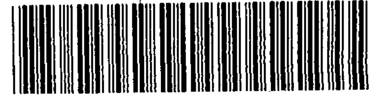

LINEAR LIBRARY
C01 0068 1441

UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

Analytical Differentiation by Computer Using a Symmetrical List Processor

by

Barry Martin Lichtman, B.Sc. (Hons.)

A Thesis Prepared under the Supervision of

Professor D. G. Parkyn

in Partial Fulfilment of the Requirements

for the Degree of Master of Science in Computer Science.

University of Cape Town

Cape Town

September, 1972

The copyright of this thesis is held by the University of Cape Town.
Reproduction of the whole or any part may be made for study purposes only, and not for publication.

The copyright of this thesis vests in the author. No quotation from it or information derived from it is to be published without full acknowledgement of the source. The thesis is to be used for private study or non-commercial research purposes only.

Published by the University of Cape Town (UCT) in terms of the non-exclusive license granted to UCT by the author.

University of Cape Town

ACKNOWLEDGEMENTS

My first thanks are to my supervisor, Professor D. G. Parkyn, Head of the Department of Computer Science at the University of Cape Town, for his advice and for the time he sacrificed in reading the first draft of this thesis while preparing to go overseas.

My appreciation is extended to the University Computer Centre whose facilities were so vital to this thesis. My thanks especially to the computer operators whom I harrassed so much while developing my programs.

I should also like to express my appreciation to those of my friends who made suggestions which I was able to use, notably to L. Drew for a suggestion on how to convert an infix-ordered expression to Polish prefix ordering.

The Council for Scientific and Industrial Research are greatly thanked for the finance extended to me by the award of a Post-B. Sc. (Hons.) bursary for the full period of two years during which I worked for my degree.

My thanks are due to Mrs J. Wienburg for the trouble she took over typing the first draft of this thesis.

Finally I should like to thank my parents, whose encouragement when I was despondent was an enormous boost to my morale and without whose financial assistance and interest in my having a higher education, I should never have reached this stage in my studies.

University of Cape Town

CONTENTS

page

List of Illustrations v
List of Tables vi
Abstract vii
Chapter 1 The History and Problems of Symbolic Differentiation by Computer 1
1.0 Introduction 1
1.1 History of Symbolic Manipulation . . 1
1.2 Problems Encountered in Automated Algebraic Manipulation 3
Chapter 2 The Symmetrical List Processor (SLIP) . . 5
2.0 Introduction 5
2.1 SLIP Data Structure 7
2.2 Storage Allocation and Reclamation . . 10
2.3 The SLIP Routines 13
2.3.1 The Primitives 13
2.3.2 The FORTRAN Routines 16
2.3.2.1 Storage Allocation 16
2.3.2.2 Data Manipulation and Testing Routines 17
2.3.2.3 Sequencing Mechanism 19
2.3.2.4 Readers and Advance Routines . 20
2.3.2.5 Description Lists 23
2.3.2.6 Recursion 24
2.3.2.7 Input/Output Routines 27
Chapter 3 DERIV, A Symbolic Differentiation Package . 29
3.0 Introduction 29
3.1 Trees, Polish Notation and Algebraic Expressions 29
3.2 The SLIP Differentiation Package DERIV 32
3.3 The Routines in DERIV 43
3.3.1 Differentiation 43
3.3.2* The Differentiation Function ALGDIF 46
3.3.2.1 Description 46
3.3.2.2 Simplification 47
3.3.3 Input/Output Routines in DERIV . . 48

University of Cape Town

page

3.4 Design Philosophy of DERIV 60
3.5 Testing and Debugging of DERIV . . . 62
3.6 The Capabilities of DERIV 63
3.6.1 Examples and Estimates of Their Efficiency 63
3.6.2 Inefficiencies in Core and Time Utilization 66
Bibliography 69
Appendix A SLIP: FORTRAN and ASSEMBLER Listings . . 72
Appendix B DERIV: FORTRAN Listings 109
Appendix C Flowcharts of the DERIV Routines 140
Appendix D Examples of the Use of DERIV 148
Index of Function Listings 170

