ANALYSIS OF FORTRAN USE IN A UNIVERSITY ENVIRONMENT

A Thesis Submitted for the Diploma of

Imperial College

by

P. G. REDDY, Ph.D.

Centre for Computing and Automation
Imperial College of Science and Technology
University of London

October 1970

Acknowledgements

This dissertation has been written for presenting as part of the requirement of the Diploma of Membership of the Imperial College. The author wishes to acknowledge the valuable assistance and encouragement that has been provided by Mr. E. B. James and Mr. P. Whitehead.

Acknowledgement is also made of the assistance of Mr. G. Richards and other Computer Unit staff. The author would like to thank the authorities of I.I.T., Delhi, for allowing him to spend a year in studying Computing Science at Imperial College. Thanks are also due to the British Overseas Ministry for their kind assistance.

Lastly, the author wishes to express his deep sense of gratitude to Professor S. Gill for accepting him into his department for the Diploma course.

CONTENTS

Abstract

This work is an investigation of methods of analysing the performance of a university computing system. This involves the development of programs for analysis of system output tapes, with the production of statistics concerning error messages, execution ratios and distribution of program times. There is also presented a detailed analysis of Fortran programs involving statement frequency and types of error.

1. INTRODUCTION

1.1 Objectives

The purpose of this work is to investigate the behaviour of a computing system as it processes a typical batch of programs in a university environment.

This is achieved by an analysis of the system output which provides an historical trace in considerable detail. It consists of program listings, reports during compilation and execution and results from each program in turn. From this raw data it is possible to develop a wide variety of interesting statistics which may indicate both the efficiency with which the system processes user programs and also the level of proficiency of the users. It seems that the information produced by such an analysis should prove of fundamental importance to the designers of new computing systems and should also indicate ways in which the user can be trained to make more effective use of the system.

It is claimed that this analysis should be of great use to teachers of Fortran and in general to compile writers. There is a pressing demand for the introduction of computer education at school level and a recent IFIP Conference has given a clear cut recommendation on such education.

1.2 The Computing System

The system under analysis is the PUFFT system for processing small to medium size Fortran programs at Imperial College. In this section we provide background information on the use of computers in colleges, on the use and development of Fortran as the language for college programs and the structure of the PUFFT system.

1.2.1 Computers in the University

The modern electronic computer has produced a revolution in science, engineering, industry and commerce and has permeated our society. Even though a relatively small percentage of computers have been located on university and college campuses, colleges and universities have played a key role in computer development. In fact, the more advanced system programs that permit all computer users easier and more satisfactory access to computers have been developed by the universities (see Refs. 1 and 2).

Computers are used by an increasing number of students either to do homework or laboratory problems, or to learn about the design and operation of computers themselves. Like other new technicological developments, the computer has been over-promoted, abused and misused. All too often an extensive brute force calculation has been made on a computer when some thoughtful analysis would have reduced the problem to one requiring only a modest computer application and perhaps none at all. Better education in computer use should improve this situation. The broad scale reliance of our increasingly technical society on computer systems, formal languages, and the related problem-solving procedures will eventually mean that everyone should have a basic non-technical understanding of the field, much as everyone is now expected to understand something of history, arithmetic, biology, etc. In fact, this has already been started on an experimental basis at the grade school level. It is already generally recognised that the engineers will make so much use of computers that it has been essential to give them instruction in the use of computers as part of their undergraduate training. As use of computers is becoming widespread, students in more and more areas will require training in their use as part of a normal undergraduate education.

Over the past few years there has been a vigorous development of problem-oriented languages, such as Fortran, ALGOL, COBOL, and PL/1. Their use has made it possible within a one-term course to impart the technical knowledge needed for the transcription of a well-defined problem into a computer program. The advantage in having courses designed to develop proficiency in a programming language in schools and colleges is that such courses can form the basis for further formal education in computer-oriented problem analysis and in more advanced programming logic and language.

Engineers have already accepted the computer as an educational aid, and the students of engineering are using the computer extensively as a calculating instrument throughout their undergraduate career. While the same kind of acceptance is anticipated in other academic areas, an even greater impact on education is now occuring through the use of computers for non-numeric processing. This is simply the use of machines for manipulating entities that are not represented in pure numeric form. Included within this non-numeric field is general information storage and retrieval, algebraic manipulation, and linguistic work of all kinds.

Writing a machine language program is usually far more difficult and time-consuming for the programmer than writing a program in a problem-oriented language, such as Fortran or ALGOL. Basic training in computer programming is therefore best carried out in a high level language.

1.2.2 Use of Fortran, History

There is a growing tendency for programs in all application fields to be written in Fortran (mainly because of the worldwide availability of Fortran compilers). Fortran will probably be the most commonly used programming language in the universities and colleges for at least the next five to ten years, again mainly because of the ready availability of Fortran compilers on all machines.

Out of many dialects existing in Fortran, there are two, Fortran II and Fortran IV, which are most commonly used. Fortran II is now officially obsolete, although thousands of Fortran II programs are still in use. The name Fortran was derived from the words IBM Formula TRANslation System, which described the primary object of the original language. The processor (compiler) for this language was modified in 1958 to accept programs written in an augmented Fortran language which was commonly known as Fortran II. From 1962, compilers for Fortran IV began to appear, which provide important additional facilities. In 1964, the American Standard Association drew up a specification of, essentially, Fortran IV which has been widely accepted as the standard Fortran. All Fortran IV compilers should accept a program written to the ASA specification.

It should be noted, however, that of the processors described in references 12 - 16 , very few appear to have complete compatibility with ASA Fortran (see references 3 , 4 , 17 , and 18). In addition two other Fortran processors may be of importance to users. Both of these processors provide for very fast compilation, but may produce less efficient object coding than the manufacturers compilers. PUFFT is used on IBM 7090/4 machines and its specification is nearly the same as version 13 Fortran IV for the IBM 7094 machine.

It is in many ways remarkable that a language as old as Fortran is still in use at all, and the extended Fortran is also able to stand the competition of newer languages such as ALGOL and PL/1. One reason for its survival seems to be the fact that its very limitations, while not too serious from the user's point of view, make it relatively easy to compile. Fast compiling is particularly important for a language much used in teaching since student's exercises spend most of their time in the compiling phase, and indeed the same consideration is important in any research-based computer centre where the ratio of the programs under development to those in production will always remain high.

Another important feature of Fortran is the care with which it has been extended to cover special applications - rather diverse examples can be found in syntax description 5 , in the list processing 6 , in simulation 7 , and on character manipulation 8 . Suggestions for improving Fortran from the user's point of view, mainly by removing irksome and unnecessary restrictions as well as by creating useful extensions to the language, are given by Healy 10 . Further suggestions based on efficiency considerations as seen by the software writer and the machine manager are given in reference 11 . So it appears that there is still considerable scope for development.

1.2.3 The PUFFT System

The PUFFT System at Imperial College handles the load of small-scale Fortran programs produced by students under instruction or researchers developing subsections of larger programs. Programs of this size form an increasingly important part of the work load as computing enters the curriculum of all students.

The development of the PUFFT System is due to Professor S. Rosen and a small group of workers at Purdue University, where the computing requirements are similar to those at Imperial College (reference 1).

PUFFT handles job-to-job sequencing, translating and running of batches of programs submitted to it in Fortran IV source language, and it was primarily designed as an independent operating system for the IBM 7090/94 computer. The entire system is resident in just over 16 thousand words of immediate access storage during both compilation and execution. During compilation the system communicates only with an input tape, from which it reads the source program, and an output tape, on which it writes the program listing. Everything else is handled in the main core memory.

System Routines: The various system routines that reside in the lower part of the core memory (the storage assignment for this is the locations 0 to 17K where K represents $2^{10} = 1024$) are (a) the executive routine, (b) the input-output package, (c) the compiling routines, (d) the subroutine library and (e) diagnostic message routine. A brief description of the various routines follows:

(a) The Executive Routine: This handles the monitor functions in PUFFT. It reads and interprets control cards and controls the sequence of funtions within a job and between jobs.

(b) I/O Package: The I/O routines that reside in the system area are used by the system and by the object programs that are produced. At compile time they provide a double buffered system for reading BCD card images from the input tape, and for writing lines edited for printing the system output tape. A pair of 22 word buffers is provided for BCD input, and the same pair of buffers is used at run time for reading from the input tape or from any other BCD tapes that the object program may specify. Two 120 word buffers are provided in the system area to provide for a blocking factor of at least five lines per block on output to the system tape. BCD output on other tapes during output would use the same buffers.

(c) The Compiling Routines: PUFFT requires that all statements that have to do with type declaration and storage allocation must appear at the beginning of the subprogram in a prescribed order since it is designed as an absolute load-and-go compiler. These include explicit type statements, DIMENSION statements, COMMON, EQUIVALENCE and DATA statements.

(d) The Subroutine Library: All the standard Fortran library subroutines are found in PUFFT and they are part of the resident system. The subroutine library uses about 2600 words of core memory.

(e) The DIAGNOSTIC Message Routine ERROR: A system like PUFFT, intended for use by students and for debugging, must be able to recognise and provide detailed diagnostic messages for all kinds of errors in the source language program. This routine is called ERROR and is designed such that a large number of different error messages can be printed without using an excessive amount of space for their storage.

An error message is selected by a parameter word which designates upto three phrases from a library that could contain a maximum of 511 phrases. Each phrase is encoded in a single computer word which selects a sequence of upto five English words from a table of 127 different words of upto 12 characters each. A phrase may require an insert provided by the calling sequence, in which case the insert is provided in the arithmetic registers and the position of the insert within the phrase is specified in a field of the parameter word which selects the message. The actual words that appear in messages are stored only once, regardless of the number of messages in which they appear. A phrase appears only once in the phrase list, even if it is used in many different messages. Several hundred different messages, and the coding required to select them and to transmit them to the output routine, all take slightly more than 500 words of core storage. The parameter word also transmits a severity code 0, 1, 2 or 3. Most messages are only warnings. Execution is deleted only for serious errors whose severity code is 3.

Error messages produced during the compilation appear in the listing of the source language program. An error message is printed immediately after the source language statement that was on the last card read before the error was detected.

2. RESULTS AND DISCUSSION

In this chapter we provide the results of an analysis of the operation of the PUFFT system over several typical working days.

The analysis may be divided into three main sections. Firstly, an overall measurement of the size of the programs and success in execution; secondly, an analysis of the reasons for failure of the unsuccessful programs; and finally, a survey of the frequency of use of the various features of the Fortran language as evidenced from the program listings.

There are two sets of results for sections (1) and (3) in our analysis. One is based on statistics from all programs run during one week of five working days, the other refers to jobs run during two different days of two other weeks. Both studies are presented to provide some measure of the consistency of the results.

2.1 Execution Analysis

The distribution of program size is given in Fig. 1. The number of jobs in each storage requirement category is shown as a histogram.

From this we conclude that relatively very small jobs are much more frequent than jobs requiring more than eight thousand words of storage. ABout 60% of the jobs take less than 4K, and about 90% of jobs take less than 8K of storage. This implies that in a college where student instruction represented the main part of the computer use, a very high proportion of the work load could be processed by a computer with 8K words of 36 bits storage, not including the system. In fact, out of 1047 jobs analysed within a week, most jobs (339) take less than 1000 words.

Table I gives details of successful and unsuccessful programs. Out of 1047 jobs examined, 48% were successfully executed although 80% of the jobs went into execution. About 32% of the jobs which went into execution were terminated either because of execution errors, or because of excess output (there is an automatic limit to the number of lines which can be printed out). Only 20% of the jobs were rejected during compilation. This justifies the PUFFT system in containing a richer vocabulary of warning messages than error messages indicating failure.

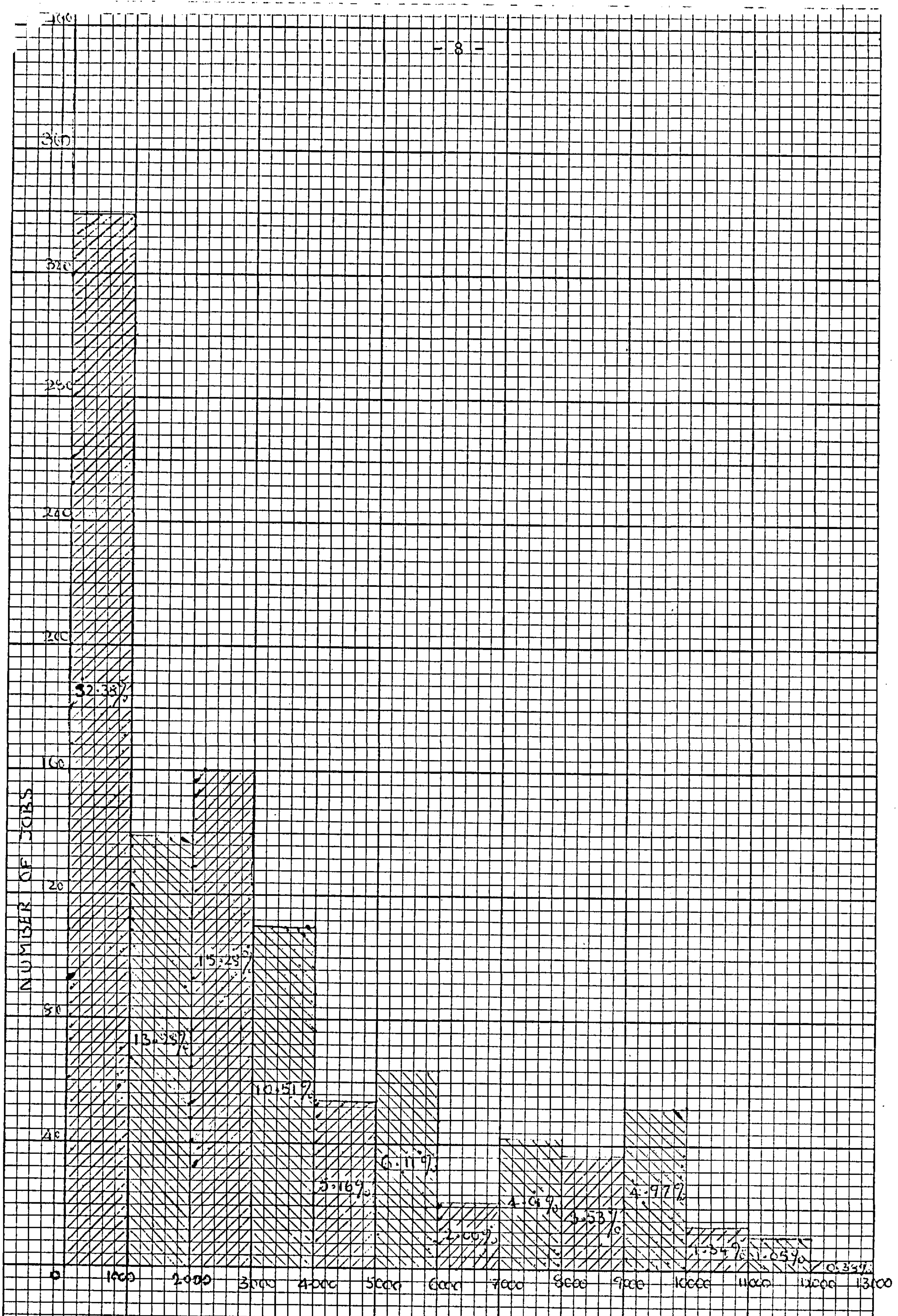


Fig. 1: JOB FREQUENCY VERSUS IAS

About 23% of the jobs were terminated with execution error messages and 9% with excess output. This is calculated from the total of error messages of level 3.

