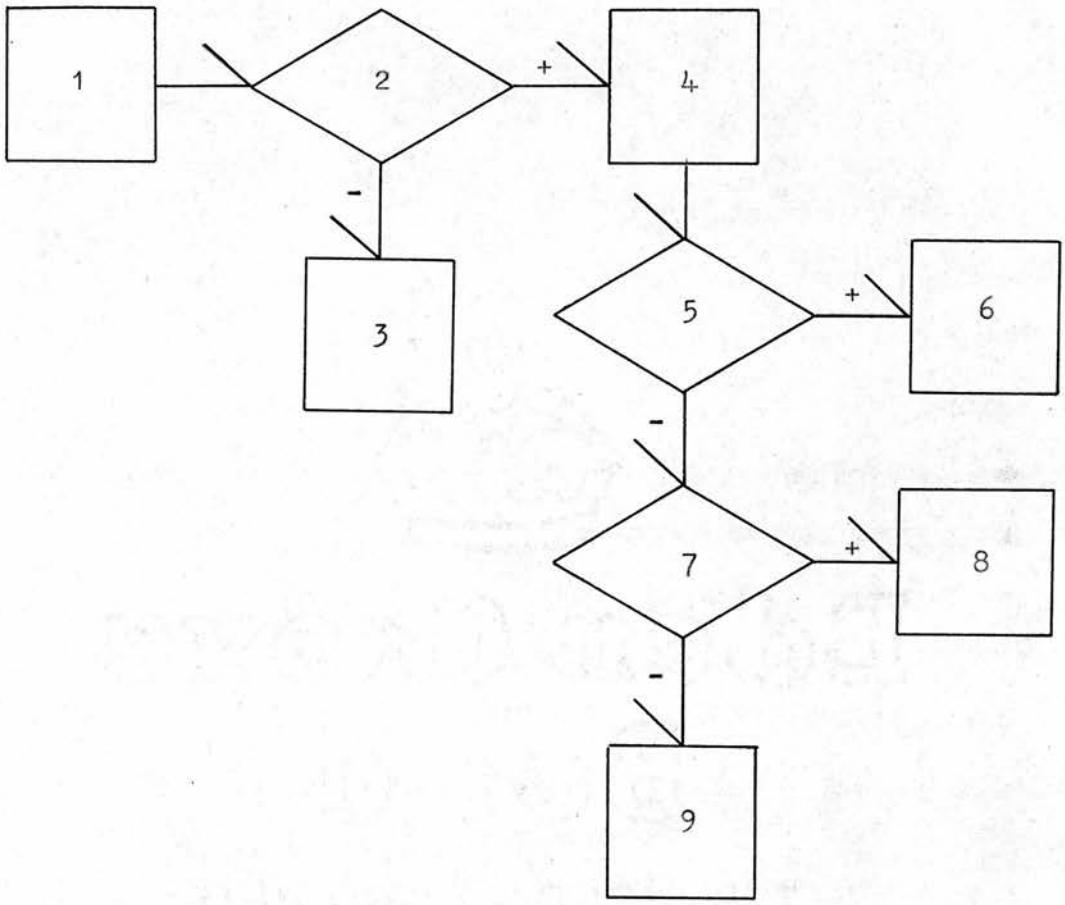
7: Is character a plus.

8: Branch to 12 (Figure 6).

9: Error.

Figure 6(B)

DASH ROUTINE



THE PROGRM SUBPROGRAM

As previously mentioned STAT-CHAT has the facility to provide the user with a FORTRAN source deck to execute the subprograms requested in his conversation. This is achieved by a Symbolic Programming Language subprogram named PROGRM. The flow chart for this subprogram is given in Figure 7.

If a user indicates, by punching a digit in the appropriate field of the data header card, that he requires a FORTRAN source deck to be punched then the indices of all the subprograms he requests in his conversation along with the parameters passed to them are stored away by the main line program. When he either terminates the conversation (by requesting NONE) or asks for a new block of data to be read inthen the PROGRM subprogram is called repeatedly until all the stored indices and parameters have been passed to it. This is accomplished through a common area of core store. After the execution of this calling sequence the main line program either calls exit terminating the computer run or else, if the user requested new data to be read in this operation is performed.

On each branch to PROGRM the index passed is tested.

If this is not a terminating index (ie. either 1 for call exit or 14 for read new data) then a look up is initiated. This look up scans a stack of indices previously passed to PROGRM. If the index is not found it is added to the stack (stack 1) and the next part of the program - the store routine - is begun. If the index is found in stack 1 then an immediate jump is made to the store routine.

The store routine places the index on another stack (stack 2) and stores the parameters passed to the subprogram with the index on equivalent levels in a series of parameter stacks.

If a terminating index is found then an add routine is entered. Stored in PROGRM are the lengths of all the subprograms in the STAT-CHAT system and thus by examining the indices storedoon stack 1 and then adding the length of the corresponding subprogram to a base value the total length of the user's requested source program can be estimated. The add routine carries out this operation and tests whether or not the program will fit in core (60,000 characters) without requiring subprograms to be declared as local.

Having determined the size of the source program the punching routine is then entered. The monitor control

cards and the program header cards are first punched, then the main program (see Example 9 given below). The index values stored in stack 2 are examined in turn and a corresponding name taken from a list of subprogram names. This name along with parameters taken from the parameter stack are set up in a card field to form a call statement to the appropriate subprogram and a card punched. If the subprogram is to be declared as local then the name is also set up in the field of a local card. This process continues until the stack is empty upon which the program terminator cards, and if required local declaration cards, are punched. Control then returns to the main line program.

Example 9: A FORTRAN Source Deck

The following program was obtained to execute the four procedures:

(i) DATA PRINT

(ii) MEANS

(iii) CORRELATION

(iv) Linear REGRESSION of variable 2 on variable 1

requested by a user during his "conversation".

```
$$JOB                                                    (1)
$$FORX                                                   (2)
      DIMENSION X(1000),A(18),T(80),IP(2,2)              (3)
      COMMON A                                           (4)
      READ 1,(A(I),I=1,18)                               (5)
    1 FORMAT (9(2A4))                                    (6)
   10 FORMAT (1H1,80A1//)                                (7)
      CALL MCREAD (X,M,N,ICARDS,T)                       (8)
      PRINT 10,(T(I),I=1,80)                             (9)
      CALL MCPRNT (X,M,N)                               (10)
      PRINT 10,(T(I),I=1,80)                            (11)
      CALL MCMEAN (X,M,N)                               (12)
      PRINT 10,(T(I),I=1,80)                            (13)
      CALL MCCORL (X,M,N)                               (14)
      PRINT 10,(T(I)I=80)                               (15)
      CALL MCREGS (X,M,N,1,2)                           (16)
      CALL EXIT                                         (17)
      END                                               (18)
$$$$                                                    (19)
```

Cards (1), (2) and (19) are Monitor control cards.

Statement (5) reads the names of the data vectors.

Statement (8) calls the subroutine which reads in the data block.

Statements (10), (12), (14) and (16) call the appropriate

subprograms requested by the user.
Statements (9), (11), (13) and (15) cause a new page to be taken for each set of results and the title, as specified by the user with his data, to be printed.