University of Cape Town

LIST OF ILLUSTRATIONS

figure		page
1.	Partial Word Designators of an 1108 Word	8
2.	The Layout of U-SLIP Nodes	8
3.	Example of a U-SLIP List Structure	11
4.	An Empty List	11
5.	Diagrammatical Representation of a Tree	30
6.	Representations of Algebraic Expressions by Binary Trees	30
7.	Flowchart: Conversion of Infix to Polish Suffix (Reverse Polish) Ordering	33
8.	Flowchart: Conversion of Infix to Polish Prefix Ordering	34
9.	Flowchart: Conversion of Polish Prefix to Infix Ordering	35
10.	Layout of DERIV Node Types	38
11.	Flowchart: Simplification of the Derivative of "L + R"	49
12.	Flowchart: Simplification of the Derivative of "L - R"	50
13.	Flowchart: Simplification of the Derivative of "L * R"	51
14.	Flowchart: Simplification of the Derivative of "L / R"	52
15.	Flowchart: Simplification of the Derivative of "L ! R"	53
16.	Flowchart: Recognition of Input Variables and Constants	58

University of Cape Town

LIST OF TABLES

table page

1. The Operators Allowed in the DERIV System 40
2. Differentiation Rules as Used in DERIV 41
3. Comparison of Three Different Approaches to Example 3 65
4. Comparison of Two Methods of Programming Example 4 65

University of Cape Town

ABSTRACT

The Symmetrical List Processor SLIP; developed by Professor Joseph Weizenbaum of MIT, was implemented with considerable modifications and additions on the University of Cape Town computer.

A package to perform automated analytical differentiation (DERIV) was developed using SLIP. Basic simplification techniques as well as convenient input and output routines were included. The package was tested extensively and a rough comparison drawn with the abilities of various computer languages and programs which include the same facility as DERIV.

University of Cape Town

CHAPTER 1

THE HISTORY & PROBLEMS OF SYMBOLIC DIFFERENTIATION BY COMPUTER

1.0 Introduction.

A brief history of automated symbolic manipulation with emphasis on differentiation procedures is followed by an exposition of the difficulties commonly encountered in this field, such as running out of core space and the consistent representation of algebraic expressions.

1.1 History of Symbolic Manipulation.

The earliest efforts were by Nolan (8) and Kahrimanian (9), who independently of one another (1953) wrote computer programs for performing differentiation. Because of severe restrictions imposed by the technology of the time, their routines were necessarily primitive.

The programs mentioned above are described in some detail in (10). The algebraic expression to be differentiated has in (9) to be completely parenthesized and encoded in a 3-address code. This transformation begins with the innermost set of parentheses and works outward. The resultant tabular form of the input expression thus has the simpler elementary functions listed first and then, following in sequence, the more complex elementary functions. The differentiation program operates on this tabular form of the expression and generates a similar form to represent the resultant derivative. The final tabular output has to be recombined by the user in order to obtain an analytic expression for the derivative.

Nolan's program (8) also requires encoding of the expression to be differentiated by the user. However, the transformation into tabular form in this case is started at the outermost parenthesis-pair of the fully parenthesized input expression and is progressed inwardly. Upon input to the computer a simple differentiation operator is introduced. By recursive application of the elementary rules of differentiation this operator is advanced down through the lines of the tabular form of the input expression until it is finally eliminated. During this process the tabular form of the expression is transformed by the elementary rules of differentiation so that when the differentiation

University of Cape Town

operator has been eliminated from the form, the resultant tabular form represents the desired derivative. This must then be decoded by the user into an analytical form of the derivative.

Slagle (11) has written a LISP program using the differentiation procedure described by Nolan as part of his SAINT automatic integrator.

Subsequent programs have been written using the same technique of differentiation with the refinement that the encoding of the mathematical expressions from the parenthesized linear form into tabular form is done automatically prior to differentiation and the inverse transformation before output.