Timing can be summarised as follows:

Average compile time per program	-	3.3 seconds
Average execution time per program	-	49 seconds (approximately)

TABLE I

		Percentage
Total time required for the compilation of the jobs:	57.55 minutes	
Total time taken for the execution of the jobs:	853.258 "	
Number of jobs:	1047	-
Number of jobs without listing:	27	2.57
Number of jobs went into execution:	842	80.42
Number of jobs terminated by stop:	507	48.42
Number of jobs terminated:	670	63.99
Total number of error messages including execution messages:	4132	-
Various types of execution statements:	2019	48.86
Errors of level 0:	2187	52.93
Errors of level 1:	76	1.84
Errors of level 2:	652	15.78
Errors of level 3:	1217	29.45

2.2 Error Analysis

In a university, a large portion of the users of a computing facility are students learning programming and graduate students or research investigators who are not full-time programmers, but are faced with the need of using the computer as an educucational or research tool. For these people the concern of finding and correcting errors in their programs as quickly as possible outweighs the need for efficient object code and sophisticated programming features. For this purpose the PUFFT system gives a satisfactory service. However, the diagnostic messages are not fully documented and it seems that a teacher would need details of the relative frequencies of errors if he is to help his students to correct their mistakes. From a knowledge of the errors he is able to assess the relative importance of statements in Fortran in as much as they give rise to errors and he can concentrate his teaching on eliminating these errors as far as possible.

The error counts in the compiler performance are distributed into the following categories:

(i) Compilation time errors associated with

(a) statement format or sequence
(b) subscripts
(c) arithmetic expressions
(d) FORMAT statements
(e) I/O statements
(f) reference and definition of entities
(g) statement format punctuation
(h) identifiers
(i) DO statements
(j) GOTO and IF statements
(k) storage allocation
(l) system control, exceeding time limits and others.

(ii) Execution time errors associated with

(a) arithmetic faults
(b) computed GOTO operations
(c) I/O operations
(d) reference and definition of entities
(e) library function parameters
(f) subprogram references
(g) limit controls

The results are given in table II. This includes both compilation and execution errors. 5.48% of messages are produced during execution in a total of 1693 reported errors. The remainder occur during compilation.

TABLE II

Distribution of compile time errors	(per cent of total number of compilation errors):
Arithmetic assignment:	1.81
Statement format and sequence:	1.95
Identifiers:	2.50
General Punctuation:	0.63
Reference and definition:	66.84
FORMAT statements:	2.06
DO statements:	0.12
I/O statements:	0.06
System errors:	7.94
Storage allocation:	7.06
Type errors:	8.12
All others (including subscripts):	0.81

Distribution of execution time errors	(per cent of total number of execution errors):
I/O operations:	23.65
Reference and definitions:	25.75
Arithmetic faults:	50.60

From tables III and VII we see that 60 or approximately 17% of the total of 352 jobs contain execution errors and 21% have compilation errors. The number of compilation errors seems to be rather high from this sample survey. They average about 21.5 per program. Most of the compilation errors are reported as "illegal BCD characters" and "inconsistent equivalence of variable". "Illegal BCD character" might arise from several basic mistakes.

but this sample has been completely biased by the existence of one particular program with over 300 mistakes of this type, arising from the use of the wrong punching code on the card punch used.

Regarding "inconsistent equivalence of variable" which has occured the greatest number of times in our survey, it seems likely that it arises mainly from overflow in the various program and data areas which are not directly under the control of the user programmer. Other reasons include illegal common statements or incorrect formatting of an EQUIVALENCE statement or DATA statement. However, the specification of the cause of this particular error message seems outside the purpose of the current work.

Other errors which are seen more frequently in our analysis are:

* "statement type error", due to incorrect specification of the type of variable (REAL or INTEGER)
* array not defined in DIMENSION or COMMON statement
* undefined statement number, subroutine or variable (this error is self-explanatory)
* error in FORMAT at ....
* mixed type operation, - which is caused due to mixing up of REALS and INTEGERS in an expression
* illegal card below, - mainly caused due to the incorrect or extra system card
* check use of array
* number required, statement
* statement cannot be reached

Most of these errors are due to the transposition of cards by careless handling. Surprisingly, there are very few punching errors. (We are not including "illegal BCD character" under this classification as explained already). There are some instances of illegal sequencing of statements, a few cases of missing job cards, and main program defined twice, statement number used twice have also been noticed.

Among the execution errors the commonest are arithmetic faults due to underflow or overflow, illegal subscripts, and encountering the end of the data file unexpectedly. These mistakes do not imply a lack of understanding in the use of program statements, but rather carelessness in the specification of certain numeric values in the program.

TABLE III

ERROR 'STATEMENTS' and their frequency

Level	Statement	Frequency	Percentage
3	ILLEGAL PUNCTUATION - EXECUTION DELETED	9	0.53
3	ILLEGAL REAL CONSTANT - EXECUTION DELETED	4	0.24
3	ILLEGAL SUBSCRIPT AT ...... - EXECUTION TERMINATED	24	1.42
2	ILLEGAL STATEMENT NUMBER - ERROR IGNORED	6	0.35
2	ILLEGAL SEQUENCING OF STATEMENT - STATEMENT IGNORED	22	1.30
3	ILLEGAL USE OF BCD CHARACTER - EXECUTION DELETED	368	21.74
3	ILLEGAL USE OF FUNCTION NAME - EXECUTION DELETED	2	0.12
3	ILLEGAL USE OF PERIOD - EXECUTION DELETED	1	0.06
2	ILLEGAL CARD BELOW IGNORED	105	6.20
3	ILLEGAL USE OF LOGICAL UNIT 00006 - EXECUTION TERMINATED	0	0.00
1	ILLEGAL COMMA IGNORED - WARNING ONLY	1	0.06
3	ILLEGAL ARRAY NAME ...... - EXECUTION DELETED	6	0.35
3	ILLEGAL OPERATOR - EXECUTION DELETED	6	0.35
3	ILLEGAL INTEGER VARIABLE OR CONSTANT - EXECUTION DELETED	6	0.35
3	ILLEGAL END OF STATEMENT - EXECUTION DELETED	1	0.06
3	ILLEGAL VARIABLE NAME - EXECUTION DELETED	2	0.12
3	ILLEGAL NUMBER OF SUBSCRIPT FORMAT - EXECUTION DELETED	1	0.06
3	INCONSISTENT EQUIVALENCE OF VARIABLE - EXECUTION DELETED	655	38.69
3	INDEX AND LIST LENGTH INCONSISTENT - EXECUTION DELETED	1	0.06
2	STATEMENT NUMBER REQUIRED - STATEMENT CANNOT BE REACHED - WARNING ONLY	23	1.36
3	STATEMENT NUMBER ... USED TWICE - EXECUTION DELETED	10	0.59
3	STATEMENT TYPE ERROR - STATEMENT IGNORED - EXECUTION DELETED	120	7.09
3	SYSTEM ERROR AT OCTAL ..... - EXECUTION DELETED	13	0.77
3	PARENTHESIS ERROR - EXECUTION DELETED	4	0.24

TABLE III cont.

Level	Statement	Frequency	Percentage
3	PARITY ERROR ON UNIT 00005 - EXECUTION DELETED	2	0.12
3	PROGRAM OR ARRAY TOO LONG - EXECUTION DELETED	1	0.06
3	PUNCTUATION ERROR - EXECUTION DELETED	1	0.06
0	OVERFLOW AT ...... - WARNING OUTPUT FIVE TIMES ONLY	20	1.29
2	MISSING OR ILLEGAL JOB CARD	7	0.41
3	MIXED TYPE OPERATION - EXECUTION DELETED	23	1.36
3	MAIN PROGRAM MISSING - EXECUTION DELETED	1	0.06
2	MAIN PROGRAM DEFINED TWICE - FIRST MAIN PROGRAM USED	4	0.24
3	ARRAY ...... NOT DEFINED - EXECUTION DELETED	34	2.01
3	ARGUMENT ILLEGAL IN EQUIVALENCE STATEMENT - EXECUTION DELETED	1	0.06
3	AT OCTAL ... , ... USED - UNDEFINED SUBROUTINE OR VARIABLE OR STATEMENT NUMBER - EXECUTION DELETED	34	2.01
3	VARIABLE ..... USED AS FUNCTION - EXECUTION DELETED	6	0.35
3	VARIABLE OR CONSTANT TOO LONG - EXECUTION DELETED	3	0.18
3	HOLLERITH COUNT TOO LONG - EXECUTION DELETED	1	0.06
0	UNDERFLOW AT ..... AC AND MQ - WARNING OUTPUT FIVE TIMES ONLY	27	1.59
2	END CARD MISSING - CALL EXIT USED - WARNING ONLY	3	0.18
2	EOF ERROR - CALL EXIT USED - WARNING ONLY	2	0.12
0	EOF READ ON UNIT 00005 - EXECUTION TERMINATED	19	1.12
3	ERROR IN FORMAT AT ... - EXECUTION DELETED	32	1.89
3	NESTED DO'S OVERLAP - EXECUTION DELETED	2	0.12
1	CHECK USE OF ARRAY ..... - WARNING ONLY	72	4.25
3	CONSTANT REQUIRED FOR DATA	0	0.00
1	WRITE MISSPELLED - WARNING ONLY	1	0.06
3	BCD OUTPUT RECORD TOO LONG - ... AT OCTAL ... - EXECUTION TERMINATED	1	0.06
2	...... DIMENSIONED TWICE - ERROR IGNORED	6	0.35
		1693	100.00

2.3 Fortran (Frequency of Statements) Usage Analysis

We have analysed 1047 jobs to determine the frequency of use of Fortran statements. Slight inaccuracies are introduced due to the use of the UNLIST facility on 27 programs. The system statements for these 27 jobs are included, although the corresponding Fortran statements are not available on the output tape.

TABLE IV

Analysis by card type

Total cards analysed: 150,934
(these included system cards, blank cards, etc)

	Number	Percentage
System cards:	3,225	2.14
Fortran cards:	146,967	97.37
Blank cards:	742	0.49
	150,934	100.00

Fortran cards can further be classified as:-

	Number	Percentage
Fortran statements:	141,672	96.40
Continuation statements:	5,295	3.60
	146,967	100.00

Fortran statements can be classified as:-

	Number	Percentage
Program statements:	124,097	87.59
Comment statements:	17,575	12.41
	141,672	100.00

From table IV we can see that an average number of statements per program is about 144.

We now examine the distribution of various Fortran statements from table V. In the analysis the statements like ENDFILE, REWIND, BACKSPACE, PAUSE are not mentioned because out of the 1047 jobs, not even a single job contains them. Even the informatted READ statement has not been detected. Among the statements, arithmetic assignment dominates, followed by a DO statement. It should be noted that the various control statements like DO, GOTO, IF, CALL, RETURN are frequently used and they are about 26% of the total statements. The statements FUNCTION, COMPLEX, EQUIVALENCE, ENTRY, BLOCK DATA, EXTERNAL, LOGICAL, NAMELIST, and PRINT were very little used.

This analysis has interesting implications for Fortran teachers. It seems clear that in the early stages, many instruction types can be omitted entirely from the instruction program in favour of increased concentration on assignment and control of the simplest kind. Although similar measurements on more experienced user's programs indicate a similar neglect of many types of instruction, it should be pointed out that we cannot assume that more experienced users do not use the more unusual parts of a programming language from choice. It may be that they remain ignorant of the potential advantages of the more advanced facilities due to poor or non-existent training.

Table V, which follows, shows the number of statements of each type found from the 1020 user's programs and their percentage proportion:

TABLE V

Frequency of Fortran Statements

Statement	Numbers	Percentage
Arithmetic Assignment	55,841	45.00
DO	12,360	9.96
WRITE	5,787	4.66
GOTO	4,562	3.68
FORMAT	7,397	5.96
FUNCTION	31	0.02
IFC	8,205	6.61
INTEGER	275	0.22
RETURN	2,466	1.99
REAL	409	0.33
READ(5,	1,384	1.12
READ	0	0.00
CONTINUE	8,238	6.64
COMMON	2,295	1.85
COMPLEX	61	0.05
CALL	5,154	4.15
END	3,079	2.48
EQUIVALENCE	39	0.03
EXTERNAL	86	0.07
ENTRY	63	0.05
DIMENSION	2,680	2.16
DATA	427	0.34
DOUBLE PRECISION	278	0.22
SUBROUTINE	2,004	1.61
STOP	917	0.74
BLOCK DATA	17	0.01
LOGICAL	11	0.01
NAMELIST	30	0.02
PRINT	1	0.00
	124,097	100.00

2.4 Further Analysis

As a back-up to the detailed analysis for the system performance and for statement frequency, and also data to obtain more information on the various types of errors made by the PUFFT users, a further 352 complete programs have been analysed. The results are displayed in table VI.

There is surprisingly a similar situation regarding the core management. This situation agrees with the previous one, that most of the jobs (about 90%) lie between 0 - 8K words, and the maximum number of jobs occupy core below 1000 words of core store. (See table VI.)

From table VII we can see that the execution time is less than the previous analysis. This could be explained by the fact that there are a few compilation errors more than before. Regarding the execution percentage it is comparable with the previous analysis. Here the successful jobs are 50%. 11.9% of jobs went into execution but were terminated because of the excess output. Approximately 17% contain execution errors and 21% compilation errors. The total number of compilation errors are abnormal in this. They average about 21.5 per program.

It has already been shown that out of the total number of 352 jobs, 22 jobs are without listings. In this case the total number of cards analysed, including the system cards and blank cards, amounts to 26,503.