The core storage capacity of the 1620 II computer is 60,000 characters. Of these locations 7,801 are taken up by the system control programs leaving 52,199 locations for user programs. The STAT-CHAT main program occupies 5,770 locations, the conversational subprogram 16,150 and another 6,340 locations are filled by subprograms which require to be loaded into core store. The common area of core store extends from 59,999 down to 49,685 thus 13,625 locations are available for FORTRAN subprograms. In the present implementation this has so far proved to be sufficient.

STAT-CHAT has been frequently used as a demonstration program and has successfully introduced computing to numerous research scientists (in such University departments as Botany) who, previous to an introduction to STAT-CHAT, had never contemplated processing experimental data by computer. STAT-CHAT thus appears to have achieved its

aim in bridging the gap between the scientist with no programming knowledge and his utilisation of the computer's calculating powers.

Though the subprograms of STAT-CHAT are for the computation of statistics the methods employed are amenable for the conversational execution of varying types of subprogram and though developed on a single program machine it is hoped that the experience gained will be useful in the implementation of conversational systems for use with multi-access systems.

KEY TO FIGURE 7

THE PROGRM SUBPROGRAM

1: Is index a terminator?

2: Is index stored on stack 1?

3: Store index on stack 1.

4: Store index on stack 2.

5: Store parameters.

6: Calculate length of source program.

7: Will program fit into core store?

8: Set up monitor card to indicate number of local records.

9: Punch monitor cards and program header cards.

10: Is stack 2 empty?

11: Are local cards required?

12: Punch terminating cards.

13: Punch terminating cards and local cards.

14: Set up name and parameters in card field.

15: Punch card.

16: Is subprogram to be declared local?

17: Set up name in local card field.

Figure 7

PROGRM SUBPROGRAM

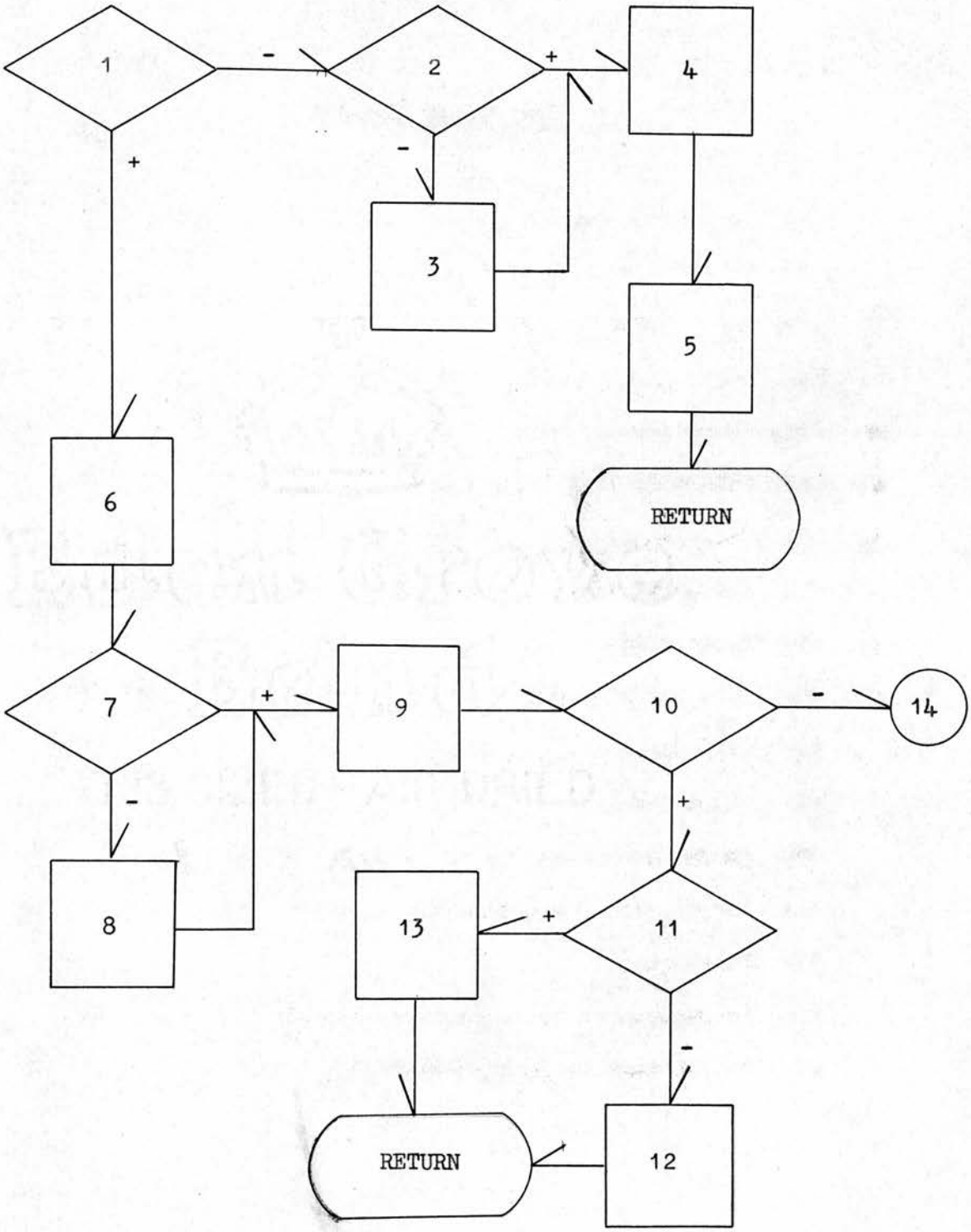
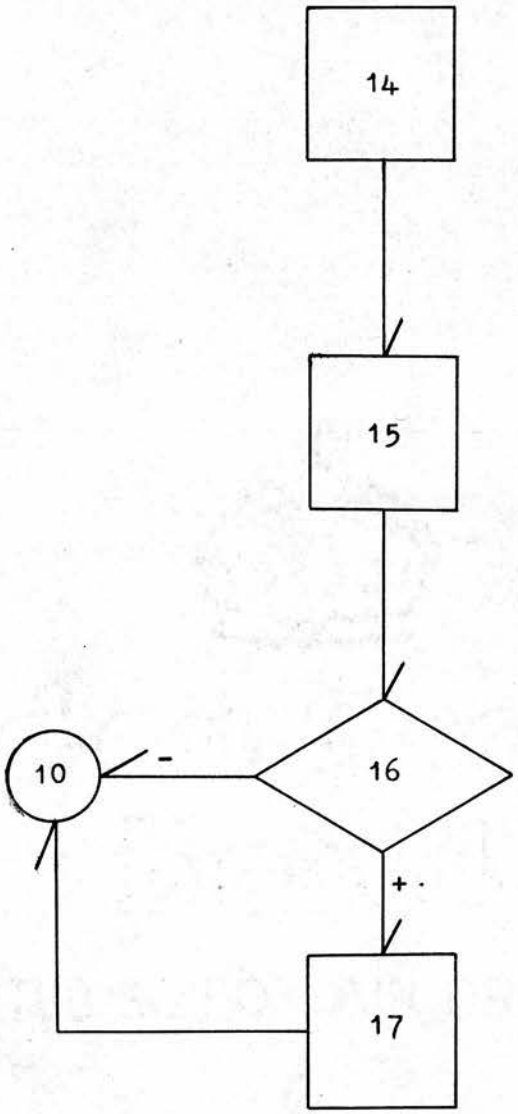


Figure 7 (continued)



CHAPTER 5

DISCUSSION

The goal of all conversational mode programs is to permit the user to have direct communication with an operating system. The efficient running of any such program depends on the subsystems which make it possible which must be so designed as to minimise library searching when combining available procedures for special applications.