Hanson et al. (10) have a different approach. The expression in analytical form (with only necessary parentheses included) is read into the program and converted into an internal form consisting of triples (left operand-operator-right operand), where the operators (including elementary functions of one argument represented as binary operators with dummy left operands) are ordered according to a precedence assigned to each of them. These triples are in effect equivalent to Polish prefix notation (see 3.1). The triple table for the derivative can then be built up by applying the differentiation rules for the various operators to the triples in sequence. This is then decoded into an output analytical form with only necessary parentheses included (possible since the precedence of the operators is known). This program was extended to get rid of redundancies caused by multiplication by 1, addition of 0 etc. Such extension was necessary to save storage space. The internally used triples are stored in matrix form in the computer. (10) includes flowcharts of the procedures used in this program.

More recently list processing has been widely used for performing symbolic manipulation. There are several programs written in LISP which are specifically intended for differentiation, or include such a facility. A general algebraic language called REDUCE 2 (13) was developed by A. C. Hearn for use in quantum electrodynamics. SLIP was used by Lapidus and Goldstein (6) to develop an algebraic package (SYMBOLANG) which included differentiation. Since SLIP was found to be slow, they intended rewriting the system using a faster list processor.

There are several other techniques which have been applied to symbolic differentiation. A syntax-directed compiler was

University of Cape Town

used by Schorr (14). P. J. Smith (15) used arrays and pointers for differentiation, avoiding the necessity of using a list processor or recursion. Wengert (16) and Wilkins (17) have developed a semi-analytical technique for obtaining the derivative of an expression at a point. This is of limited usefulness.

Most of the modern high-level languages used for symbolic manipulation incorporate differentiation as well. FORMAC, ((18), (19) & (20)), has this facility and since it also has formidable simplification techniques, it has the capability of outputting a derivative in a reasonably concise form. However, it has heavy core requirements, restricting the size and complexity of expressions it can deal with considerably at most installations where it is used.

Less general purpose languages can deal more efficiently with the core requirements of an automatic differentiator, e.g. REDUCE 2 (13). MATHLAB, also written in LISP is intended for use in "conversational mode" (21). ALAM (22), also LISP based, has differentiation facilities, as has the B-code system of Barton et al. (23). These two systems have been used extensively in calculations in general relativity. (23) also provides a table (p. 39) comparing the time and core requirements of four algebraic manipulation systems, results which seem to indicate that for the program tested, B-code is faster and uses less core than the other systems in the comparison. Numerous other systems exist which have facilities for doing automatic differentiation.

1.2 Problems Encountered in Automated Algebraic Manipulation.

The basic problems are the vast amounts of storage which are used by such routines in order to solve realistic problems, and the time consumed in the process. That the former problem should occur is clear if one considers for instance the expansion of a product of two polynomials of n terms each, in which case as many as n^2 terms are produced. In general, when expansion of factored expressions is attempted, storage utilization tends to increase exponentially rather than linearly. Only in some specialised cases can this be avoided. Differentiation also tends to expand expressions at a faster than linear rate. For instance tests carried out with the SLIP differentiator described in Chapter 3 indicate this clearly.

The other problem is that symbolic manipulation is very much slower than numerical use of a computer. In (23) the table

University of Cape Town

mentioned above indicated times of the order of minutes taken to perform symbolic calculations. (23a) also illustrates the problems encountered clearly.

The first of the above-mentioned problems is far more restrictive than the last. Hence simplification techniques are vital to any symbolic manipulation program. This was realised early in the development of this field of computer science. Current trends seem to be to develop a mathematical theory of simplification for application in practice. Involved are the ideas of zero equivalence ((34), (28)), regular simplification algorithms and canonical forms (29) for simplifying algebraic expressions. J. Moses (24) discusses this in some detail. Automatic simplification has been developed considerably in FORMAC (19).

Another attempt to reduce core and time restrictions is by restricting the class of algebraic expressions dealt with. ALTRAN (25) deals with rational functions. G. E. Collins has written a system called PM (26) for handling polynomials. W. S. Brown's ALPAK system was meant for truncated power series, polynomials and rational functions. The computing times of several systems including PM are compared in (27). Collins and Brown, among others, have done theoretical work in polynomial algebra ((30) to (33)) which should pay dividends in future algebraic manipulation systems. Extensions of the theoretical and practical innovations to other classes of functions are also likely.