Out of them

	Number	Percentage
System cards:	842	3.18
Fortran cards:	25,585	96.53
Blank cards:	76	0.29
	26,503	100.00

In Fortran cards there are -

Fortran statements:	24,543	95.93
Continuation statements:	1,042	4.07
	25, 585	100.00

Among the Fortran statements there are -

Program statements:	20,971	85.45
Comment statements:	3,572	14.55
	24,543	100.00

It should be noticed here that the average number of statements per program is equal to 78.

The various other program statements are further classified in table VIII. From this table it could be observed that the percentage of arithmetic assignment statements is about 3.5% higher than the previous analysis and this could be the reason for the increase in the compilation time compared to the previous one.

TABLE VI

Core Store Management

CORE IN K (K = 1000 words)			Number of Jobs	Percentage
0	-	K	126	35.80
K	-	2K	48	13.64
2K	-	3K	39	11.08
3K	-	4K	19	5.40
4K	-	5K	44	12.50
5K	-	6K	11	3.13
6K	-	7K	10	2.84
7K	-	8K	21	5.96
8K	-	9K	21	5.96
9K	-	10K	5	1.42
10K	-	11K	2	0.57
11K	-	12K	5	1.42
12K	-	13K	1	0.28
			352	100.00

Table VII gives a summary of performance of record data samples for 2 days of PUFFT runs in September 1970.

TABLE VII

Overall System Statistics - Second Batch

Time required for compilation of 352 jobs:	29.462 minutes
" " " execution " 352 "	323.530 "
Average compilation time per program:	5 seconds
" execution " " " "	55.1 "

	Number	Percentage
Total jobs:	352	-
Jobs without listing:	22	6.22
Jobs went into execution:	278	78.98
Jobs terminated by stop:	176	50.00
Jobs terminated:	218	61.93
Total error messages:	2,425	-
Total execution messages:	672	27.71
Subprograms compiled from DISK:	60	2.47
Errors of level 0:	756	31.18
Errors of level 1:	74	3.05
Errors of level 2:	178	7.34
Errors of level 3:	1,417	58.43
Programs with compilation errors:	74	21.02
Average number of compilation errors per program:	21,62	

Table VIII gives the frequency of the various Fortran features for the second survey.

TABLE VIII

Fortran Statement Analysis - Second Batch

Statement	Number	Percentage
Arithmetic Assignment:	10,203	48.65
DO:	1,974	9.41
WRITE:	1,392	6.64
GOTO:	754	3.60
FORMAT:	1,597	7.62
FUNCTION:	5	0.02
IF(:	1,250	5.96
INTEGER:	22	0.10
RETURN:	185	0.88
REAL:	84	0.40
READ(5, :	484	2.31
READ(:	2	0.01
CONTINUE:	1,031	4.92
COMMON:	197	0.94
COMPLEX:	18	0.08
CALL:	529	2.55
END:	388	1.85
EQUIVALENCE:	5	0.02
EXTERNAL:	23	0.11
ENTRY:	0	0.00
DIMENSION:	335	1.60
DATA:	97	0.46
DOUBLE PRECISION:	30	0.14
SUBROUTINE:	194	0.93
STOP:	162	0.77
BLOCK DATA:	3	0.01
LOGICAL:	6	0.03
NAME LIST:	1	0.00
PRINT:	0	0.00
	20,971	100.00

2.5 Fortran Course Survey

In the previous sections we have surveyed the performance of the PUFFT system used by various students and research workers who are moderately experienced in writing programs. In this section our object is to acquaint ourselves with the performance of students who are learning to program.

Tables IX to XIII present information from the performance records collected from student programs submitted for the Fortran course given by the Imperial College Computer Unit from September 28th to October 2nd. This course was an introductory programming course with an enrolment of 131 students. One observation which might be of interest in this summary is the distribution of various error messages and the various Fortran statements.

The distribution of program size is given in table IX. The number of jobs in each storage requirement category is shown. 471 jobs out of 478 occupied store between 0 to 1000 words of storage. As expected the execution time is much less than the compile time, since the course is introductory.

Regarding success and failure ratios, it could be concluded from table X that about 59% went into execution and out them 49% were successful. About 8% of the jobs went into execution but they were terminated because of time trap (there is an automatic limit for the time that a program can take). This might have been caused by permanent looping. The possibility of excess output does not arise here since the examples set for the programming course do not take many lines of output. About 1.4% contain execution errors and 41% compilation errors. Surprisingly, the average compilation errors per program are not many, about 3. The average number of statements per program is about 29.

The position of the various Fortran statements is shown in tables XI and XII. From these it can be observed that the arithmetic assignment statements frequency is about 3% less than the previous surveys. The next statements which are in commanding position are FORMAT and WRITE. Even the READ statement is frequently used. Altogether the I/O statements are about 21% and hence we can safely emphasize that proper care should be taken in teaching the I/O statements. This is particularly relevant since 23.5% of errors are due to incorrect FORMAT statements (see table XIII).

Most of the errors that have been committed are compilation errors. Most frequent are:

* error in Format at ...
* mixed type operation ...
* at octal ... , ... used - undefined subroutines or variable or statement number
* statement number required
* statement type error

Errors due to wrong format specification have been observed as the most frequent. It would be interesting to know how many times this error has been caused due to the wrong specification of H-format, but this is not available from the analysis. There are few spelling mistakes and these are not of a serious nature, since they produce only warning messages. But the arithmetic faults and type mistakes are worth noting carefully since they cause termination of the program.

A shortcoming of the performance reporting system is that the present procedures for submitting runs do not collect and retain enough information to trace the history of a student's trial runs for a particular problem assignment. Although such information would be extremely valuable, obtaining it would present the impractical requirement that each student accurately reports details of each of his runs, such as assignment number, trial number, and purpose of trial.

TABLE IX

Core Store Management

Core in K	Number of Jobs	Percentage
0 - K	471	98.54
K - 2K	7	1.46

Time:

Average compile time per program:	1.1 seconds
Average execution time per program:	0.5 seconds

TABLE X

Total time required for the compilation of the jobs:	8.82 minutes	
Total time taken for the execution of the jobs:	3.53 "	
		Percentage
Number of jobs:	478	
Number of jobs without listing:	0	
Number of jobs went into execution:	280	58.57
Number of jobs terminated by stop:	234	48.95
Number of jobs terminated:	273	57.11
Total number of error messages including execution messages:	2232	
Various execution statements:	787	35.26
Errors of level 0:	816	36.56
Errors of level 1:	64	2.86
Errors of level 2:	350	15.68
Errors of level 3:	1002	44.90

The total number of statements per program is approximately 29.

TABLE XI

Total number of cards analysed: 13,964

	Number	Percentage
System cards:	1,584	11.35
Fortran cards:	12,377	88.63
Blank cards:	3	0.02
	13,964	100.00

Fortran cards can further be classified as:

	Number	Percentage
Fortran statements:	12,188	98.48
Continuation statements:	189	1.52
	12,377	100.00

Fortran statements can be classified as:

	Number	Percentage
Program statements:	11,286	92.60
Comment statements:	902	7.40

TABLE XII

Number of statements of each type found from 478 user's programs

Statement	Number	Percentage
Arithmetic assignment:	5082	45.03
DO:	72	0.63
WRITE:	1294	11.46
GOTO:	258	2.29
FORMAT:	1537	13.62
FUNCTION:	-	-
IF(:	485	4.29
INTEGER:	3	0.03
RETURN:	141	1.24
REAL:	6	0.05
READ(5, :	664	5.89
READ:	-	-
CONTINUE:	55	0.49
COMMON:	272	2.41
COMPLEX:	-	-
CALL:	273	2.41
END:	418	3.71
EQUIVALENCE:	-	-
EXTERNAL:	-	-
ENTRY:	-	-
DIMENSION:	280	2.48
DATA:	-	-
DOUBLE PRECISION:	-	-
SUBROUTINE:	183	1.62
STOP:	249	2.22
BLOCK DATA:	-	-
LOGICAL:	-	-
NAMELIST:	-	-
PRINT:	-	-
OTHERS:	14	0.13

TABLE XIII

Various error statements, their frequency and percentage

Level	Statement	Frequency	Percentage
3	ILLEGAL USE OF BCD CHARACTER - EXECUTION DELETED	0	0.00
3	ILLEGAL USE OF LOGICAL UNIT 00006 - EXECUTION DELETED	0	0.00
3	ILLEGAL USE OF BCD CONSTANT - EXECUTION DELETED	0	0.00
3	ILLEGAL USE OF LOGICAL UNIT 00005 EXECUTION DELETED	3	0.21
3	ILLEGAL USE OF PERIOD - EXECUTION DELETED	6	0.42
3	ILLEGAL USE OF FUNCTION NAME - EXECUTION DELETED	0	0.00
3	ILLEGAL USE OF LOGICAL OPERATION - EXECUTION DELETED	2	0.13
3	ILLEGAL BCD CHARACTER - EXECUTION DELETED	17	1.18
2	ILLEGAL CARD BELOW IGNORED	64	4.43
1	ILLEGAL COMMA IGNORED - WARNING ONLY	2	0.13
3	ILLEGAL CHARACTER IN DATA - AT OCTAL ... - EXECUTION DELETED	9	0.63
3	ILLEGAL SUBSCRIPT AT ... - EXECUTION TERMINATED	9	0.63
3	ILLEGAL FUNCTION OR SUBROUTINE NAME - EXECUTION DELETED	1	0.07
3	ILLEGAL NUMBER OF SUBSCRIPT FOR ... - EXECUTION DELETED	1	0.07
2	ILLEGAL SEQUENCING OF STATEMENT - ERROR IGNORED	24	1.66
2	ILLEGAL STATEMENT NUMBER - ERROR IGNORED	9	0.63
3	ILLEGAL PUNCTUATION - EXECUTION DELETED	15	1.04
3	ILLEGAL REAL CONSTANT - " "	24	1.66
3	ILLEGAL VARIABLE NAME - " "	0	0.00
3	ILLEGAL ARRAY NAME .. - " "	7	0.48
3	ILLEGAL OPERATOR - " "	18	1.26
3	ILLEGAL END OF STATEMENT - "	1	0.07
3	ILLEGAL INTEGER VARIABLE OR CONSTANT - EXECUTION DELETED		

TABLE XIII cont.

Level	Statement	Frequency	Percentage
3	ILLEGAL NUMBER OF SUBSCRIPT FORMAT - EXECUTION DELETED	0	0.00
3	INCONSISTENT EQUIVALENCE OF VARIABLE - EXECUTION DELETED	0	0.00
3	INDEX AND LIST LENGTH INCONSISTENT - EXECUTION DELETED	0	0.00
3	INTEGER CONSTANT SUBSCRIPT REQUIRED AT .. - EXECUTION DELETED	8	0.56
3	ERROR IN FORMAT AT ... - EXECUTION DELETED	340	23.54
0	EOF READ IN UNIT 00005 - " "	25	1.74
2	EOF ERROR - CALL EXIT USED - WARNING ONLY	6	0.42
2	END CARD MISSING - CALL EXIT USED "	22	1.52
1	END FILE MIS-SPELLED - WARNING ONLY	4	0.29
3	PARENTHESIS ERROR - EXECUTION DELETED	42	2.91
3	PARITY ERROR ON UNIT 00005 - "	0	0.00
3	PROGRAM OR ARRAY TOO LONG - "	0	0.00
3	PUNCTUATION ERROR - "	1	0.07
1	PAUSE MIS-SPELLED - WARNING ONLY	2	0.13
2	STATEMENT NUMBER REQUIRED - STATEMENT CANNOT BE REACHED - WARNING ONLY	96	6.65
3	STATEMENT TYPE ERROR - EXECUTION DELETED	90	6.22
3	SYSTEM ERROR AT OCTAL .. - " "	10	0.70
3	STATEMENT NUMBER ... USED TWICE - "	15	1.04
3	STATEMENT NOT COMPLETE - "	7	0.48
2	SUBROUTINE ... DEFINED TWICE -	1	0.07
3	MIXED TYPE OPERATION - EXECUTION DELETED	110	7.61
2	MISSING OR ILLEGAL JOB CARD	77	5.32
3	MAIN PROGRAM MISSING - EXECUTION DELETED	1	0.07
2	MAIN PROGRAM DEFINED TWICE - FIRST MAIN PROGRAM USED	24	1.66
0	UNDERFLOW AT ... AC and MQ - WARNING OUTPUT FIVE TIMES ONLY	7	0.48
3	VARIABLE OR CONSTANT TOO LONG - EXECUTION DELETED	3	0.21
3	VARIABLE USED ... AS FUNCTION - "	29	2.00
3	VARIABLE ... USED AS SUBROUTINE - "	2	0.13
0	OVERFLOW AT ... - WARNING OUTPUT FIVE TIMES ONLY	0	0.00
3	NESTED DO'S OVERLAP - EXECUTION DELETED	0	0.00

TABLE XIII cont.

Level	Statement	Frequency	Percentage
1	CHECK USE OF ARRAY - WARNING ONLY	36	2.49
3	CONSTANT REQUIRED FOR DATA - EXECUTION DELETED	0	0.00
1	WRITE MIS-SPELLED - WARNING ONLY	1	0.07
1	FORMAT MIS-SPELLED - " "	2	0.13
1	DIMENSION MIS-SPELLED " "	2	0.13
3	AT OCTAL ... , .. USED - UNDEFINED (SUBROUTINE) OR (VARIABLE) OR STATEMENT NUMBER - EXECUTION DELETED	118	8.16
3	ARRAY ... USED AS SUBROUTINE - EXECUTION DELETED	1	0.07
3	ARRAY ... NOT DEFINED - EXECUTION DELETED	58	4.01
3	HOLLERITH COUNT TOO LONG - " "	11	0.76
3	BCD OUTPUT RECORD TOO LONG - ... AT OCTAL - EXECUTION TERMINATED	1	0.07
2	FUNCTION ... DEFINED TWICE - ERROR IGNORED	5	0.34
2	... DIMENSIONED TWICE - ERROR IGNORED	46	3.18
2	... MUST BE VARIABLE NAME - EXECUTION DELETED	1	0.07
1	... DEFINED IN COMMON TWICE - FIRST USED	22	1.52
		1445	100.00

3. DESCRIPTION OF THE ANALYSER PROGRAM

The input to the analysing program is the system output tape produced by PUFFT. It is therefore necessary to process a sequence of records, each one representing one line of the printed output which is normally returned to the user. Fig. 2 gives a typical sequence of output records. In fact the physical record on tape is of variable length and does not necessarily correspond to the logical record to be processed. A simple buffering system overcomes this. The reading of the system tape naturally must be carried out by an assenbly code routine. A description of this routine, called VARRED, is given in section 3.6. After a record has been transferred to main store, its tape is recognised and relevant action taken as follows:

(a) $JOB This card marks the start of each job. So a job count is made and also checked for $UNLIST cards in order to separate out the number of jobs without listings.