The conversational system must be programmed to carry out a meaningful dialogue. Though not necessarily generating ad lib responses the conversation can be made meaningful even with a limited vocabulary of understandable input. Responses can be varied depending upon the previous dialogue, the sophistication of the user's questions and responses, and the results of the calculations.

The most important single aspect in the design of any conversational procedure is the form and language used for communication. In general a balance must be struck between English prose and a cryptic code to minimise the demands on the user in terms of deciphering and typing. The most utilised form of conversation consists of questions

and answers with occassional statements, the latter being for informative purposes. Having decided upon the language of communication the hardware on which it is to be implemented must be chosen. Many conversational systems are based for reasons of speed and ease of comprehension on visual displays (eg. DOCUS, STATPAC). A strong disadvantage of these systems however is that they are non-recording and no permanent record of results is readily made available. If a printing or typing device is utilised then a complete transcript of the user/machine dialogue is recorded. Not only do display systems have the disadvantage of being non-recording but they also impose restrictions on the format of the output this being determined by the dimensions of the display screen which are generally quite small. Display oriented systems are, however, of much value where results need not be recorded and in situations where other forms of input/output device are not suitable eg. sketching (Schmedel 1968).

Conversational programming brings about a change in the working conditions of the programmer. These changes in the process of programming are the result of the system surrounding the programming language.

Where a conversational language is designed for the

formulation of programs such as is JOSS it must provide concise and powerful statements that enable a dialogue between user and program to be entered and changed rapidly. The process of editing code is considered by some to be the heart of a good on-line system. The ability to rapidly modify a program at the console is an important one for conversational users and various schemes for modifying programs have been developed. These vary from techniques, as in JOSS, where numbered lines are changed or deleted to more advanced display oriented systems where editing is done by context.

With the capacity for entering and editing code there must also be a compiler or interpreter to allow execution to be made. A rapid compiler is required. Techniques for speeding up compilers include limiting them to "one pass" linking at load rather than compile time, "incremental" compilers where single statements can be compiled and added to an existing program and the maintenance of two compilers, one for program checkout and another for when the program reaches the production stage. User/ compiler interaction is valuable allowing the compiler to query the user regarding what it considers to be error conditions and permitting the user to change the program before compilation is complete. This interaction could be extended to include questions

that would aid the compiler to produce a better code.

On-line use of interpreters has proved to be of some value. Interpreters can find many errors during execution time that are difficult or impossible to detect immediately with a compiled program.

The ability to detect and correct an error while working on-line is one of the advantages of conversational computing. The successful on-line debugging system must permit the inspection and modification of program and data to be made simply and concisely.

Using an on-line system such as JOSS or the E.S.I. an appreciable compression in time is possible. The entire process from coding to checkout can be repeated within a short space of time whereas using conventional computing systems the same process could, depending upon the turn round time of the computing laboratory, take several days or even weeks.

Schwartz (1965) does not recognise as programming languages the language of JOSS, the Culler-Fried System and others which are exclusively for use at on-line consoles regarding such systems as attempts to assist the nonprogrammer.

These languages are however for the formulation and compilation of programs and possess Raphael's properties of a programming language and thus, in the opinion of the writer must be regarded as such whether or not they are of general application.

In systems such as the Lincoln Reckoner, where pre-programmed routines are available, conversational usage holds several advantages over batch processing techniques. The user can approach the console without a complete pre-formed plan of what, exactly, he is going to do and is thus able to direct the usage of the computer to explore further any interesting result which may arise. This interaction between man and computer, made possible by the conversational approach, is ideal in such as data analysis where, as pointed out by Tukey and Wilk (1966) the choice of steps in the analysis may depend upon the result of those preceding.

The memory capabilities of the computer can be made use of, not only to store program instructions but also computed results. If the results of each step in an analysis program are stored away then it is possible to use them as an input for some other step. This ability is however only of use where either the memory capacity of the machine

is large or the data sets to be dealt with are small, though the required storage space could be curtailed by storing not all but only some of the results, these being indicated by the user. The latter approach is the one adopted in the Lincoln Reckoner.

In general, user oriented conversational systems involving the automatic application of various prewritten routines do not utilise the program storage capabilities of the computer. However a few do. In such as the Lincoln Reckoner chains of instructions can be stored and formulated into blocks, similar to procedures of ALGOL, given names and executed as though they were but a single instruction. The STAT-CHAT system has the capacity to store only one depth of instruction in the conversational mode though, as previously described, a complete program can be stored in the form of a FORTRAN source deck.

As detailed by Licklider (1960) man excells in the heuristic phase of problem solving while the computer does so in the execution of implicit instructions. THough conversational on-line usage the human problem solver can direct the computer to explore his theories and hunches which, by normal batch processing would not be possible. In the general run of computer applications the heuristic and

algorithmic aspects of problem solving are almost wholly separated. The heuristic contributions are generally made by the user before the problem reaches the computer where the algorithmic phase controls.

Though, as previously described, several user oriented conversational mode systems have been implemented a serious obstacle to a closer interaction between man and machine is that of language. Real time co-operation necessitates the use of a different principle of communication and control from that employed in such as FORTRAN or ALGOL. Computers must be instructed with complete sequences of steps to reach a particular objective while humans tend merely to head in the general direction of the goal.

Licklider (1965) recognises two forms of on-line program for man-machine co-operation:

(i) problem solving, hill climbing self organising programs.

(ii) The real time concatenation of pre-programmed segments and closed subroutines which the human operator can call into action simply by name.

STATPAC, the Lincoln Reckoner and STAT-CHAT are three

systems based on the latter principle while The General Problem Solver of Newell, Shaw and Simon (Minsky 1966) falls into the former grouping.

The conversational programs which approach a state of close partnership between man and machine are those utilising a natural communication language. It is with such "intelligent" systems that much of the future of conversational programming will lie.

Though such programs as ELIZA and STUDENT have a limited vocabulary and range of applications they are successful problem solvers and their development has paved the way for more comprehensive systems.

Several interactive systems, such as DOCTOR (a development of ELIZA), have been directed at teaching (see Example 10) and in this field of application have proved to be of much use.

Example 10: from Weizenbaum (1967) . Conversation based on a script written by Dr Edwin Taylor - Education Research Center of M.I.T.

PLEASE CONSIDER HOW WE MIGHT SYNCHRONIZE TWO CLOCKS TO

READ THE "SAME TIME". ONE OF THE CLOCKS IS IN SAN FRANCISCO THE OTHER IS IN NEW YORK. AS YOU THINK ABOUT THE PROBLEM, ASK ME ANY QUESTIONS THAT OCCUR TO YOU. WHEN YOU ARE READY TO PROPOSE A SYNCHRONISATION METHOD TYPE "READY". THEN I WILL QUIZ YOU ABOUT YOUR PROPOSED METHOD.

WHAT DO YOU MEAN BY SYNCHRONIZE.