It is nevertheless improbable that the field of symbolic manipulation will ever be completely free of core and time inefficiencies, since such problems seem intrinsic to the very nature of this work. Technological advances such as faster circuitry and laser based storage devices of enormous capacity should relieve some of the restrictions which are encountered in modern algebraic computer languages.

University of Cape Town

CHAPTER 2

THE SYMMETRICAL LIST PROCESSOR (SLIP)

2.0 Introduction.

SLIP is a list processing system distinguished by the symmetry of its lists: each list element is linked to both its predecessor and its successor. In contrast to most other list processors, SLIP is not an independent language but is intended to be imbedded in a general purpose higher language such as FORTRAN, in this case UNIVAC 1108 FORTRAN V (36). The original version of SLIP was developed by Professor J. Weizenbaum at MIT. His paper on SLIP (1) presents a complete documentation of the system, including a FORTRAN listing which is almost complete, and a statement of the underlying philosophy. Of particular note is the method of reclaiming unneeded nodes of lists for reuse (see 2.2).

In addition to the FORTRAN functions and subroutines, some basic assembly language routines ("primitives") are required. These are of a basic nature inaccessible to FORTRAN since they manipulate parts of words, extract the addresses of words and perform various other fundamental tasks. These primitives have to be implemented for each installation where SLIP is to be used. This is not difficult and this version of SLIP (henceforth referred to as U-SLIP) has its primitives coded in UNIVAC 1108 ASSEMBLER (37).

This approach to list processing has several advantages. The flexibility of FORTRAN is combined with the facility of manipulating list structures. It is fairly easy to install SLIP, since the coding of the majority of routines is available in (1) and SLIP is well documented in (1) and (2). The FORTRAN routines are largely machine independent, and the primitives upon which they are based are simple to code and debug once the linkage from FORTRAN to ASSEMBLER is known for a particular machine. This allows radical modifications of the node design of SLIP lists with the only programming changes necessary being to the primitives. However, if the number of words per node is changed, problems are encountered as was the case with U-SLIP. This is

discussed below. Additional routines may be added to the SLIP "library" by the user without much difficulty.

Disadvantages of a system such as SLIP include inefficiencies which a good compiler or interpreter may overcome. For instance, LISP is probably much faster in performing basic list processing operations than SLIP.

SLIP has been implemented at several installations for use in symbolic manipulation of algebraic expressions, as well as in other areas ((4) to (7)). An article by D. K. Smith (2) was found very useful in clearing up some matters dealt with only briefly in (1). It also includes excellent examples of the use of SLIP.

U-SLIP differs in several respects from Weizenbaum's original version (henceforth referred to as W-SLIP). It corresponds more closely to the official UNIVAC-supported SLIP package (3), known as X8-SLIP. However, it differs from this as well and hence a full description of U-SLIP is included below.

The SLIP routines operate primarily by side-effects, i.e. when used as functions, the function value is less important than the operations performed by the routine in obtaining the value. The subroutines are invoked by a FORTRAN CALL statement whereas the functions may be invoked this way or in the normally used manner, but not by both methods for a particular function in the same FORTRAN program.

The nodes of U-SLIP have had to be rearranged from those of W-SLIP since the latter was able to use the partial word features of an IBM 7090 computer. The UNIVAC 1108 has different partial word features, being a 36-bit word machine. A U-SLIP node uses three memory words rather than two. Though this means that 50% more core is utilized, more information can now be stored per node, vital to the SLIP differentiator (see Chap. 3). This change of node-size from W-SLIP meant that many changes had to be made to FORTRAN routines listed in (1), many of a decidedly tricky nature, and unfortunately much time was spent in debugging U-SLIP. More primitives had to be written to access the additional fields of a node made available by the larger node size.

The routines of U-SLIP were grouped into several subprograms with multiple entry points for efficiency in creating a program library. The grouping was effected in such a way that

University of Cape Town

no routine calls another (directly or indirectly) which resides in the same group. This was necessitated by FORTRAN V compiler restrictions. The ASSEMBLER primitives reside in two separate subprograms, with multiple entry points. Included are the basic primitives required to access the various fields of a node as well as the linkages necessary to implement SLIP-recursion (see 2.3.2.6), some character, bit and logical operations and various diagnostic aids.