(b) Other system records These are detached as starting with a $ character and a single count of occurance is encountered.

(c) Error records These are recognised by the pattern '**n** at the beginning, where each n is an integer between 0 - 3. According to the value of n the level of the error is known.

(d) Fortran statements Details of this analysis is discussed in section 3.2. Section 3.1 gives the description of tree.

3.1 Description of Tree

Usually the process of syntax analysis of any high level language involves the determination of the structure of a statement by using a processing algorithm which expresses the total structure of all possible statements in the language. In fact this structure could be expressed more or less explicitly as a tree structure. This should be provided once for all as part of the algorithm and should also be unaltered in the process of analysis so every statement is analysed as if it were the first ever. The initial structure of the tree provided for the analysing algorithm expresses the structure of all 'expected' language statements

Layout of the various types of record on the PUFFT output tape. Each record occupies 16 or 17 words.

```
1      2     3     4     5     6     7     8     9     10    11    12    13    14    15    16    17    18
                                           Time  Output lines
1      $JOB     Jobnumber    name    PUFFT   3.0     500                  A01802                 Date          ≠
       $*    NAMECARD                                                                                          ≠
       $EXECUTE        PUFFT                                                                ≠
       $IBJOB                                                                               ≠
        IMP. COL. PUFFT VERS  date                       ≠1

                       Any Fortran statement would have this type of layout on the system output tape

             DIMENSION                                                                60321      ≠


'**n**       EXECUTION      ≠
'**n**       EXECUTION TERMINATED BY STOP     AT OCTAL 62035      ≠

1  TIME FOR JOB     0.27MINS      Date               A01803100201 68          PUFFT 3.0   003000     ≠

          385 LINES OUTPUT.       ≠
OCOMPILATION  0.0242MINS          EXECUTION  0.2478MINS         3111DATA    5228PROGRAM          ≠
```

FIG. 2.

and also all expected error statements too. The tree should also carry a memory of the frequency of all statements. The general principle is that the tree structure expressing the language and error statements structure is no longer static, but dynamic. The structure is provided initially and is altered in many different ways as a result of the continuing process of matching incoming statements. With all the possible incoming statements it can be seen that the tree will develop a complex and extensive structure. The matching process may then become too time consuming to be practical. There is thus a fundamental requirement for an automatic restructuring process which aims continually to minimise the complexity of the total matching process. This could be realised by the use of a 'branch swopping' technique. The set of alternative nodes at any point in the tree which represents the set of alternative structures which might be present at a certain point in the input statement, can be continually rearranged so that those most frequently relevant are nearest to the root of the tree, and therefore are reached more quickly.

When a node is transferred in this way, its attachments to lower nodes must remain undisturbed. Based on this philosophy, the two routines MTRPUT and TRSWOP have been written. (Refer to sections 3.3 and 3.4 for details).

These tree structures are produced and manipulated with the help of the various Minislip routines. For details about the various character handling routines and Minislip routines, refer to Appendix I and II.

3.2 Description of the Main Program

Before describing the main routine it is necessary to specify in outline the function of the various subroutines called as follows:

(1) SLIP routines (for details see Appendix I).

(2) MTRPUT routine which helps to form a tree whose various branches are the various key words of the Fortran statements, such as READ, WRITE, FORMAT, CALL, etc. and the key words of various possible error statements, for example, ILLEGAL PUNCTUATION, STATEMENT NUMBER REQUIRED, etc. A detailed description of this routine is given in the next section.

(3) TRSWOP routine. This subroutine does the job of matching the incoming statements against the key words in tree form and classifying the various statements from the PUFFT output tape. Every time a statement (key word) is recognised the count of the number of occurences of the start is increment. If the key word corresponding to a particular output statement is not matched then the "new" keyword is added to the tree.

(4) TPRINT routine. This subroutine prints out the frequency of various statements analysed from PUFFT output tape.

(5) VARRED. An IOCS routine helps to open the PUFFT output tape and deblocks it. This mainly brings out the various statements on the PUFFT output tape and stores them in the temporary buffers.

(6) BCDINT is a FUNCTION subprogram whose main job is to convert the BCD integer characters into integers.

Initially, a collection of cells (about 10,000) called the List of Available Space (LAVS) are created with the help of a SLIP routine called INITAS. The Fortran array A is declared at the beginning of the program to provide enough space for all cells. Out of the LAVS a cell is detached and it is addressed as a list header LSWOP. (This prints to the top of the tree.) The various branches of the tree are the Fortran key words such as READ, etc. In a similar way, a tree with the top accessed via LSWIP is formed to represent the various possible error statements as its various branches.

The correct magnetic tape unit for input is selected, and the program now starts to read the records from the PUFFT tape, one at a time, into a buffer, using the VARRED routine. The buffer is then tested for the various types of record and action is taken as follows:

(1) A $JOB record. This indicates the beginning of a new job unless it also contains ENDRUN, which will indicate the final card to be processed. As each normal $JOB record is encountered, the count of the number of jobs is incremented.

(2) A report from the system during compilation or execution. These are recognised by the presence of the sequence **n** where n is 0, 1, 2 or 3. An execution report will have EXECUTION, EXECUTION TERMINATED or EXECUTION TERMINATED BY STOP. All others will be error messages with the level of severity indicated by 0, 1, 2 or 3.

(3) Job timing. This is recognised by "TIME FOR JOB" and the relevant time is recorded. This gives total time for compilation and execution.

(4) Separate compile and execute times. These occur in a separate record recognised by the presence of "OCOMPILATION".

(5) Number of output lines. Recognised by "LINES OUTPUT".

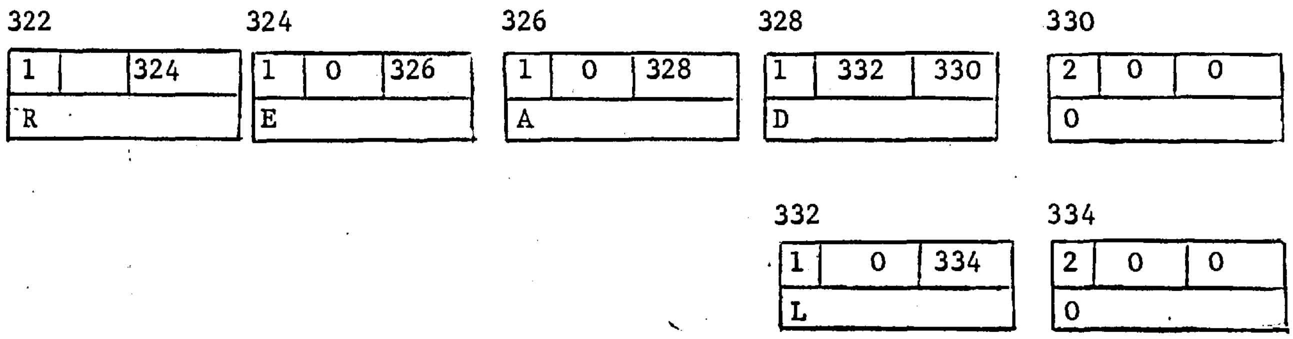
The remaining part of the analysis program tests for the various types of Fortran statement. For this purpose the record which has been stored in the buffer with six characters to each word is unpacked to Al format in an array called LINE. This is then tested in succession for comments, continuation, and possible completely blank records. The statements employing an equal sign but not DO statements are next discovered and the relevant count is incremented. Finally, the rest of the Fortran records are matched on the Fortran key word tree.

The main program is written in ASA Fortran. The subroutines which it makes use of are described below. They also are written in Fortran except as stated otherwise.

3.3 Subroutine MTRPUT

The purpose of this subroutine is to create a tree structure in which each branch represents a permissible string of characters which will later be matched against a PUFFT record. Two trees are created, one containing the error statement, the other containing the standard Fortran key words. The tree is built up from a sequence of data cards, each specifying a branch. Each call of MTRPUT results in the addition of a new branch to the tree whose root is specified in the call.

Part of the tree of Fortran key words produced by MTRPUT might be as follows:

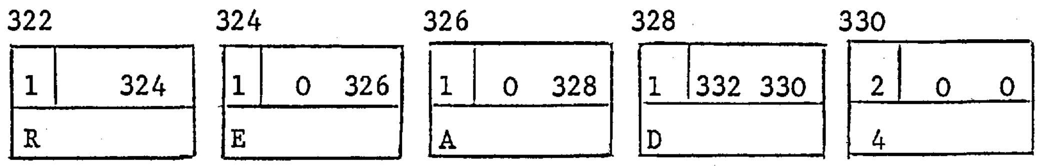


The subroutine is written in Fortran with calls to the Minislip routines for setting up the list structure.

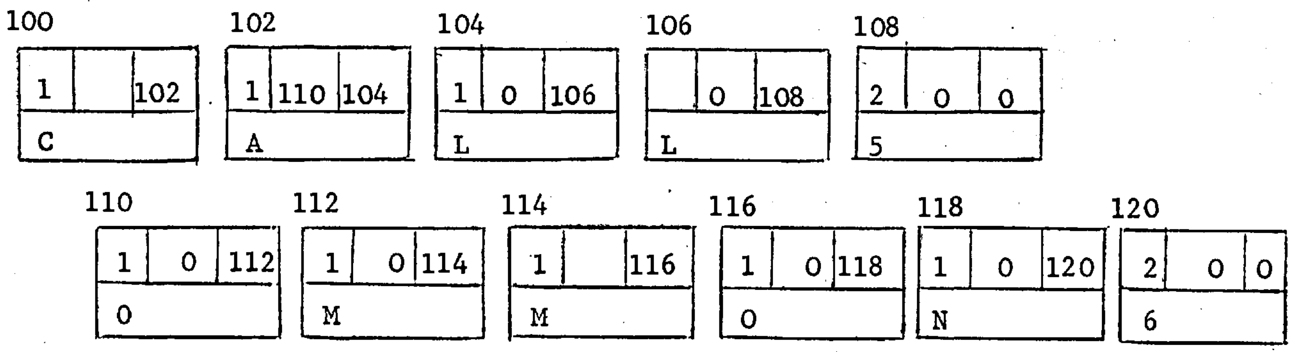
3.4 Subroutine TRSWOP

The main purpose of this routine is to obtain the frequency of various Fortran and error statements contained in the PUFFT record. The characters of a statement from PUFFT output tape is being matched with the existing tree created by MTRPUT. On finding a successful match the count of such statements is made. In matching for a successful match, it swops the various branches of the tree in order to minimise the complexity of the total matching process.

Suppose before calling TRSWOP the number of READ statements have been counted as 3, later, when it is called again to match the Fortran key word READ, then the addition is done as follows:



The branch swopping is carried out as follows: Suppose the part of the tree containing Fortran key words CALL and COMMON is as given below:



Later, on calling TRSWOP, it has to match with the input statement COMMON, a Fortran key word, then it swops the whole branch COMMON by lowering the branch ALL, a part of CALL.

If the input statement cannot be matched with any of the existing branches of the tree, then an additional branch will be created by this routine.

The subroutine is written in Fortran with calls to the Minislip routines for setting up the list structure.

3.5 FUNCTION Subprogram BCDINT

Name: BCDINT.

Purpose: For picking up the numbers from the PUFFT output tape whose characters are in BCD characters.

Example of use: Y = BCDINT (BCDNUM, NUMCHR).

Arguments: BCDNUM - BCD number to convert NUMCHR - Number of characters to be converted.

Description: BCDINT is a FUNCTION subprogram and when it is used it will return a value which is the binary integer equivalent of the BCD number BCDNUM. The value is stored in Y. Any non-numeric characters are treated as zeros, but blanks are ignored. (Note: Y and BCDINT must be declared as same type, i.e. both real or both integer, in the calling program.)

Language: MAP

3.6 Subroutine VARRED

Name: VARRED

Purpose: For opening the PUFFT output tape and to deblock the records.

Example of use: CALL VARRED (NTAPE, BUFF, NUMWRD, IEOF).

Arguments: NTAPE - Logical tape unit number. BUFF - Buffer array into which the words of the next Logical Record are placed. NUMWRD - Number of words found in Logical Record is placed in it. If zero on calling VARRED, the present Physical Record is truncated and the next Logical Record is taken from the next Physical Record. IEOF - Address to which to transfer on reading an end of file.

Description: On entering the VARRED, IOCS subroutine the parameters are stored and NUMWRD is tested for zero, (if zero, the end of the present Physical Record is forced). The Logical tape unit is converted to the address of the File Control Block (FCB) and the file is tested for open, (open if not). The value of the FCB is compared with the value the last time the routine was called. If they are not equal a new Physical Record is read into internal system buffers and its size contracted. This size is tested for zero and another Physical Record is read if it is. Words are now unpacked from the internal buffer into BUFF until either an end of Logical Record character (072 at the end of a word) or the end of Physical Record is encountered. The number of words found is stored in NUMWRD and the return is made (after adjusting the number of words left in the internal buffer).

Language: IOCS (MAP)

3.7 Subroutine TPRINT

This subroutine is written in Fortran with calls to the Minislip routines for printing out the trees which were created by MTRPUT and TRSWOP.

On calling this routine it starts printing out the branch of the tree till it encounters the end of the branch. The value in the data word of the end of branch is pointed out. It indicates the frequency of that particular statement. Then it checks for the other branches of the tree and prints out the whole tree.

4. CONCLUSIONS AND FUTURE DEVELOPMENT OF THE METHODS

We feel that this type of analysis should prove of fundamental importance to the designers of new computing systems and should also indicate ways in which the user can be trained to make more effective use of the system. The most pertinent aspect of this analysis is that even an effective and efficient system like PUFFT can produce error messages which are very ambiguous. For example, during the course of the analysis we came across an error "inconsistent equivalence of variable" which was observed more frequently than any other. Without considerable research into the compiler it does not seem possible to determine the meaning of this statement or to be certain of the user program faults which give rise to it. The situation is similar for the error statement: "* illegal use of BCD character". Both of these errors in fact lead to an immediate deletion of execution and so the need for further analysis is pressing.

Another particularly significant result is that the majority of programs could perfectly well run on a system with more limited storage and this could lead to a reconsideration of the type of machine necessary for teaching purposes.