FOR THE PRESENT MAKE USE OF THE EVERYDAY MEANINGS OF SUCH PHRASES AS "SAME TIME" AND "SYNCHRONIZE" LATER WE WILL DISCUSS THE MEANINGS OF THESE TERMS MORE CAREFULLY.

MAY I USE TELEGRAPH SIGNALS.

CERTAINLY, FEEL FREE TO USE TELEGRAPH.

READY.

NOW I WILL QUIZ YOU ON YOUR SYNCHRONIZATION METHOD. DOES YOUR METHOD OF SYNCHRONIZATION INVOLVE THE PROPAGATION OF SOME SIGNAL - OR MATERIAL OBJECT - FROM ONE PLACE TO ANOTHER.

YES

WHAT KIND OF PROPAGATING SIGNAL OR TRAVELLING OBJECT DO YOU PROPOSE TO USE.

A WATCH.

WILL YOUR WATCH MOVE AT A SPEED VERY MUCH SMALLER THAN THE SPEED OF LIGHT.

OF COURSE.

GOOD. TRAVELLING CLOCKS CAN BE USED TO SYNCHRONISE DISTANT CLOCKS TO ARBITRARILY HIGH ACCURACY IF THEY TRAVEL SLOWLY ENOUGH. YOU MAY LEAVE THE PROBLEM BY PUSHING THE "QUIT" BUTTON TWICE. IF YOU WOULD LIKE TO EXPLORE FURTHER THE MATTERS OF PRINCIPLE ASSOCIATED WITH OTHER METHODS, CHOOSE ANOTHER WAY TO COMMUNICATE BETWEEN THE DISTANT CLOCKS. WHAT KIND OF PROPAGATING SIGNAL OR TRAVELLING OBJECT DO YOU PROPOSE TO USE.

LIGHT.

WELL, OF COURSE THE LIGHT SIGNAL WILL NOT FOLLOW THE CURVATURE OF THE EARTH. SO LET US ASSUME THAT SOME SYSTEM OR RELAY TOWERS IS USED. NOW LET ME ASK, WILL THE LIGHT SIGNAL TRAVEL DIRECTLY BETWEEN NEW YORK AND SAN FRANCISCO OR

WILL THE LIGHT SIGNAL TRAVEL TO OR FROM SOME INTERMEDIATE POINT.

IT WILL TRAVEL FROM ST. LOUIS IN BOTH DIRECTIONS.

IS YOUR INTERMEDIATE POINT EQUIDISTANT FROM NEW YORK AND SAN FRANCISCO.

LET'S SAY IT IS, YES.

MY UNDERSTANDING IS THAT YOU EXPECT TO USE TWO SIGNALS AND THAT THESE SIGNALS WILL EITHER CONVERGE ON OR BE SENT OUT FROM A POINT EQUIDISTANT FROM NEW YORK AND SAN FRANCISCO. IS MY UNDERSTANDING ON THIS POINT CORRECT.

YES

WILL THE PAIR OF SIGNALS CONVERGE ON THE INTERMEDIATE POINT OR BE SENT OUT FROM THIS PEINT.

THEY WILL BE SENT OUT FROM THIS POINT.

WILL THE TWO SIGNALS LEAVE THE INTERMEDIATE POINT AT THE SAME TIME.

CERTAINLY.

AHA, I'VE GOT YOU. YOU ARE TRYING TO SYNCHRONIZE THE NEW YORK AND SAN FRANCISCO CLOCKS TO READ THE "SAME TIME". HOW CAN IT BE VALID TO ASSUME A "SAME TIME" FOR THE DEPARTURE OF A PAIR OF SIGNALS FROM THE MIDPOINT.

BECAUSE THE PAIR OF SIGNALS LEAVE FROM THE SAME POINT.

RIGHT.
THE PHRASE "SAME TIME" HAS AN UNAMBIGUOUS MEANING WHEN APPLIED TO EVENTS THAT OCCUR NEAR THE SAME POINT IN SPACE. NOW YOU ARE GOING TO CONTINUE YOUR ARGUEMENT BY SAYING THAT THE TWO SIGNALS THAT START FROM THE MIDPOINT AT THE SAME TIME WILL PRESUMABLY REACH NEW YORK AND SAN FRANCISCO AT THE SAME TIME. THESE SIGNALS CAN THUS BE USED TO SYNCHRONIZE THE NEW YORK AND SAN FRANCISCO CLOCKS. IS THIS THE WAY YOUR ARGUEMENT WILL CONTINUE.

YES.

ONE FINAL QUESTION. WHAT ASSUMPTION ARE YOU MAKING ABOUT THE SPEEDS OF THE TWO SIGNALS THAT TRAVEL IN OPPOSITE DIRECTIONS.

THAT THE SPEEDS ARE THE SAME.

VERY GOOD. YOU HAVE DEVISED A VALID METHOD FOR SYNCHRONIZING TWO CLOCKS, ONE IN NEW YORK AND ONE IN SAN FRANCISCO.

The Coordinated Science Laboratory at the University of Illinois has developed an automatic teaching system called PLATO (Bitzer and Braunfeld 1965). The PLATO system controls the simultaneous teaching of a number of students while still allowing each to proceed through the lesson material independently and has served to demonstrate the flexibility which a high speed computer endows in an automatic teaching system. Using a computer based system permits versatility of teaching logics and methods, since this merely requires a change of control program and, in addition, the ready access to the decision making capacities of a large computer permits complicated decisions to be made for each student.

Though Computer Aided Instruction (C.A.I.) is an expanding technique in education and results have, on the whole been encouraging it must be remembered however that many problems arise in the programming of a conversational teaching machine concerned not only with explaining fully

a problem to the student but also with the interpretation and assessment of his answers. Generally teaching programs give long typeouts of instructions for the student and involve the use of large language dictionaries enabling the "instructor" to cope with a large number of differently worded replies.

Conversational usage of computers is a major new phenomenon in its own right and not simply another computer application. Conversational systems, no matter their purpose, contrast with automatic systems where man feeds information into a machine, as the computer is made to search, compute and display information until the user is satisfied. Techniques for on-line problem solving using interactive systems have yet a long way to develop before realising a state of man-machine symbiosis between user and computer as envisaged by Licklider (1960) but, in the writer's opinion, it is through user oriented systems, utilising the automatic application of pre-written routines that such a partnership can be approached. This philosophy has been behing the design of STAT-CHAT as, to the problem solver, "inside information is of no value" (Shaw 1968) the solution of the problem in hand being the main objective in mind.

The applications of conversational computation are largely unrestricted, techniques for on-line problem solving generally being suitable for "whatever you do in the course of a normal day". (Orr 1968)

ACKNOWLEDGEMENTS

My grateful thanks are due to Dr A. J. Cole who supervised my work throughout.

This study was undertaken during the tenure of a Scientific Research Council postgraduate course studentship.

APPENDIX 1

STAT-CHAT USER'S MANUAL

"STAT CHAT"

A CONVERSATIONAL PROGRAM FOR ONLINE DATA ANALYSIS

R.M. Campbell

Computing Laboratory

University of St. Andrews

May 1968

INDEX

INTRODUCTION

"Stat chat" is approgrammed technique designed to ease the use of a data processing system by a person who is primarily concerned with interpreting the significance of a data set and who is often unable to spend the time necessary to master a programming language.