The input/output routines implemented in U-SLIP are those of W-SLIP and not the X8-SLIP routines mentioned in (3).

2.1 SLIP Data Structure.

The most notable characteristic of SLIP as a list processor is the symmetry of its lists. Since the linking of list elements is done in both the forward and backward directions, lists do not have a preferred orientation. The last element is as easily obtained as the first.

The basic unit of storage in U-SLIP is the cell consisting of three consecutive memory words; each cell is divided up into various fields, according to the function of the particular cell (node). The ID-field, occupying the leftmost six bits (S1) of the second word of a cell distinguishes between the various SLIP cell-types.

ID = 0: datum cell. The datum is not interpreted as the name of a list (name-format is explained below).

ID = 1: datum cell in which the data is interpreted as the name of a list.

ID = 2: cell is the header of a list (unique to the list).

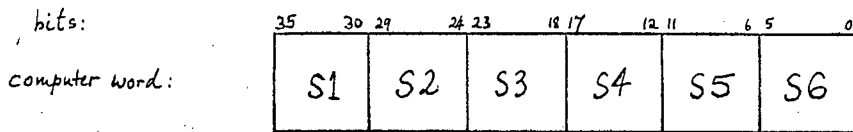
ID = 3: cell is the reader of a list (see 2.3.2.4).

In addition to these cells, there are description list (DL) datum cells which do not have an ID-field (see 2.3.2.5).

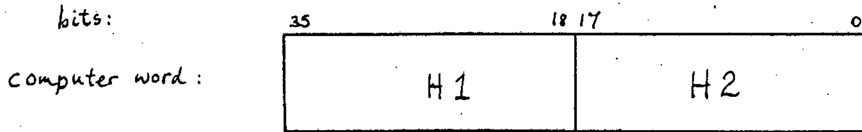
Figures 1 & 2 indicate the partial word features of a 36-bit 1108 computer word (with the various partial word designators), and a diagrammatic outline of the division of U-SLIP cells into fields, which are identified in most cases by means of the names of the functions used to reference the values stored in them.

The 1108 computer uses a 6-bit code ("fielddata") for alphanumeric and special symbols and hence one computer word can hold a maximum of six fielddata characters. Thus the datum of a U-SLIP cell can hold six symbolic characters, or integer

University of Cape Town



S ≡ sixth



H ≡ half

Fig. 1. Partial Word Designators of an 1108-Word.

Fig. 2. The Layout of U-SLIP Nodes.

node type: | node layout: | computer word:

(a). Header Cell (ID = 2):

				computer word
LNKL			LNKR	1
ID	MRK	FLG	HAF (unused)	2
DL			LCNTR	3

(b). Sublist Datum Cell (ID = 1):

				computer word
LNKL			LNKR	1
ID	MRK	FLG	HAF	2
DATUM (Sublist Name)				3

(c). Datum Cell (ID = 0):

				computer word
LNKL			LNKR	1
ID	MRK	FLG	HAF	2
DATUM				3

(d). Reader Cell (ID = 3):

				computer word
LPNTR			LNKR	1
ID	unused	unused	unused	2
LOFRDR			LCNTR	3

(e). Description List Cell:

		computer word
LNKL (negative)	LNKR	1
ATTRIBUTE		2
VALUE		3

University of Cape Town

or real numbers.

The LNKL and LNKR fields store the addresses of the preceding and succeeding cells on a list. All cells except those used as readers are part of a unique list, either created by the user or the "list of available space" (AVAIL) which is maintained by the SLIP system. The LNKL field is not always used. AVAIL itself is only linked in the forward (downward, left-to-right) direction. Reader cells are also only linked in the forward direction, and here the H1 field of word 1 of a node is used to contain the address of the cell to which the reader is currently pointing (LPNTR). This is as in W-SLIP. In contrast to W-SLIP, description (attribute-value) lists do not occupy two nodes, but utilizing the extra space available in a U-SLIP node, the attribute is stored in word 2 and the corresponding value in word 3. (X8-SLIP allows both forms of DL). Since there is thus no space for an ID-field, the LNKL field has the negative value of the address in a datum cell of a DL to distinguish it from the header cell of the DL which has the link-address in LNKL positive.