In conclusion, it appears to the author that what seems at first sight to be a comparatively mundane job of analysis on data of limited interest has yielded some surprising results which should be of considerable concern to systems designers. The work has also provided valuable experience in comparatively complex data structures and analysis, and valuable insight into methods of statement recognising. It is hoped that the programming system presented here will be employed continually to monitor the effectiveness of the PUFFT system.

APPENDIX I

Character Handling in Fortran

Although Fortran was designed for numerical calculations, a great deal of useful work can be carried out in this language in the field of non-numeric data processing or text manipulation, in which the quantities being processed are not integer or floating-point numbers, but strings of characters. Areas in which Fortran has been found useful include information retrieval, automatic documentation, the automatic generation of flow charts and even the construction of compilers. In scientific calculations, it is often necessary to produce graphs or contour maps on the line printer, and here again Fortran can be employed.

The minimum unit of information manipulated directly in Fortran is one machine word. If individual characters are to be processed separately, then they canbe dealt with in two ways:

1) By arranging to store one character in a word. Since a character requires 6 bits of a word in the IBM 7094 the rest of the bits must be filled in with blank characters. This method involves using A1 format exclusively for input and output.

2) It is possible to fill the word with characters and have special assembly code routines available for manipulating strings of characters and for packing and unpacking words.

Character Handling routines in PUFFT and IBSYS

There are three routines available for manipulating strings of characters and for packing and unpacking words. Strings are stored in arrays, each word contains six alpha-numeric characters, except the last which may contain fewer than six characters (which are stored left-adjusted). A string may be read in as Hollerith information or may be compiled in data statement.

The character positions are referred to as being consecutively numbered and each character is referred to by its position in the string.

CALL GET(S,I,T) : Get the Ith character from string S and place it in T (left-adjusted). The last five characters of T are blanks. S is not affected.

CALL PUT(S,I,T) : Put the left-most character of T into the Ith position of string S. The remaining characters in string S are not affected and T is not altered.

CALL KOMPAR(S,T,I) : Compare logically the contents of <u>word</u> S with the contents of <u>word</u> T.

If C(S) are less than C(T), then I = -1
If C(S) are equal to C(T), then I = +0
If C(S) are greater than C(T), then I = +1

<u>Example</u>: Suppose there is a string of 14 characters, (ABCDEFGHIJKLMN) set up in an array called PLACE. PLACE will have to have three words, so DIMENSION PLACE (3).

The first word will contain ABCDEF
The second word will contain GHIJKL
The third word will contain MNbbbb (b = blank)

CALL GET (PLACE , 13 , TEST) will leave TEST with: Mbbbbb.

Then,

CALL PUT (PLACE , 1 , TEST) will leave PLACE(1) with MBCDEF
If S contains ABCDEF and T contains ABCDEF, then CALL KOMPAR(S,T,I) will put -1 in I.

When KOMPAR is used for comparing characters in a word, the values actually compared are the binary representation of the characters in storage. The 7094 code is given below. Notice that successive letters of the alphabet are in ascending binary value, but BLANK is higher than A-R and lower than S-Z. If blanks are to be ignored, then the required sorting order is not simply on magnitude.

Character	Storage	Character	Storage	Character	Storage	Character	Storage
	Octal		Octal		Octal		Octal
1	01	A	21	J	41	/	61
2	02	B	22	K	42	S	62
3	03	C	23	L	43	T	63
4	04	D	24	M	44	U	64
5	05	E	25	N	45	V	65
6	06	F	26	O	46	W	66
7	07	G	27	P	47	X	67
8	10	H	30	Q	50	Y	70
9	11	I	31	R	51	Z	71
blank	60	+	20	-	40	0	00
+	13	.	33	$	53	,	73
'	14	)	34	*	54	(	74

APPENDIX II

Minislip Routines

SLIP (Symmetric List Processor) is a list processing system in which each list cell carries both a forward and a backward link as well as alphabetic or numeric data. The fundamental information module with which SLIP deals is a pair of consecutive words called a cell. Out of the 36 bits contained in an IBM 7094 computer word the first word of the cell has an identifier (ID) field of 6 bits length, the left link field (LNKL) is of 15 bits length, and the right link field (LNKR) is also of 15 bit length. The second word of the cell is used for alphabetic or numeric data. The cell as a whole is referenced by the address of its first word. There are various SLIP routines available at Imperial College to handle lists and trees. There are handouts prepared by Mr. E. B. James, and these are given below.

LISTS and TREES in Fortran

The basic item of information is a cell, which occupies two consecutive 36-bit words.

ID	LNKL	LNKR
	DATA	

The first word contains: A six-bit identification field, ID, which can take values between 0 and 64; Two fifteen-bit address fields, LNKL and LNKR. Generally, the ID field denotes what the cell is used for and LNKL and LNKR provide addresses to connect the cell into the list or tree. The second word is used for alphabetical or numeric data. The cell as a whole is referenced by the address of its first word.

The following ten functions and subroutines, written as one MAP routine with ten entry points called MCOPS perform basic operations on the components of a cell.

The example will make use of the cell with contents as follows:

17	392	68
	A B C D E F	

Assume this cell is stored at locations 1544 and 1545. Also that the Fortran variable NAME is stored at location 1544.

1. Function ID (NAME) obtains the values of the Identification field
2. " LNKL (NAME) " " " " " LNKL address
3. " LNKR (NAME) " " " " " LNKR address

Examples: ID (NAME) = 17 , LNKL (NAME) = 392 , LNKR (NAME) = 68.

4. INHALT (N) obtains the contents of the machine location whose address is the value of the Integer variable N.

5. CONT (N) is exactly the same as INHALT (N), but is a Real function. Both are required to overcome the difficulty of unrequired type changes in assignment statements.

Example: Let the Fortran variable N have the value 1544, that is, the location is assigned to N contains the integer 1544. Then the value of INHALT (N) is 17 392 68 , the value of LNKR (INHALT (N)) is 68, the value of ID (INHALT (N)) is 17. INHALT (N + 1) is the contents of the data word 1545 = ABCDEF.

6. MADOV (NAME). This obtains the machine address of the Fortran variable NAME.

Example: The value of MADOV (NAME) is 1544. The previous six functions have obtained values connected with a cell. The following four routines, which can be called as subroutines or functions, store values in various parts of a cell.

7. STRDIR (I, N). Stores value I in machine location occupied by variable N.

Example: CALL STRDIR (I, N). If I contains ABCDEF, N contains ABCDEF after the call. If used as a Function the process described takes place and also STRDIR (I, N) returns the value ABCDEF.

8. STRIND (I, L). Stores value I in the machine location addressed by the contents of L.

Example: If L has value 1545 and I contains ABCDEF, then CALL STRIND (I, L) puts ABCDEF into machine location 1545. Its value when used as a function is the value of I.

9. SETDIR (I, J, K, N) puts the values of Fortran integer variables I, J, K into the ID, LNKL, and INKR fields of the machine location occupied by Fortran variable N. If any of I, J, K have a negative value, then the corresponding field is unaltered in N. If used as a function, returned value is the new N.

<u>Example</u>: CALL SETDIR (I, J, K, N). If I = 17, J = 392, K = 68, N is set to:

ID↓	LNKL↓	LNKR↓
17	392	68

CALL SETDIR (-1, -1, -1, N) leaves N unaltered.

10. SETIND (I, J, K, L) has the same effect as SETDIR, but the values are set into the machine location whose address is the value of Integer variable L.

<u>Example</u>: If L contains 1544 and I, J, K are as in the previous example, then CALL SETIND (I, J, K, L) sets the machine location 1544 to

17	392	68

All the variables in the previous routines can of course be replaced with arithmetic expressions where appropriate.

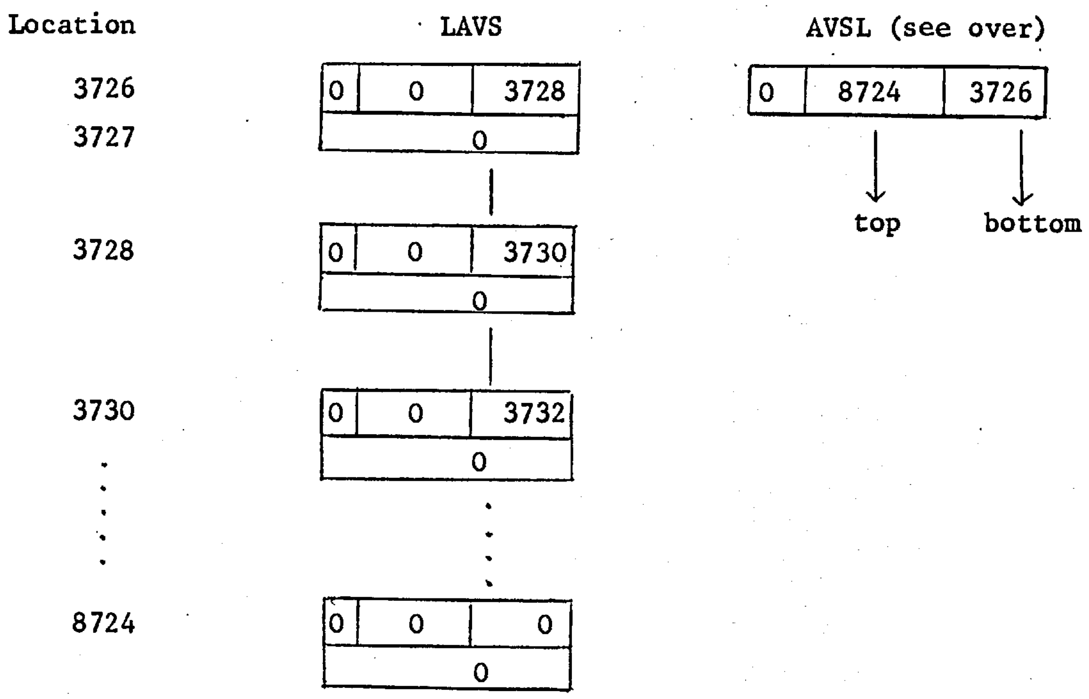
<u>Example</u>: CALL SETIND (ID (NAME), LNKL (NAME), LNKR (NAME), NAME) does nothing at all.

The following routines are written in Fortran IV and reference MCOPS:

1. ERROR(J): Writes error messages. Used only internally to the other routines.

2. INITAS(A, N): A subroutine used at the start of a program to create a collection of cells, called the List of Available Space, or LAVS. The Fortran array A is declared at the beginning of a program, of sufficient size to provide space for all cells to be created.

Example: INTEGER A(5000) followed by CALL INITAS(A,5000) will create a structure with 2500 cells (N/2, or (N-1)/2 if N is odd). If A(1) is stored at 3726, then the structure will be as follows:



A special location AVSL is the pointer to LAVS and carries in its LNKR the address of the first cell and in its LNKL the address of the last cell on the list.

Example: In the list above, LNKR (AVSL) = 3726, LNKL (AVSL) = 8724. If LNKR (AVSL) becomes zero, then there are no cells remaining in LAVS.

2. NUCELL (L). Used as a function or subroutine, detaches a cell from the list of available space. L and the function value are set to its address.

Example: CALL NUCELL (L) with the above LAVS will detach the first cell from LAVS. LNKR (AVSL) becomes 3728 and L receives the value 3726.

3. RCELL (L). A subroutine only; attaches the cell whose address is in L to the LAVS.

The following routines set up and manipulate symmetric lists, which enable the structure to be traced in either direction. The LNKR in each cell carries the address of the succeeding cell in that list, the LNKL carries that of the preceding cell.

4. LIST (M) creates an 'empty' list ready for later use by detaching one cell from LAVS and setting it up as a list 'header'. M (and the function value if used as a function) are set to the address of the header cell. When created, the header cell has its LNKL and LNKR both set to its own address.

Example: CALL LIST (M) with LAVS as in original example. M becomes 3726. LNKL and LNKR of cell 3726 are set to 3726.

Later, when cells are added to the new list, the LNKL and LNKR of the header cell contain the address of the bottom cell and the top cell of the list. The list is referred to via the Fortran name M.

5. LOCT (M). A function only, obtains the address of the header cell of list M.

6. NAMTST (M). A logical function, has value TRUE if M is the name of a list.

7. LISTMT (M). A logical function, has value TRUE if list M is empty.

8. NEWTOP (I, M). Inserts the value of I at the top of the list M. Used as a function, its value is the address of the cell used to store the value I.

9. NOKTOP (M). Removes the top cell from the list M, and returns it to the LAVS. Used as a function, its value is the data item of the cell removed.

10. NOKOFF (L). Removes the cell whose address is the value of L from its surrounding list structure and returns it to LAVS. Used as a function, its value is the data item of the cell deleted.

11. NOKBOT (M). As NOKTOP but removes the bottom cell in the list.

12. NXTRGT (A, L). Inserts a cell immediately below the cell whose address is the value of L. Called "nextright" because LNKR is used to point down the list, LNKL points up. Used as a function, its value is the address of the cell inserted.

13. MTLIST (M). This removes all the cells on list M, leaving it empty, that is, with only a header. Used as a function, its value is the contents of Fortran variable M.

REFERENCES

1. Rosen, S., Spurgeon, R. A., Donnelly, J. K. - PUFFT - The Purdue University Fast Fortran Translater. Comm. A.C.M., Vol. 8, P.661, (1965).

2. Shantz, P. W., German, R. A., Mitchell, J. G., Shirley, R. S. K., Zarnke, C. R., - WATFOR - The University of Waterloo Fortran VI Compiler. Comm. A.C.M., Vol. 10, P.41, (1967).

3. American Standards Association: Fortran Vs Basic Fortran - A programming language for information processing in automatic data processing systems. Comm. A.C.M., Vol 7, P.591, (1964)

4. American Standards Association: Appendixes to ASA Fortran Comm. A.C.M., Vol. 8, P.287, (1965).

5. Leavenworth, B. M., Fortran VI as a syntax language. Comm. A.C.M., Vol. 7, P.72, (1964).

6. Weizenbaum, J., Symmetric List Processor, Comm. A.C.M., Vol. 6., P.524, (1963).

7. Pritsker, A. A., Kiviat, P. J. SIMULATION with GASP II - A Fortran based simulation language. Prentice-Hall, (1969).

8. Healy, M. J. R., Bogert, B. P., Fortran Subroutines for time-series analysis, Comm. A.C.M., Vol. 6, P.32, (1969).

9. Pyle, I. C., Character manipulation in Fortran, Comm. A.C.M., Vol. 5, P.432, (1962).

10. Healy, M. J. R., Towards Fortran IV. Computer Journal, Vol. 11, P.169, (1968).

11. Hendry, D. F., Samet, P. A., Towards Fortran IV., Part 2. Fortran in the modern world, Computer Journal, Vol. 12, P.218, (1969).