Direct communication with the computer is achieved by a simple conversational mode procedure via an on-line typewriter, information being supplied by the user in response to specific questions put by the computer.

The user is required only to be conversant with the few rules of grammar and simple vocabulary of operations as set out in the following pages.

PRELIMINARY REQUIREMENTS

Preparation of data

"Stat chat" can deal with up to 9 vectors of numbers all of which must be of equal length and the sum lengthsof which must not exceed 1000. Thus if there are M vectors each of length N then M x N must not exceed 1000.

In punching the data cards the equivalent elements of each vector must be recorded on the same card. Each element is allocated 8 card columns and if no decimal point is punched in the field there is assumed to be 3 places of decimals, however the punching of the point overrides this (see appendix).

Twocards must precede the data set:

(i) title card : a user reference which can be punched in any or all of the 80 columns and which is required for output identification.

(ii) header card: with the values of M, the number of data vectors, and N, the number of elements in each vector. These values must be recorded as decimal numbers in the card fields 1-10 and 11-20 respectively.

Provision is made for an alternative to 8 column fields for vector elements. This is possible only if all numbers are recorded with a decimal point and the individual vector elements are separated either by a sign or by, at the least, one blank column. To make use of this provision a non-zero digit must be punched in column 30 of the header card.

Preparation of program deck

To make use of "$$at chat" the following cards arerequired:

card (i) columns 1-6: ++ JOB

card (ii) columns 1-12: ++ XEQSMCCHAT
columns 28-30: 304

card (iii) starting in column 1: *LOCALMCCHAT, MCINPT, MCMEAN,
MCSTDV, MCCORL, MCARGE, MCDTCK,
MCPOLY, MCRECP, MCRNGE,

card (iv) starting in column 1: *LOCALMCADON, MCSOFF, MCMINT,
MCDINT, MCSEXP, MCSLOG, MCTAVG,
MCTVAR, MCGENS, MCPRNT,

card (v) starting in column 1: *LOCALMCTOTS, MCSUM1, MCNATL,
MCANTI, MCROOT, MCSINE, MCCOSS,
MCTANS, MCLNAT, MCAUNT,

card (VI) starting in column 1:*LOCALMCPROG, MCMULT, MCDIVE,
MCADD2, MCSUB2, MCLOGS, MCXPOT,
MCCHIS, MCSTUD, MCREGS

card (vii) title card

card (viii) header card

data cards

card (ix) (the last card) columns 1-4: ≠≠≠≠

Note: sets of cards (i)-(vi) and (ix) are available at the computing laboratory. It is only necessary for the user to prepare cards (vii) and (viii) and his data and to insert these before the last card.

≠ is a record mark and is represented by a multipunch code of 0,2,8 in the same card column.

Provision is made for a third method of data input.

To make use of this option a non zero digit must be recorded in column 35 of the header card and a "format card" must follow the header card in the program deck. The "format card" must be punched in the following way:

starting in column 1: 30074 FORMAT (--------)
where the data format is recorded within the parentheses.

The program deck must also be altered in the following way:

card (ii) columns 28-30: 305

card (vi) starting in column 1: *LOCALMCPROG, MCMULT,
MCDIVE, MCADD2, MCSUB2, MCLOGS,
MCXPOT, MCCHIS, MCSTUD, MCREGS,

a new card must be added to the deck following card (vi) punched as follows:

starting in column 1: *LOCALCLFMAT

This method of data input is intended for the more experienced computer usár.

Note: if more than 5 figures appear before the decimal point then errors will occur in the print out of the results of certain statistics.

VOCABULARY OF OPERATIONS

The following are the names of the data processing procedures which are available to users of "stat chat".

Results are printed on the line printer unless otherwise stated.

Data check: prints out the first and last data cards.

Data print: gives a print out of each data vector.

Range: prints the largest and smallest values of each data vector.

Means: prints out the mean value of each data vector

$$\bar{x} = \sum X_i / N$$

Standard deviation: prints out the mean value and standard deviation of each data vector

$$SD = \sqrt{\frac{\sum X_i^2 - (\sum X_i)(\sum X_i) / N}{N - 1}}$$

Correlation: prints out a matrix of product moment correlation coefficients for all possible pairs of data vectors

$$r^2 = (\sum xy)^2 / \sum x^2 . \sum y^2$$

T test: prints out a matrix of values for students t for all possible pairs of data vectors using Bessel's correction to obtain a best estimate of the population standard deviation of the discrepencies

$$t = \frac{\left|\bar{X} - \bar{x}\right| \sqrt{n}}{o} \qquad \text{where} \quad o = s \sqrt{\frac{n}{n-1}}$$

Arrange: prints any chosen data vector in descending numerical order with the corresponding elements of any other chosen data vector but leaves the original vectors unchanged.

Chi squares: prints the value of chi square for any chosen pair of data vectors

Natlog: will transform the data matrix by taking the $\log_e$ of each element.

Exponential: will transform the data matrix by taking the exponential value of each element.

Log: will transform the data matrix by taking the $\log_{10}$ of each element.

Antilog: will transform the data matrix by taking the antilog of each element.

Scatter diagram: plots on the digital plotter a scatter diagram between any chosen pair of data vectors. As no other statistics can be requested after scatter diagrams have been drawn users are advised to keep the drawing of such diagrams until all other statistics have been chosen and executed. Up to 10 scatter diagrams can be drawn in one call (see section "talking withethe computer" Q(x)).

Variance: this statistic will test whether the variances o_1^2 and o_2^2 are equal providing both come from a normal population. Results are determined operating under a 95% confidence interval for chosen data vectors.

Test Means: this will test whether the means u_1 amd u_2 of chosen data vectors are equal providing both come from a normal population. This test will first assume $o_1^2 \neq o_2^2$ ie. unequal variances and then $o_1^2 = o_2^2$. Results are determined operating with a user selected confidence interval of 0.90, 0.95 or 0.99.

General: will characterise a particular data vector by performing elementary statistical calculations (point estimates) determinning the 0.95 and 0.99 confidence intervals for the sample mean, assuming the data is normally distributed and generating a histogram of up to 21 intervals.

Totals: this will print the sums of the elements in each of the data vectors.

Sum: is as above but is calculated only for one chosen data vector. The result is given on the typewriter.

Read new data: allows a new data set to be read in replacing that already in store.

Regression: offers a choice between linear and polynomial.

(i) linear: gives the linear regression equation $y = mx + c$ for any chosen pair of data vectors.

$$m = \frac{\sum xy - \frac{(\sum x)(\sum y)}{N}}{\sum x^2 - \frac{(\sum x)^2}{N}}$$

$$c = \frac{(\sum x)(\sum xy) - (\sum y)(\sum x^2)}{(\sum x)^2 - N(\sum x^2)}$$

(ii) polynomial: gives an approximating polynomial up to a chosen degree not greater than 15 by the method of least squares printing the back solutions for the final iteration.

Two "statistics" are provided to enable the user to transform data vectors.

(1) Combine: allows the following procedures.

(i) Add: transforms any chosen data vector by adding to its elements the corresponding elements of any other chosen data vector.

(ii) Subtract: transforms any chosen data vector by subtracting from its elements the corresponding elements of any other chosen data vector.

(iii) Multiply: transforms any chosen data vector by multiplying its elements by the corresponding elements of any other chosen data vector.