12. Atlas Computing Service. University of London, Fortran IV Manual, (1967).

13. I.B.M. Corp. 7090/7090 IBSYS Operating System, Version 13, Fortran IV Language. File No. 7090-25, Form C28-63903-3, (1966).

14. I.B.M. Corp. System /360 Fortran IV Language. File No. 5360-25, Form C28-6515-4, (1967).

15. Control Data 6400/6500/6600 Computer System. Fortran Extended Ref. Manual. Publication No. 6017 6600, (1967).

16. I.C.L. 1900 series Fortran Technical Publication No. TL 1167, (1966).

REFERENCES cont.

17. American Standard Fortran, U.S.A. Standards Institute. X3.9 March 7 (1966).

18. American Standard Basic Fortran, U.S.A. Standards Institute X3.10 March 7 (1966).

19. James, E. B., Principles of adaptive analysis. Computing Science Research Report No. 20. Imperial College, London, (1970).

```
          B00805      10016034                                                   DATE 09/22/70

$IBFTC MAIN     DECK

MAIN          -  EFN   SOURCE STATEMENT   -   IFN(S)   -

C
        INTEGER BCDINT, CARD(80),A(20000),BUFF(120),LINE(80),RCARD
        DIMENSION IREAD(4),IWRITE(5),IFORMT(6),IDAT(4)
        DIMENSION KORD(80)
        DATA IREAD/1HR,1HE,1HA,1HD/
        DATA IWRITE/1HW,1HR,1HI,1HT,1HE/
         DATA IFORMT/1HF,1HO,1HR,1HM,1HA,1HT/
        DATA IDAT/1HD,1HA,1HT,1HA/
        DATA SMASKI/077777700000/
        DATA SMASKJ/001254600000/
        DATA RMASKJ/077777007777/
        DATA RMASKI/0145454005454/
        DATA SMASI1/077000000000/
        DATA RMASJ1/001000000000/
        DATA SMASI2/077777777000/
        DATA RMASJ2/053414622000/
        DATA SMASI3/077777777777/
        DATA RMASJ3/0256725236463/
        DATA SMASI4/077777700000/
        DATA RMASJ4/031464500000/
        DATA SMASI5/000000007777/
        DATA RMASJ5/000000006325/
        DATA SMASI6/077777777777/
        DATA RMASJ6/0514431452163/
        DATA SMASI7/077777777777/
        DATA RMASJ7/0002346444731/
        DATA SMASI8/077777777777/
        DATA RMASJ8/0432163314645/
        DATA SMASI9/0000000777700/
        DATA RMASJ9/0000000227000/
        DATA SMASK1/0777777770000/
        DATA RMASL1/0626346470000/
        DATA AMASKI/0000000777777/
        DATA AMASKJ /0000000254524/
        DATA BMASKI/0007777770000/
        DATA BMASKJ/0005164450000/
        DATA CMASKI/0007777777777/
        DATA DMASKJ/0006445433162/
        DATA NC/1HC/
        DATA NEQUAL/1H=/
        DATA NBLANK/1H /
        DATA ND/1H$/
        DATA NDD/1HD/
        DATA NOO/1HO/
        DATA RCARD/6H      /
        WRITE(6,1)
    1   FORMAT(1H1)
        LDEPTH=6
```

```
C  CALL A SLIP ROUTINE TO CREATE A STRUCTURE WITH 10000 CELLS CALLED
C  LIST OF AVAILABLE SPACE
      CALL INITAS(A,10000)
C  DETACH FIRST CELL FROM LAVS,CALL IT AS LSWOP(I.E)COLLECT ADDRESS FOR
C  TOP OF TREE
      CALL NUCELL(LSWOP)
C  SET THE IDENTIFICATION FIELD,LNKL AND LNKR ADDRESS OF MACHINE LOCATION
C  LSWOP INITIALLY AS ZEROS
      CALL SETIND(0,0,0,LSWOP)
      CALL NUCELL(LSWIP)
      CALL SETIND(0,0,0,LSWIP)
   43 CONTINUE
C  READ A CARD WHICH CONTAINS INFORMATION TO BE STORED IN THE MACHINE
C  LOCATIONS ONE CHARACTER IN EACH LOCATION IN TREE FORM
      READ(5,41) CARD
   41 FORMAT(I1,79A1)
      IF(CARD(1).EQ.0) GO TO 42
      CALL MTRPUT(LSWOP,CARD,2,1+CARD(1))
      GO TO 43
   42 CONTINUE
      READ(5,3000) KORD
3000  FORMAT(2I1,78A1)
      IF(KORD(1).EQ.0)GOTO3033
      CALL MTRPUT(LSWIP,KORD,3,2+10*KORD(1)+KORD(2))
      GOTO42
3033  CONTINUE
C  INITIALISATION
      NJOB=0
      KD=0
      KCONT=0
      KC=0
      KAR=0
      KDD=0
      MARK=0
      NEXCT=0
      NUMWRD=120
      NEXC=0
      NEXSTP=0
      L=1
      KE=0
      KBC=0
      NERR=0
      LEVL0=0
      LEVL1=0
      LEVL2=0
      LEVL3=0
      JONLST=0
      KDOLST=0
C  SET THE STATEMENT NUMBER FOR END OF FILE MARK
30    ASSIGN 20 TO IEOF
      NTAPE=18
      WRITE(6,2000)
2000  FORMAT(1X,7HJOB NO.,2X,10HTOTAL CORE,2X,16HCOMPILATION TIME,2X,
     114HEXECUTION TIME,2X,17HTOTAL TIME IN SEC)
      DO10I=1,5000
C  INITIALISE THE BUFFERS
```

```
        B00805        10016034                                          DATE 09/22/70
        MAIN       -  EFN    SOURCE STATEMENT   -  IFN(S)   -

      DO 61 IJN=1,NUMWRD
1     BUFF(IJN)=RCARD
 CALL VARRED ROUTINE WHICH OPENS THE PUFFT OUTPUT TAPE
      CALL VARRED(NTAPE,BUFF,NUMWRD,ILOF)
 CHECK UP FOR 1   $JOB FROM THE BUFFERS
      IF(AND(BUFF(L  ),SMASI1).NE.RMASJ1)GOTO1011
      IF(AND(BUFF(L+1),SMASI2).NE.RMASJ2)GOTO1011
 IF IT IS JOB CARD THEN TEST FOR  ENDRUN CARD
      IF(AND(BUFF(L+3),AMASK1).NE.AMASKJ)GOTO1014
      IF(AND(BUFF(L+4),BMASK1).NE.BMASKJ)GOTO1014
      GOTO 20
014    CONTINUE
 COUNT NUMBER OF JOBS
      NJOB=NJOB+1
 COUNT NUMBER OF SYSTEM CARDS
      KD=KD+1
 SET A FLAG
      NN=0
      GOTO 10
011   CONTINUE
 LOOK FOR ERROR STATEMENTS.IF  THE FIRST BUFFER CONTAINS THE MARK
 *** ** THEN EITHER IT MUST BE AN ERROR OR AN EXECUTION MESSAGE
      IF(AND(BUFF(L),RMASKJ).NE.RMASKI)GOTO1021
 COUNT NUMBER OF ERROR AND EXECUTION MESSAGES
      NERR=NERR+1
 SET A FLAG
      NN=2
      CALL GET(BUFF(L),4,KBUF)
 CONVERT BCD CHARACTER TO AN INTEGER NUMBER
      INT6=BCDINT(KBUF,1)
      INT6=INT6+1
 COUNT THE VARIOUS LEVEL OF ERROR MESSAGES
      GOTO(71,72,73,74),INT6
71    LEVL0=LEVL0+1
      GOTO75
2     LEVL1=LEVL1+1
      GOTO75
3     LEVL2=LEVL2+1
      GOTO75
4     LEVL3=LEVL3+1
5     CONTINUE
 TEST FOR THE 'EXECUTION' FROM THE BUFFERS TO CLASSIFY THE VARIOUS
 EXECUTION MESSAGES SUCH AS EXECUTION TERMINATED AND TERMINATED BY
 STOP AND COUNT THEM
      IF(AND(BUFF(L+2),SMASI3).NE.RMASJ3)GOTO1012
      IF(AND(BUFF(L+3),SMASI4).NE.RMASJ4)GOTO1012
 SET A FLAG
      NN=1
      IF(AND(BUFF(L+3),SMASI5).NE.RMASJ5)GOTO11
      IF(AND(BUFF(L+4),SMASI6).NE.RMASJ6)GOTO11
      IF(AND(BUFF(L+5),SMASI9).NE.RMASJ9)GOTO12
      IF(AND(BUFF(L+6),SMASK1).NE.RMASL1)GOTO12
      NEXSTP=NEXSTP+1
      GOTO 10
2     NEXCT=NEXCT+1
      GOTO 10
```

```
1     NEXC=NEXC+1
      GOTO 10
1021  CONTINUE
C  FOR 'OCOMPILATION' TO FIND THE TIMES FOR COMPILATION AND EXECUTION,
C  THE NUMBER OF MEMORY WORDS THE JOB HAS OCCUPIED
      IF(AND(BUFF(L   ),SMAS17).NE.RMASJ7)GOTO1031
      IF(AND(BUFF(L+1),SMAS18).NE.RMASJ8)GOTO1031
      INT1=BCDINT(BUFF(L+11),4)
      INT2=BCDINT(BUFF(L+13),6)
      ICORE=INT1+INT2
      INT6=BCDINT(BUFF(L+2),6)
      AI=INT6
      T=AI*1.E-2
      INT3=BCDINT(BUFF(L+3 ),2)
      AI=INT3
      TCOM=T+AI*1.E-4
      INT4=BCDINT(BUFF(L+7 ),6)
      INT5=BCDINT(BUFF(L+8 ),3)
      AJ=INT4
      AI=INT5
      TEXC=AJ*1.E-1+AI*1.E-4
      T=60.0*(TCOM+TEXC)
      WRITE(6,18)NJOB,ICORE,TCOM,TEXC,T
18    FORMAT(4X,I2,7X,I5,3(7X,F10.5))
      GOTO 10
1031  CONTINUE
C  LOOK FOR THE COMMENT STATEMENTS AND COUNT THEM
      IF(NN.NE.0)GOTO10
      CALL GET(BUFF(L+1 ),1,KBUF)
      IF(KBUF.NE.NC) GOTO1044
      KC=KC+1
      GOTO 10
1044  CONTINUE
C  THIS PART OF PROGRAM TESTS FOR THE $ MARK TO KNOW THE NUMBER OF
C  SYSTEM CARDS ARE THERE.ALSO TEST FOR '$UNLIST'
      IF(KBUF.NE.ND) GOTO1041
      IF(AND(BUFF(L+1),CMASKI).NE.DMASKJ)GOTO1441
      JUNLST=JUNLST+1
      KDULST=KDULST+3
      GOTO10
1441  CONTINUE
      KD=KD+1
      GOTO 10
1041  CONTINUE
C  THE FOLLOWING WILL TEST FOR THE FORTRAN STATEMENT KEEPING IN MIND THE
C  RECORD LENGTH OF A FORTRAN STATEMENT TAKES 16 OR 17 WORDS OF STORE
      IF(NUMWRD.EQ.16 .OR. NUMWRD.EQ.17)GOTO1059
      GOTO10
1059  CONTINUE
      IF(BUFF(L+15).NE.RCARD)GOTO1058
      GOTO10
1058  CONTINUE
      CALL GET(BUFF(L+15),1,KBUF)
      INT7=BCDINT(KBUF,1)
      IF(INT7.EQ.5 .OR. INT7.EQ.6) GOTO1051
1051  CONTINUE
```

```
           B00805      10016034                                        DATE 09/22/70
           MAIN        -  EFN    SOURCE STATEMENT  -  IFN(S)  -

C  TEST AND COUNT BLANK CARDS
      DO777J=3,15
      IF(BUFF(J).NE.RCARD)GOTO1061
777   CONTINUE
      KBC=KBC+1
      GOTO10
1061  CONTINUE
      IF(BUFF(L+14).NE.RCARD)GOTO10
1012  CONTINUE
C  EACH CHARACTER OF BUFFER IS PACKED IN  LINE(BUFF CONTAINS SIX
C  CHARACTERS IN ONE WORD AND THEY ARE PACKED IN SIX WORDS  REPRESENTED
C  BY AN ARRAY LINE)
      N=0
      DO 121 IJ=2,14
      DO 121 II=1,6
      N=N+1
      CALL GET(BUFF(IJ),II,LINE(N))
 121  CONTINUE
      IF(NN.EQ.2)GOTO1013
      IF(LINE(1).NE.NC) GO TO 45
      GOTO 10
  45  CONTINUE
      KE=KE+1
C    TEST FOR EQUALS
      DO 55 K55=1,72
      IF(LINE(K55).EQ.NEQUAL) GO TO 34
      GO TO 55
  34  CONTINUE
      MARK=1
      GO TO 57
  55  CONTINUE
C   TEST FOR CONTINUATION
      IF(LINE(6).NE.NBLANK) GO TO 56
      GO TO 57
  56  CONTINUE
      KCONT=KCONT+1
      GOTO 10
  57  CONTINUE
013   CONTINUE
C  SQUEEZE UP TO COLUMN 7
      I88=7
      K88=7
C    REMOVE BLANKS BETWEEN CHARACTERS IN A STATEMENT
  85  IF(LINE(I88).NE.NBLANK) GO TO 84
      I88=I88+1
      GO TO 85
  84  CONTINUE
      LINE(K88)=LINE(I88)
      K88=K88+1
      I88=I88+1
      IF(I88.GT.72) GO TO 88
      GO TO 85
  88  CONTINUE
      IF(NN.EQ.2)GOTO1015
      IF(MARK.NE.1) GO TO 89
C  TEST FOR 'DO' STATEMENTS,IF THE STATEMENT IS A DO GOTO58
```

```
      IF(LINE(7).EQ.NDD.AND.LINE(8).EQ.NOO) GO TO 58
C   OTHERWISE TEST FOR READ, WRITE, FORMAT, AND DATA STATEMENTS
      DO 13 MN1=1,4
      MB1=MN1+6
      IF(LINE(MB1).NE.IREAD (MN1))GOTO600
13    CONTINUE
      GOTO89
600   CONTINUE
      DO 14 MN2=1,5
      MB2=MN2+6
      IF(LINE(MB2).NE.IWRITE(MN2))GOTO601
14    CONTINUE
      GOTO89
601   CONTINUE
      DO 15 MN3=1,6
      MB3=MN3+6
      IF(LINE(MB3).NE.IFORMT(MN3))GOTO602
15    CONTINUE
      GOTO89
602   CONTINUE
      DO 16 MN4=1,4
      MB4=MN4+6
      IF(LINE(MB4).NE.IDAT(MN4))GOTO1899
16    CONTINUE
      GOTO89
1899  CONTINUE
C   COUNT THE ARITHMETIC STATEMENTS
      KAR=KAR+1
      MARK=0
      GOTO 10
   58 KDO=KDO+1
      MARK=0
      GOTO 10
   89 CONTINUE
      IDOLL=K88-1
      IF(LINE(IDOLL).EQ.ND)LINE(IDOLL)=NBLANK
1015  CONTINUE
      IF(NN.EQ.2)GOTO1442
      LDEPTH=32
C    CALL TREE SWAP ROUTINE TO CLASSIFY VARIOUS STATEMENTS