(iv) Divide: transforms any chosen data vector by dividing its elements by the corresponding elements of any other chosen data vector.

(2) Change: allows the following procedures.

(i) Add: transforms any chosen data vector by adding to each of its elements a specified integer scalar.

(ii) Subtract: transforms any chosen data vector by subtracting from each of its elements a specified integer scalar.

(iii) Multiply: transforms any chosen data vector by multiplying each of its elements by a specified integer scalar.

(iv) Divide: Transforms any chosen data vector by dividing each of its elements by a ppecified integer scalar.

(v) Reciprocal: transforms any chosen data vector by taking the reciprocal value of each of its elements.

(vi) Root: transforms any chosen data vector by taking the square root of each of its elements.

(vii) Sine: Transforms any chosen data vector by taking the sine of each of its elements.

(viii) Cos: transforms any chosen data vector by taking the cosine of each of its elements.

(ix) Tan: transforms any chosen data vector by taking the tangent of each of its elements.

(x) Log: transforms any chosen data vector by taking the $\log_{10}$ of each of its elements.

(xi) Antilog: transforms any chosen data vector by taking the antilog_{10} of each of its elements.

(xii) Natlog: transforms any chosen data vector by taking the $\log_e$ of each of its elements.

(xiii) Exponential: transforms any chosen data vector by taking the exponential value of each of its elements.

Note: the permitted range for scalars is 1-99999.

TALKING WITH THE COMPUTER

Communication between user and machine is via the online console typewriter which requires all user replies to be terminated with the R/S key (release and start) located at the top left of the keyboard.

Typing slips can be corrected by the CORR key on the top row of the keyboard which erases one letter per depression. A count is kept of uncorrected errors and invalid replies, only 9 being allowed before the "stat chat" system is abandoned.

A request by the user for help can be made at any reply point (except the very first) by typing the word 'help'. This will result in either a list of possible replies or a reference to a particular section of this manual.

The following are the possible questions which the computer may pose, with the appropriate answers.

Q(i) Have you read the manual? If not type 'no' and press the R/S key.

A(i) The only valid reply is 'yes'. After this has been given the data block is read in.

Q(ii) Do you wish a list of new statistics?

A(ii) Answer 'yes' or 'no'. On answering 'yes' a list of procedures added to the system after this version of the manual was written is given.

Q(iii) Do you wish to name the variables?

A(iii) Answering 'yes' allows the user to assign names to each of the data vectors. If this is not required answer 'no'.

Q(iv) Variable 1=

A(iv) If 'yes' was the response to question (iii) the user must give names of not more than 8 characters for each variable. These names appear as headings to the results. This question is repeated until all the data vectors have been named.

Q(v) Which statistic do you require?

A(v) Answer with one of the names from the vocabulary of operations.

Q(vi) Did you want XXXXX ?

A(vi) If an operation was mis-spelt the above question is asked with a calculated guess at the statistic probably intended. Answer 'yes' or 'no'.

Q(vii) Linear or polynomial?

A(vii) If regression was asked for the choice must be made of linear or polynomial. Answer either with linear if the linear regression equation is required or with poly or polynomial if the polynomial regression equation is required.

Q(viii) To what degree?

A(viii) If polynomial regression has been requested the degree to which the polynomial is to be taken to must be given. Answer with a number not greater than 15.

Q(ix) For which variables?

A(ix) If one of the operations which requires chosen variables has been selected then these must be given here. There are various forms of answer as follows:

1,2. data vectors one and two are passed to the operator as the x and y values respectively.

1,2,3. data vectors one and two and one and three are passed to the operator as x and y values respectively. The operation is executed twice.

1,2-4. data vectors one and two, one and three, one and four are passed to the operator as the x and y values. The operation is executed three times.

The above forms of syntax can be combined by the use of a + eg. 1,2+2,3,4+1,3-5.

A list of up to 30 characters can be given in this way.

After the execution of an operation the user is asked to select another statistic and the process continues until 'none' is the answer to this question, upon which "stat chat" comes to an end.

Note: If an operation requiring selected data vectors has just been executed and the same operation is required again then the answer 'same' in reply to 'which statistic do you require?' is valid. Similarly if the same set of vector parameters, as in a previous statistic, are required the reply 'same' is valid for the question 'for which variables?'.

When a new data set is read in by the execution of the 'read new data' operation the user has the opportunity to rename his variables before any operations are executed. Similarly when the data vectors are altered in any way eg. by 'change' or 'combine' the user is given the opportunity to rename the variables.

With the change and combine statistics the user is invited to select one of the routines in these procedures. Variables can be delimited for the combine routine as for any other statistic requiring chosen vectors (see Q(x)) but the change routine will deal with only one vector at a time. After execution of a change or combine routine the user is invited to request

another and this continues until this reply is 'none'.

PUNCHING A PROGRAM DECK FOR FUTURE USE

Provision is made for the user who wishes to execute at some future date the same operations as he requests in his 'conversation' with the computer. Only 20 operations (ie. subprogram executions) will be executed by this method and a separate program is provided for each data set. As there is a limit to the number of operations which can be executed in this way an indication is given to the conversational user when this limit is reached. To make use of this provision a non-zero digit must be punched in column 25 of the header card which precedes the data.

Execution of this program deck requires a similar data set as did the conversational program with the addition of a 'names' card which precedes the title card. This names card must contain the names of the data vectors punched in 8 column fields. The data set is placed before the last card of the punched program.

Scatter diagrams are not drawn by this program.

CONVENTIONAL USE

For the user who does not wish to use the conversational method of processing his data provision is made for a more conventional method.

To make sure of this provision the user must specify in a number of parameter cards which precede his data codes for the operations he requires and also any additional parameters which these operations require.

These cards are as follows:

2 Means
3 Standard deviation
4 Correlation
5 Scatter diagram
6 Data check
7 Arrange
8 Range
9 Data print
10 Chi squares
11 T test
12 Linear regression
13 Polynomial regression

14 Read new data

19 Logs (base 10)

20 Exponential

27 General

28 Variance

29 Test means

30 Total

31 Sum

32 Logs (base e)

33 Antilogs (base 10)

Combine routines:

15 Multiply

16 Divide

17 Add

18 Subtract

Change routines:

21 Multiply

22 Divide

23 Add

24 Subtract

25 Logs (base 10)

26 Exponential

34 Root

35 Sine

36 Cosine

37 Tan

38 Logs (natural)

39 Antilogs (base 10)

40 Reciprocal

The code numbers of new routines included after the manual was prepared are given with the names on requesting a list of new statistics in the conversational node program.

Data preparation is as for execution of the punched program deck requiring the names of the data vectors to be given on the 'names' card.