      CALL TRSWOP(LSWOP,LINE,7,LDEPTH)
      GOTO10
1442  CONTINUE
      LDEPTH=45
      CALL TRSWOP(LSWIP,LINE,7,LDEPTH)
10    CONTINUE
20    CONTINUE
      RF=NJOB
      RD=NEXC
      RD=100.*RD/RF
      WRITE(6,26)NJOB,NEXC,RD
26    FORMAT(1X,13HNO OF JOBS = ,20X,I3/1X,33HNO OF JOBS WENT INTO EXECU
     1TION = ,I4,5X,F6.2,22H PERCENT OF TOTAL JOBS)
      RD=NEXCT
      RD=100.*RD/RF
      WRITE(6,91)NEXCT,RD
      B00805      10016034                                                      DATE 09/22/70
      MAIN        -  EFN   SOURCE STATEMENT  -  IFN(S)  -

1       FORMAT(1X,24HNO OF JOBS TERMINATED = ,9X,I4,5X,
       1F6.2,22H PERCENT OF TOTAL JOBS)
        RD=NEXSTP
        RD=100.*RD/RF
        WRITE(6,92)NEXSTP,RD
2       FORMAT(1X,33HNO OF JOBS TERMINATED BY STOP =  ,I4,5X,
       1F6.2,22H PERCENT OF TOTAL JOBS)
        RF=NERR
        RD=LEVL0
        RD=100.*RD/RF
        WRITE(6,93)NERR,LEVL0,RD
3       FORMAT(1X,28HTOTAL NO OF ERROR MESSAGES = ,I4/1X,23HERRORS OF LEVE
       1L ZERO = ,10X,I4,5X,
       1F6.2,23HPERCENT OF TOTAL ERRORS)
        RD=LEVL1
        RD=100.*RD/RF
        WRITE(6,94)LEVL1,RD
4       FORMAT(1X,23HERRORS OF LEVEL  ONE = ,10X,I4,5X,
       1F6.2,23HPERCENT OF TOTAL ERRORS)
        RD=LEVL2
        RD=100.*RD/RF
        WRITE(6,95)LEVL2,RD
5       FORMAT(1X,23HERRORS OF LEVEL  TWO = ,10X,I4,5X,
       1F6.2,23HPERCENT OF TOTAL ERRORS)
        RD=LEVL3
        RD=100.*RD/RF
        WRITE(6,96)LEVL3,RD
6       FORMAT(1X,23HERRORS OF LEVEL THREE= ,10X,I4,5X,
       1F6.2,23HPERCENT OF TOTAL ERRORS)
        NEX=NEXC+NEXCT+NEXSTP
        RD=NEX
        RD=100.*RD/RF
        WRITE(6,97)NEX,RD
7       FORMAT(1X,38HVARIOUS TYPES OF EXECUTION MESSAGES = ,I4,5X,
       1F6.2,23HPERCENT OF TOTAL ERRORS)
        WRITE(6,778)
        WRITE(6,3333)JUNLST,KDULST
333     FORMAT(1X,34HNO OF JOBS WITH UNLIST SYSTEM CARD    I4/
       11X,40HSYSTEM CARDS ASSOCIATED WITH UNLIST JOBS  I4)
        K=KD+KBC+KC+KE
        WRITE(6,100) K
100     FORMAT(20H CARDS ANALYSED            I6)
        RK=K
        RD=KD
        RD=100.*RD/RK
        WRITE(6,101)KD,RD
 101 FORMAT(20H SYSTEMS CARDS             I6,4H     F6.2,19H PER CENT OF TOTAL   )
       1TAL   )
        KF=KE+KC
        RF=KF
        RF=100.*RF/RK
        WRITE(6,103) KF,RF
 103 FORMAT(20H FORTRAN                   I6,4H     F6.2)
        WRITE(6,104) KCONT
 104  FORMAT(20HOCONTINUATION CARDS   I6)
        WRITE(6,770)KBC
```

```
770   FORMAT(12H0BLANK CARDS  I6)
 105  FORMAT(20H0FORTRAN STATEMENTS   I6)
      KT=KF-KCONT
      WRITE(6,105)KT
      KS=KT-KC
      WRITE(6,106) KC
 106  FORMAT(20H0COMMENT CARDS        I6)
      WRITE(6,107) KS
 107  FORMAT(20H0PROGRAM STATEMENTS   I6)
      RS=KS
      RAR=KAR
      RAR=100.*RAR/RS
      WRITE(6,108) KAR,RAR
 108  FORMAT(20H0ARITHMETIC          I6,4H      F6.2,10H PER CENT  )
      RDO=KDO
      RDO=100.*RDO/RS
      WRITE(6,109) KDO,RDO
 109  FORMAT(20H DO STATEMENTS       I6,4H      F6.2,10H PER CENT  )
      WRITE(6,110)
 110  FORMAT(52H0PER CENT FREQUENCY OF OTHER STATEMENTS AS FOLLOWS     )
    CALL TREE PRINT ROUTINE TO PRINT THE VARIOUS OTHER STATEMENTS
      CALL TPRINT(LSWOP,32,KS)
      WRITE(6,778)
      WRITE(6,3334)
334   FORMAT(54H0PER CENT FREQUENCY OF ERROR MESSAGES ARE AS FOLLOWS   )
      NER=NERR-NEXC-NEXCT-NEXSTP
      CALL TPRINT(LSWIP,42,NER)
      WRITE(6,778)
78    FORMAT(1H1)
    CLEAR THE BUFFERS AGAIN
      DO62MASD=1,NUMWRD
2     BUFF(MASD)=RCARD
    CALL VARRED ROUTINUE
      CALL VARRED(NTAPE,BUFF,NUMWRD,IEOF)
      L=1
    TEST THE END OF TAPE, IF NOT GO AND STARTTHE PROCESS
      IF(AND(BUFF(L),SMASKI).NE.SMASKJ)GOTO030
      WRITE(6,1000)
000   FORMAT(24H END OF TAPR ENCOUNTERED)
      STOP
      END
```

```
IBFTC 9MTRPT  DECK
MTRPT      -  EFN   SOURCE STATEMENT  -  IFN(S)  -

  SUBROUTINUE MTRPUT CREATES TREE STRUCTURE IN THE FORM OF CELLS.
  CHARACTER IN THE BRANCH AND IEND IS THE LAST CHARACTER OF THE BRANCH

     SUBROUTINE MTRPUT(LHEAD,CARD,IBEGIN,IEND)
     INTEGER CARD(80)
  POINT TO LAST AS THE TOP OF TREE
     LAST=LHEAD
     DO66K=IBEGIN,IEND
  TEST THE ID FIELD OF TOP CELL. IF IT IS ZERO THEN IT MEANS IT IS NOT
  FILLED WITH THE INFORMATION. OTHERWISE EITHER IT IS FILLED IN CASE
  IT IS 1 OR END OF BRANCH IN CASE IT IS 2.
     IF(ID(INHALT(LAST)).EQ.0) GO TO 77
  TEST WHETHER THE PRESENT CELL CONTAINS THE INFORMATION WHICH MATCHES
  WITH THE CHARACTER UNDER CONSIDERATION.
  33 IF(INHALT(LAST+1).EQ.CARD(K)) GO TO 22
     IF(LNKL(INHALT(LAST)).EQ.0) GO TO 44
     LAST=LNKL(INHALT(LAST))
     GOTO 33
  CREATE NEW CELL TO LEFT (DOWN)
  44 CALL NUCELL(LINK)
     CALL SETIND(1,LINK,-1,LAST)
     LAST=LINK
     CALL SETIND(1,0,0,LAST)
     CALL STRIND(CARD(K),LAST+1)
  55 CALL NUCELL(LINK)
     CALL SETIND(1,-1,LINK,LAST)
     LAST=LINK
     CALL SETIND(0,0,0,LAST)
     GOTO 66
  22 IF(LNKR(INHALT(LAST)).EQ.0) GO TO 55
     LAST=LNKR(INHALT(LAST))
     GOTO 66-
  CREATE NEW CELL TO RIGHT
  77 CALL STRIND(CARD(K),LAST+1)
     CALL SETIND(1,-1,-1,LAST)
     GOTO 55
  66 CONTINUE
   SET END MARK AND FILL THE DATA WORD WITH 0.
     CALL SETIND(2,0,0,LAST)
     CALL STRIND(0,LAST+1)
     RETURN
     END
```

```
IBFTC 9TRSWP   DECK

TRSWP      -  EFN   SOURCE STATEMENT  -  IFN(S)  -

  THIS SUBROUTINE HELPS TO COMPARE THE VARIOUS FORTRAN AND OTHER
   STATEMENTS OF THE OUTPUT TAPE OF PUFFT.
  IT ALSO BUILDS UP TREE. WHILE COMPARING AND ACCOUNTING THE VARIOUS
   STATEMENTS IT CONVENIENTLY SWAPS THE BRANCHES OF THE TREE.

      SUBROUTINE TRSWOP(LHEAD,CARD,IBEGIN,IEND)
      INTEGER CARD(80)
  SET AMARKER TO ZERO
      MARK=0
  INITIALISE THE LIST HEADER
      JUST=LHEAD
      LAST=LHEAD
      DO66K=IBEGIN,IEND
  COMPARE THE VALUE OF THE ID FIELD OF LAST WITH ZERO.IF IT IS ZERO
   THEN FILL UP THE DATA WORD WITH CHARACTER AND CREATE A NEW CELL
    TO RIGHT.
      IF(ID(INHALT(LAST)).EQ.0) GO TO 77
  IF IT IS NOT ZERO THEN COMPARE WITH THE CONTENTS OF THE DATA WORD
   OF LAST WITH THE CHARACTER FROM THE OUTPUT TAPE.IF THEY MATCH GOTO 22
  33 IF(INHALT(LAST+1).EQ.CARD(K)) GO TO 22
  IF THEY DO NOT MATCH CHECK UP WHETHER THE VALUE OF THE LNKL FIELD OF
   LAST IS ZERO. IF IT IS ZERO GOTO 44 AND CREATE A NEW CELL TO LEFT.
      IF(LNKL(INHALT(LAST)).EQ.0) GO TO 44
  IF IT IS NOT ZERO IN WHICH CASE THERE IS ALREADY ALINK TO LEFT AND
   HENCE SET THE MARK AS 1
      MARK=1
  SET JUST AS LAST AND LAST TO THE ADDRESS OF THE LNKL OF LAST ,
   THEN GOTO 33 AND COMPARE THE DATA WORD WITH THE CHARACTER
      JUST=LAST
      LAST=LNKL(INHALT(LAST))
      GOTO 33
C  CREATE NEW CELL TO LEFT(DOWN)
  44 CALL NUCELL(LINK)
C  SE 1,LINK INTO ID AND LNKL FIELDS OF LAST ANDLNKR FIELD UNALTERED
      CALL SETIND(1,LINK,-1,LAST)
      JUST=LAST
      LAST=LINK
      CALL SETIND(1,0,0,LAST)
C  STORE THE CHARACTER INTO THE DATA WORD OF THE CELL LAST
      CALL STRIND(CARD(K),LAST+1)
C  RAISE A NEW CELL
  55 CALL NUCELL(LINK)
      CALL SETIND(1,-1,LINK,LAST)
      JUST=LAST
      LAST=LINK
      CALL SETIND(0,0,0,LAST)
      GOTO 66
   22 CONTINUE
C  THIS PART OF THE PROGRAM DOES THE SWAPPING OF THE BRANCH OF TREE
      IF(MARK.EQ.0) GO TO 23
      ITEMP1=LNKR(INHALT(JUST))
      ITEMP2=INHALT(JUST+1)
      CALL SETIND(1,-1,LNKR(INHALT(LAST)),JUST)
      CALL STRIND(INHALT(LAST+1),JUST+1)
      CALL SETIND(1,-1,ITEMP1,LAST)
      CALL STRIND(ITEMP2,LAST+1)
      LAST=JUST
      MARK=0
   23 CONTINUE
      IF(LNKR(INHALT(LAST)).EQ.0) GO TO 55
      JUST=LAST
      LAST=LNKR(INHALT(LAST))
      IF(ID(INHALT(LAST)).EQ.2) GO TO 67
      GOTO 66
C  CREATE A NEW CELL TO RIGHT
   77 CALL STRIND(CARD(K),LAST+1)
      CALL SETIND(1,-1,-1,LAST)
      GOTO 55
   66 CONTINUE
C  SET AN END MARK FOR LAST AND THE DATA WORD TO ZERO
      IF(ID(INHALT(LAST)).EQ.2)GOTO67
      CALL SETIND(2,0,0,LAST)
      CALL STRIND(0,LAST+1)
   67 CONTINUE
C  THIS TRACKS THE COUNT OF THE VARIOUS SIMILAR STATEMENTS SUCH AS READ...
      N=INHALT(LAST+1)+1
      II=INHALT(LAST+1)
      CALL STRIND(N,LAST+1)
      RETURN
      END
```

```
$IBFTC 9TPRIN  DECK
9TPRIN      -  EFN   SOURCE STATEMENT  -  IFN(S)  -
C
C
C  THIS ROUTINE PRINTS OUT THE TREEWHICH HAS ALREADY GEEN CREATED BY
C   MTRPUT AND TRSWOP
C
C
      SUBROUTINE TPRINT(LHEAD,LDEPTH,NSTATS)
      INTEGER POPTOP
      INTEGER BLANK,LINE(120)
      DATA BLANK /6H      /
      NIL=0
      K=1
C  LIST HEADER
      LAST = LHEAD
C  CREATE AN EMPTY LIST WHICH LEAVES THE NAME LINK BOTH AS ITS VALUE
C   AND IN THE CELL LINK
      CALL LIST(LINK)
C  PUSH THE DATUM NIL DOWN ON TOP OF THE LIST LINK
      CALL NEWTOP(NIL,LINK)
C  INITIALISATION OF ARRAY LINE
   77 DO 22 I=K,LDEPTH
   22 LINE(I)=BLANK
C  STORE THE CONTENTS OF  DATA WORD OF THE LAST
   33 LINE(K)=INHALT(LAST+1)
C  INCREASE THE VALUE OF K
      K=K+1
C  TEST FOR THE END OF THE BRANCH. IF THE CONTENTS OF THE LNKR FIELD
C   IS ZERO THEN IT IS THE END OF BRANCH AND HENCE THE STATEMENT AND ITS
C    FREQUENCY IS NOTED AND VALUE IS PRINTED OUT.
      IF(LNKR(INHALT(LAST)).EQ.0) GO TO 44
C  IF IT IS NOT EQUAL TO ZERO THEN PUSH THE DATUM LAST DOWN ON TOP OF
C   LIST  LINK
      CALL NEWTOP(LAST ,LINK)
C  SET THE LAST POINTER TO THE LNKR FIELD OF LAST AND GOTO 33
      LAST=LNKR(INHALT(LAST))
      GO TO 33
   44 CONTINUE
      N= INHALT(LAST+1)
      L=K-2
      RN=N
      R=NSTATS
      R=100.*RN/R
      WRITE(6,66) R,N,(LINE(J),J=1,L)
   66 FORMAT(1H ,F7.2,3X,I6,3X,100A1)
   45 CONTINUE
C  CHECK UP WHETHER THE LNKL FIELD OF LAST IS ZERO OR NOT. IF IT IS ZERO
C    THEN LINK IS 'POPPED OFF' AS LAST WHICH MEANS IT IS RETURNED TO
C   LAVS AND THE DATUM CONTAINED THERE IN DELIVERED AS THE VALUE OF THE
```

```
C     FUNCTION. REDUCE K TILL THE WHOLE TREE IS PRINTED OUT.
      IF(LNKL(INHALT(LAST)).EQ.0) GO TO 55
      LAST=LNKL(INHALT(LAST))
      K=K-1
      GO TO 77
   55 LAST=POPTOP(LINK)
       K=K-1
      IF(K.GT.1) GO TO 45
      RETURN
       END
$IBMAP BCDINT  NOREF,M90                                                ICI00010
```

```