Program deck preparation is as follows:

card (i) columns 1-6: ++ JOB

card (ii) columns 1-12: ++ XEQSMCCALL
columns 28-30: 304

card (iii) starting in column 1: *LOCALMCCALL, MCINPT, MCMEAN,
MCSTDV, MCCORL, MCARGE, MCDTCK,
MCPOLY, MCRECP, MCRNGE,

card (iv) starting in column 1: *LOCALMCADON, MCSOFF, MCMINT,
MCDINT, MCSEXP, MCSLOG, MCTAVG,
MCTVAR, MCGENS, MCPRNT,

card (v) starting in column 1: *LOCALMCTOTS, MCSUM1, MCNATL,
MCANTI, MCROOT, MCSINE, MCCOSS,
MCTANS, MCLNAT, MCAUNT,

card (vi) starting in column 1: *LOCALMCMULT, MCDIVE, MCADD2,
MCSUB2, MCLOGS, MCXPOT, MCCHIS,
MCSTUD, MCREGS

card (vii) columns 1-10: the number of operations required. This must be no greater than 40 andmmust be punched with a decimal point.

card (viii) parameter card with the code numbers of the operations which are required. These must be recorded in two column fields and right justified.

card (ix) parameter card with the numbers of the data vectors required for those operations requiring chosen data vectors. If no such operations are requested this card is left blank. The numbers are recorded in one column fields.

Note: if the "format card" option for data input is chosen then this card must be punched in the following way:

starting in column 1: 26584 FORMAT (--------)

where the data format is recorded within the parentheses.

card (vi) of the program deck must be changedtto read:

starting in column 1: *LOCALMCMULT, MCDIVE, MCADD2,
MCSUB2, MCLOGS, MCXPOT, MCCHIS,
MCSWUD, MCREGS, CLFMAT

APPENDIX

Data set preparation

Table of data to be coded for 80 column computer cards

	variables			
observations	10.0	1.0	66.1	0.4
	1110	6.8	114.3	-0.1
	12.0	13.4	201.9	0.09
	13.0	20.1	516.7	-1.0
	14.0	30.3	601.1	-0.1

SAMPLE DATA PREPARATION

				Title card
4.0	5.0			Header card
10.0	1.0	66.1	0.4	1st data card
11.0	6.8	114.3	-0.11	2nd data card
12.0	13.4	201.9	+0.09	3rd data card
13.0	20.1	516.7	-1.0	4th data card
14.0	30.3	601.1	-0.1	5th data card

Card columns:

1 10 20 30 40.....80

Example of parameter card preparation for conventional use

The routines required are Means, Chi squares and Test Means (at 0.90 signifecance level). Chi sqaares for vectors 1 and 2 and Test Means for 3 and 4.

card (vii) starting in column 1: 3.0

card (viii) starting in column 1: 021029

card (ix) starting in column 1: 1234

card (x) starting in column 1: 00001

card (xi) names card

card (xii) title card

card (xiii) header card

data cards

card (xiV) (last card) columns 1-4 = =

Transcript of conversational mode dialogue:

```
HAVE YOU READ THE MANUAL. IF NOT TYPE NO AND PRESS R/S KEY
     YES
DO YOU WISH LIST OF NEW STATISTICS
     NO
DO YOU WISH TO NAME THE VARIABLES
     YES
VARIABLE 1= AAAAAAAA
VARIABLE 2= BBBBBBBB
VARIABLE 3= CCCCCCCC
VARIABLE 4= DDDDDDDD
VARIABLE 5= EEEEEEEE
VARIABLE 6= FFFFFFFF
WHICH STATISTIC DO YOU REQUIRE
     DATA CHECK
WHICH STATISTIC DO YOU REQUIRE
     CHANG
DID YOU WANT CHANGE
     YES
WHICH CHANGE ROUTINE DO YOU REQUIRE
     SUBTRACT
FOR WHICH VARIABLE
1    1
GIVE VALUE OF INTEGER
     1
```

card (x) parameter card containing values for any other parameters eg. maximum degree of a polynomial regression, value of an intiger scalar. Note: these numbers must be punched in 5 column fields and right justified and only 16 can be recorded therefore no more than 16 operations requiring data from these cards can be called. If no such operations are required this card is left blank. Note: for the Test Means statistic which requires a confidence level to be given this can be indicated on this card by recording the values 1,2 and 3 to indicate 0.90, 0.95 and 0.99.

card (xi) names card

card (xii) title card

card (xiii) header card

data cards

card (xiv) (last card) column 1-4: ++++

As with the conversational mode program a program deck to execute the same routines as requested by the conventional use can be obtained by punching a non-zero digit in column 25 of the header card.

```
WHICH CHANGE ROUTINE DO YOU REQUIRE
     NONE
DO YOU WISH TO RENAME THE VARIABLES
     NO
WHICH STATISTIC DO YOU REQUIRE
     DATA CHECK
WHICH STATISTIC DO YOU REQUIRE
     SUM
FOR WHICH VARIABLE
     L
INVALID TRY AGAIN
FOR WHICH VARIABLE
     6
SUM OF ELEMENTS OF FFFFFFFF = .16000000E+02
WHICH STATISTIC DO YOU REQUIRE
     CHANGE
WHICH CHANGE ROUTINE DO YOU REQUIRE
     MULTIPLY
FOR WHICH VARIABLE
     1
GIVE VALUE OF INTEGER
     0
INVALID TRY AGAIN
     1
```

```
WHICH CHANGE ROUTINE DO YOU REQUIRE
     NONE
DO YOU WISH TO RENAME THE VARIABLES
     NO
WHICH STATISTIC DO YOU REQUIRE
     DATA CHECK
WHICH STATISTIC DO YOU REQUIRE
     SCATTER DIAGRAM
NO MORE STATISTICS CAN BE REQUESTED AFTER SCATTER DIAGRAMS
HAVE BEEN DRAWN DO YOU STILL WANT SCATTER DIAGRAMS DRAWN NOW
     YES
FOR WHICH VARIABLES
     1,2.

END OF JOB
```

Note: In the above example, user replies have been inset.

APPENDIX 2

GLOSSARY

Address. A label, name or number identifying a register, location or unit where information is stored in the computer.

Binary digit. Either of the digits 0 or 1 which may be used to represent the binary conditions ON and OFF.

Buffer. A device that temporarily stores information during a transfer of information.

Compiler. A computer program that has the ability to translate a machine independent source program into machine language instructions.

Debugging. The process of locating and correcting errors in programs.

Dictionary. A list of code names or keyword accompanied by an indication of their meaning.

Executive. (=Monitor) A routine which controls the execution of other routines.

Format. The arrangement of information for input to a computer or the arrangement desired for output of information.

Hash total. The summation for checking purposes of fields which would not normally be summed.

Indexing. The act of modifying an instruction by the contents of an index register in order to obtain a new effective address.

Interpretive routine. (=Interpreter) A routine that decodes instructions written as pseudocodes and immediately executes those instructions as contrasted with a compiler that decodes the pseudocodes and ppoduces a machine language routine to be executed at a later time.

Interrupt. A break in the normal flow of a system or routine such that the flow can be resumed from that point at a later time. The iterrupt is usually from an external source.

Library. A group of standard, proven, routines which may be incorporated into larger routines.

Light pen. A pointing device which when pointed at an emitting light source on a cathode ray tube provides positional information to the data processor.

MAC. An acronym that has been variously translated as Multi-Access Computer, Machine Aided Cognition, and Man And Computer.

Machine Language. Information recorded in a form which may be used by a computer without prior translation.

MIT. An acronym for Massachusetts Institute of Technology.

Multi-programming. A technique for handling numerous programs simultaneously by overlapping or interlacing their execution and by permitting more than one program to time share machine components.

Off-line. Pertains to the operation of input/output devices or auxillary equipment not under direct control of the central processing unit.