                                                                  BCDI0001
CDINT SAVE                     ENTER,SAVE LINKAGE                              ICI00020
       STZ     DUMMY           CLEAR TEMPORARY STORE                           ICI00030
       LDQ*    3,4             LOAD BCD VARIABLE NUMBER                        ICI00040
       CLA*    4,4             LOAD CHARACTER COUNT                            ICI00050
       PAX     ,4              INTO REGISTER                                   ICI00060
AC     ZAC                                                                     ICI00070
       LGL     6               GET A CHARACTER                                 ICI00080
       LAS     ABLANK          COMPARE BLANK                                   ICI00090
       TRA     INVCHR          HIGHER-INVALID TREAT ZERO                       ICI00100
       TXI     ZAC,4,-1        EQUAL, IGNORE. GET A CHARACTER                  ICI00110
       LAS     TEN             COMPARE TEN                                     ICI00120
       TRA     INVCHR          HIGHER -INVALID TREAT ZERO                      ICI00130
NVCHR  ZAC                     TREAT AS ZERO                                   ICI00140
       ADD     DUMMY           VALID ADD TO TOTAL                              ICI00150

       TXL     BCDINT+1,4,1     TRY LAST CHARACTER                             ICI00160
       STQ     NUMBER          SAVE REST OF BCD NUMBER                         ICI00170
       XCA                                                                     ICI00180
       MPY     TEN             MULTIPLY INTEGER TOTAL BY TEN                   ICI00190
       STQ     DUMMY           SAVE IN NEW TOTAL                               ICI00200
       LDQ     NUMBER          RESTORE REST BCD NUMBER                         ICI00210
       TXI     ZAC,4,-1        GET NEXT CHARACTER                              ICI00220
BLANK  BCI     1,00000                                                         ICI00230
EN     DEC     10                                                              ICI00240
UMBER  PZE     **                                                              ICI00250
UMMY   PZE     **                                                              ICI00260
       *LDIR

       END                                                                     ICI00270
       ENTRY    VARRED

VARRED SXA      SYSLOC,4  SAVE LINKAGE
       SXA      LDIR,4    SAVE REGISTERS
       SXA      XR4,4
       STI      INDICS
       SXA      XR2,2
       CLA      3,4       ADDRESS LOG.TAPE UNIT PARAMETER
       STA      NTAPE
       CLA      4,4       ADDRESS USER BUFFER PARAMETER
       STA      BUFFER
       NZT*     5,4       TEST FOR FORCING E.O.R
       ZSD      RECADD
       CLA*     6,4       END OF FILE EXIT.
       STA      EOFEXT
       CALL     .FVIO.(NTAPE,FILE)  CONVERT LOG.TAPE UNIT TO FCB ADDRESS




       ORG      *-2
NTAPE  PZE      **
       PZE      FILE
       LAC      FILE,4    SET UP FCB ADDRESS PARAMETERS
       SCA      UNIT1,4
       SCA      UNIT2,4
       LDI      1,4       TEST FILE FOR OPEN
       LFT      040000
       TRA      NEWREC
       TSX      .OPEN,4   OPEN IF NOT
UNIT1  MON      **
NEWREC CLA      FILE      COMPARE PRESENT FILE
       CAS      LSTFIL    WITH LAST FILE
       TRA      *+2
       TRA      PARBUF    EQUAL.TRANSFER TEST EMPTY INTERVAL BUF.
       STO      LSTFIL    SAVE NEW 'LAST FILE'
READ   TSX      .READ,4   READ NEW PHYSICAL RECORD
UNIT2  PZE      **,,*+3
EOFEXT PZE      **,,READ

RECADD IORTN    **,,**    ADDRESS OF INT.BUF.+WORDS IN BUFF.
       LXD      RECADD,4
       SXD      BUFEND,4  SAVE COUNT FOR END OF BUFF.TEST.
       LXA      RECADD,4
BUFEND TXI      *+1,4,**
       SXA      BUFADD,4
PARBUF LXD      RECADD,4  TEST FOR EMPTY BUFFER
       AXT      0,2
NXTBLK TXL      READ,4,0  READ NEW PHYSICAL RECORD IF EMPTY
```

```
BUFADD CAL      **,4       EXTRACT WORD FROM BUFFER
       ANA      RECMSK
       ERA      RECMRK     TEST FOR RECORD MARK (072)
       LDQ*     BUFADD
BUFFER STQ      **,2       STORE IN USER BUFFER
       TXI      *+1,2,-1   INCREMENT WORD COUNT
       TXI      *+1,4,-1
       TZE      ENDBLK     BRANCH IF RECORD MARK
       TXH      BUFADD,4,0   GET NEXT WORD IF NOT END PHYSICAL RECORD
ENDBLK PCA      ,2         SAVE WORD COUNT OF LOGICAL RECORD

       SXD      RECADD,4   SAVE WORD COUNT OWORDS LEFT IN INT.BUFF.
XR2    AXT      **,2       RESTORE REGISTERS
XR4    AXT      **,4
       LDI      INDICS
       STO*     5,4        RETURN LOGICAL RECORD SIZE
       TRA      1,4        RETURN TO USER
FILE   PZE      **         PRESENT FILE
LSTFIL PZE      **         LAST FILE.
INDICS PZE      **
RECMSK OCT      77         MASK FOR RECORD MARK
RECMRK OCT      72          RECORD MARK.
LDIR   LDIR

       END
```

```
                                                            REMJ0001
UN01. ENTRY    IN
UN02. ENTRY    IN
UN03. ENTRY    IN
UN04. ENTRY    IN
UN07. ENTRY    IN
UN08. ENTRY    IN
UN09. ENTRY    IN
UN10. ENTRY    IN
UN11. ENTRY    IN
UN12. ENTRY    IN
UN13. ENTRY    IN
UN14. ENTRY    IN
UN15. ENTRY    IN
UN16. ENTRY    IN
UN17. ENTRY    IN
UN19. ENTRY    IN
UN20. ENTRY    IN
UN21. ENTRY    IN
UN22. ENTRY    IN
UN23. ENTRY    IN
UN24. ENTRY    IN

N       PZE     **
        END
```

```
B NO.  TOTAL CORE  COMPILATION TIME  EXECUTION TIME  TOTAL TIME IN SEC
  1        5173         0.17080          0.32920          30.00000
  2        2256         0.07280          0.00940           4.93200
  3        1473         0.06170          0.00750           4.15200
  4        1491         0.06280          0.00750           4.21800
  5        4061         0.10250          0.01470           7.03200
  6        1183         0.04830          0.05920           6.45000
  7        5151         0.02500          0.03170           3.40200
  8         437         0.02420          0.00780           1.92000
  9        3208         0.12310          0.00670           7.78800
 OF JOBS =                                 9
 OF JOBS WENT INTO EXECUTION =             7    77.78 PERCENT OF TOTAL JOBS
 OF JOBS TERMINATED =                      4    44.44 PERCENT OF TOTAL JOBS
 OF JOBS TERMINATED BY STOP =              5    55.56 PERCENT OF TOTAL JOBS
TAL NO OF ERROR MESSAGES =  32
RORS OF LEVEL ZERO =                      17    53.13PERCENT OF TOTAL ERRORS
RRORS OF LEVEL  ONE =                      0     0.00PERCENT OF TOTAL ERRORS
RRORS OF LEVEL  TWO =                      4    12.50PERCENT OF TOTAL ERRORS
RRORS OF LEVEL THREE=                     11    34.38PERCENT OF TOTAL ERRORS
ARIOUS TYPES OF EXECUTION MESSAGES =   16        50.00PERCENT OF TOTAL ERRORS
```

```
J OF JOBS WITH UNLIST SYSTEM CARD    1
YSTEM CARDS ASSOCIATED WITH UNLIST JOBS    3
ARDS ANALYSED           867
YSTEMS CARDS             25      2.88 PER CENT OF TOTAL
ORTRAN                  842     97.12

ONTINUATION CARDS        78

LANK CARDS      0

ORTRAN STATEMENTS       764

OMMENT CARDS            188

ROGRAM STATEMENTS       576

RITHMETIC               389     67.53 PER CENT
O STATEMENTS             26      4.51 PER CENT

ER CENT FREQUENCY OF OTHER STATEMENTS AS FOLLOWS
  5.56        32   WRITE(
  6.60        38   FORMAT
  0.00         0   FUNCTION
  2.08        12   IF(
  0.00         0   INTEGER
  1.22         7   CALL
  0.87         5   COMMON
  0.00         0   COMPLEX
  4.69        27   CONTINUE
  1.22         7   END
  0.00         0   ENTRY
  0.00         0   EQUIVALEN
  0.00         0   EXTERNAL
  0.52         3   REAL
  2.08        12   READ(5,
  0.00         0   READ(
  0.52         3   RETURN
  0.69         4   SUBROUTIN
  0.52         3   STOP
  0.17         1   DATA
  0.69         4   DIMENSION
  0.00         0   DOUBLE
  0.52         3   GOTO
  0.00         0   BLOCKDATA
  0.00         0   LOGICAL
  0.00         0   NAMELIST
  0.00         0   PRINT
```

```
ER CENT FREQUENCY OF ERROR MESSAGES ARE AS FOLLOWS
 12.50        2   ILLEGALPUNCTUATION
  6.25        1   ILLEGALUSEOFBCDCONSTANT
  0.00        0   ILLEGALUSEOFBCDCHARACTER
  0.00        0   ILLEGALUSEOFLOGICALUNIT00006
  0.00        0   ILLEGALUSEOFLOGICALUNIT00005
  0.00        0   ILLEGALUSEOFFUNCTIONNAME
  0.00        0   ILLEGALUSEOFPERIOD
  6.25        1   ILLEGALSUBSCRIPTAT
  0.00        0   ILLEGALSTATEMENTNUMBER
  0.00        0   ILLEGALSEQUENCINGOFSTATEMENT
 12.50        2   ILLEGALCARDBELOWIGNORED
  0.00        0   ILLEGALCOMMAIGNORED
 12.50        2   ILLEGALARRAYNAME
  0.00        0   ILLEGALREALCONSTANT
  0.00        0   ILLEGALVARIABLENAME
  0.00        0   ILLEGALOPERATOR
  0.00        0   ILLEGALINTEGERVARIABLEORCONSTANT
  0.00        0   ILLEGALENDOFSTATEMENT
  0.00        0   ILLEGALNUMBEROFSUBSCRIPTFORMAT
  0.00        0   INCONSISTENTEQUIVALENCEOFVARIABLE
  0.00        0   INDEXANDLISTLENGTHINCONSISTENT
  0.00        0   STATEMENTNUMBERREQUIRED
  0.00        0   STATEMENTTYPEERROR
  0.00        0   SYSTEMERRORATOCTAL
  0.00        0   PARENTHESISERROR
  0.00        0   PARITYERRORONUNIT00005
  0.00        0   PROGRAMORARRAYTOOLONG
  0.00        0   PUNCTUATIONERROR
 12.50        2   VARIABLECALUSEDASFUNCTION---EXECUTIONDE
  0.00        0   VARIABLEORCONSTANTTOOLONG
  6.25        1   MAINPROGRAMDEFINEDTWICE
  0.00        0   MAINPROGRAMMISSING
  0.00        0   MISSINGORILLEGALJOBCARD
  0.00        0   MIXEDTYPEOPERATION
  0.00        0   OVERFLOWAT
  6.25        1   UNDERFLOWAT
  6.25        1   ENDCARDMISSING
  0.00        0   EOFREADONUNIT00005
  0.00        0   ERRORINFORMATAT
  0.00        0   NESTEDDO'SOVERLAP
  0.00        0   CHECKUSEOFARRAY
  0.00        0   CONSTANTREQUIREDFORDATA
  0.00        0   WRITEMISSPELLED
  6.25        1   ARRAYCALNOTDEFINED---EXECUTIONDELETED

  6.25        1   ARRAYCNOTDEFINED---EXECUTIONDELETED

  6.25        1   ATOCTAL00000,CUSED---UNDEFINEDSUBRCUTIN
  0.00        0   FORMATMISSPELLED
  0.00        0   BCDOUTPUTRECORDTOOLONG
```