On-line. As above but under the direct control of the central processing unit.

Operating system. A group of programming systems operating under the control of a Monitor.

Programming system. Any method of programming problems other than in machine language.

Semantics. The meanings of words.

Subprogram. A part of a large program which can be compiled independently.

Subroutine. A program which defines desired operations and which may be included in another program to cause these desired operations.

Symbiosis. The co-operation of two organisms for their mutual benefit and existence.

Syntax. The arrangement and mutual relationships of words.

Time-sharing. A technique whereby a computer serves a large number of people at once.

REFERENCES

BITZER, D.L., and BRAUNFELD, P.G. (1965). PLATO: A Computer-Controlled Teaching System. Computer Augmentation of Human Reasoning, 89-104, eds. M.A. Sass and W.D. Wilkinson. Washington: Spartan Books, Inc..

BOLT, R.H. (1968). Computer-Assisted Socratic Instruction. Conversation Computers, 90-95, ed. W.D. Orr. New York: John Wiley and Sons, Inc..

BURSTALL, R.M., and POPPLESTONE, R.J. (1968). POP-2 Papers. Edinburgh: Oliver and Boyd.

CLARK, J.H. (1968). The Simulation of the Human Hypnotist by a Teaching Machine. I.F.I.P. Congress 68, Applications 3, H80-88.

COLIN, A.J.T. (1967). On-line Access Systems in Statistics. Journal Royal Statistical Society (Series C), 16, 3, 111-119.

COLLINS, J.S., AMBLER, P., BURSTALL, R.M., DUNN, R.D., MICHIE, D., POPPLESTONE, R.J., and PULLIN, D.J.S. (1968). MULTI-POP/67: A Cheap On-Line System for Numerical

and Non-Numerical Computing. Mini-MAC Reports: No. 5, Department of Machine Intelligence and Perception, University of Edinburgh.

COOPER, B.E., and WHITESIDE, C.M. (1963). The presentation of Experimental Data to Computer. Harwell: United Kingdom Atomic Energy Authority.

CORBIN, H.S., and FRANK, W.L. (1966). Display Oriented Computer Usage System. Proc. 21st ACM National Conference, 515-520.

CULLER, G.J., and FRIED, B.D. (1965). The TRW Two-Station On-Line Scientific Computer. Computer Augmentation of Human Reasoning, 65-88, eds. M.A. Sass and W.D. Wilkinson. Washington: Spartan Books, Inc..

GOODENOUGH, J.B. (1965). A Light Pen Controlled Program for On-Line Data Analysis. Comm. ACM., 8, 130-134.

INTERNATIONAL COMPUTERS AND TABULATORS LTD. (1967), JEAN-A 1900 System based on JOSS. Bracknell: I.C.T.

JACKS, E.L. (1968). A Laboratory for the Study of Graphical Man-Machine Communication. Conversational Computers,

143-146, ed. W.D. Orr. New York: John Wiley and Sons, Inc..

KAPLOW, R., BRACKETT, J., and STRONG, S. (1966). Man-Machine Communication in On-Line Mathamatical Analysis. Proc. Fall Joint Computer Conference, 456-477.

KEMENY, J.G., and KURTZ, T.E. (1967). BASIC programming. New York: John Wiley and Sons, Inc..

KLERER, M., and MAY, J. (1964). An Experiment in a User Oriented Computer System. Comm. ACM., 7, 4, 290-294.

LICKLIDER, J.C.R. (1960). Man-Computer Symbiosis. I.R.E. Trans. H.F.E., 4-11.

LICKLIDER, J.C.R. (1965). Man-Computer Partnership. Int. Sci. Tech., 18-26.

MEADOW, C.T., and WAUGH, D.W. (1966). Computer Assisted Interrogation. Proc. Fall Joint Computer Conference, 381-394.

MINSKY, M. (1966). Artificial Intelligence. Information, 193-210, San Francisco: W.H. Freeman and Company.

ORR, W.D. (1968). Conversational Computers. New York: John Wiley and Sons, Inc..

ORR, W.D., and THE STAFF OF BOLT BERANEK AND NEWMAN, INC. (1968). The Culler-Fried Approach to On-Line Computing. Conversational Computers, 23-28, ed. W.D. Orr. New York: John Wiley and Sons, Inc..

OSTLE, B. (1968). Statistics in Research. Iowa Univ. Press.

PFEIFFER, J.E. (1968). The Robot Draftsman. Conversational Computers, 137-139, ed. W.D. Orr. New York: John Wiley and Sons, Inc..

PIRTLE, M.W. (1968). HELP. Conversational Computers. 96-102, ed. W.D. Orr. New York: John Wiley and Sons, Inc..

POPPLESTONE, R.J. (1968). POP-1: An On-Line Language. Machine Intelligence 2, 185-194, eds. E. Dale and D. Michie. Edinburgh: Oliver and Boyd.

POPPLESTONE, R.J. (1968a). The Design Philosophy of POP-2. Machine Intelligence 3, ed. D. Michie. Edinburgh: Univ. Press.

PULLIN, D.J.S. (1967). A Plain Man's Guide to MULTI-POP Implementation. Mini-MAC Reports: No. 2, Department of Machine Intelligence and Perception, University of Edinburgh.

RAPHAEL, J. (1966). The Structure of Programming Languages. Comm. ACM., 9, 3, 67-71.

SASS, M.A., and WILKINSON, W.D. (1965). Computer Augmentation of Human Reasoning. Washington: Spartan Books, Inc..

SCHMEDEL, S.R. (1968). Electronic Sketching. Conversational Computers, 135-136, ed. W.D. Orr. New York: John Wiley and Sons, Inc..

SCHWARTZ, J.I. (1965). Programming Languages for On-Line Computing. Proc. I.F.I.P. Congress 65. New York.

SCHWARTZ, J.I. (1966). On-Line Programming. Comm. ACM., 9, 3, 199-203.

SHANNON, S., and HENSCHKE, C. (1967). STAT-PACK: A Biostatistical Programming Package. Comm. ACM., 10, 2, 123-125.

SHAW, J.C. (1964). JOSS: A Designer's View of an Experimental

On-Line Computing System. Proc. Fall Joint Computer Conference, 455-464.

SHAW, J.C. (1968). JOSS: Experience with an Experimental Computing Service for Users at Remote Consoles. Conversational Computers, 15-22, ed. W.D. Orr. New York: John Wiley and Sons, Inc..

STOWE, A.N., WIESEN, R.A., YNTENNA, D.B., and FORGIE, J.W. (1966). The Lincoln Reckoner: An Operation-Oriented, On-Line Facility with Distributed Control. Proc. Fall Joint Computer Conference, 433-444.

TUKEY, J.W., and WILK, M.B. (1966). Data Analysis and Statistics: An Expository Overview. Proc. Fall Joint Computer Conference, 695-709.

WAKS, D.J. (1967). Conversational Programming on a Small Machine. Datamation, 13, 45-49.

WEIZENBAUM, J. (1966). ELIZA-A Computer Program for the Study of Natural Language Communication between Man and Machine. Comm. ACM., 9, 1, 36-45.

WEIZENBAUM, J. (1967). Contextual Understanding by Computers. Comm. ACM., 10, 8, 474-480.

AUTHOR INDEX

SUBJECT INDEX