The ID-field of all other types of cell is stored in S1 of word 2. For cells with an ID of 0, 1 or 2, bits 7 to 12 (S2) contain a "mark" field (MRK) and bits 13 to 18 (S3) a "flag" (FLG). In W-SLIP only header cells contain a mark field to which U-SLIP header cell MRK fields are exactly equivalent. Whenever the mark of a <u>list</u> is referred to, the MRK field of its header is meant. The U-SLIP FLG field has no equivalent in W-SLIP.

For cells with ID = 0 (datum cells) there are two datum fields. The entire third word of such a cell is reserved for data. This corresponds to the datum field in W-SLIP. An additional half-word of data may be stored in H2 of word 2 of a datum cell (ID = 0 or 1). This is the HAF field (see fig. 2) and has no equivalent in W-SLIP.

Sublist cells (ID = 1) are arranged similarly to the datum cell except that while the HAF field may contain a datum, the third word is occupied by a list-name in name format (see below).

Header cells (ID = 2) contain a level (reference) counter in the LCNTR field (H2 of word 3) and the address of the header of a DL attached to the list in the DL field (H1 of word 3). The HAF & FLG fields may contain user data as they are not used

University of Cape Town

by the system. DL contains zero if a list has no DL attached. The MRK field can be used by certain functions included in SLIP.

Since each list has a unique header cell, lists may be referenced unambiguously by the address of the first word of the header cell. This is usually put into some data word in the program in name-format. This data word (FORTRAN variable) is then known as the name of the list. Name-format means that the address of the header of the list referred to occurs in both H1 and H2 of the word, e.g. if a list is created by the SLIP statement:

IONE = LIST(9)

and the address of the header cell of this list, taken from AVAIL, is 046157_8 then the variable IONE will contain the following:

046157046157_8

and at any time that the list is referenced in a program (either by IONE occurring as a function argument or if it is the datum in a sublist cell with ID = 1), the system will check whether such a list indeed exists and reference it. Similarly by simple assignment the name of the list may be transferred to some other variable in a SLIP program and yet remain a valid name of the same list.

A list A is said to be a sublist of another list B if the name of A appears as a datum in a sublist cell on B. A is a list in its own right however. Any list may occur as a sublist of another list. A tree may be represented simply using sublist nodes to point to branching-off subtrees. Common sublists are allowed. Thus a list appearing only once in memory may simultaneously be referenced as a sublist by several list structures, or several times by the same list structure. Figure 3 gives an example of a U-SLIP list structure with a reader referencing it. Figure 4 indicates an empty list.

2.2 Storage Allocation and Reclamation.

SLIP assigns storage dynamically using the list of available space, AVAIL, which is created by the SLIP initialisation routine acting upon a block of storage provided by the user. This is divided up into groups of 3 consecutive words and the whole structure linked in a forward direction.

Whenever a cell is required, the top cell of AVAIL is retrieved and the quasi-name of AVAIL, AVSL, which contains the addresses of the upper and lower limit cells of AVAIL in its H1 and H2 fields, is accordingly changed. When a cell is no

Main List

46157
047335 046162
der
ell
2 0 0 0
0 0

backward (leftward)

46162
046157 046170
op
0 0 0 0
D1

46170
046162 047335
1 0 0 0
046165 046165

forward (rightward)

47335
046170 046157
ttom
0 0 0 0
D2

Reader Stack

46516
046516 046712
3 0
046165 1

46712
046170 0
3 0
046157 0

Sublist

46165
046173 046173
2 0 0 0
0 1

46173
046165 046165
0 0 0 0
D3

Fig. 3. Example of a U-SLIP list structure to represent (D1, (D3), D2). The reader is shown pointing to the cell containing the datum D3.

047210
047210 047210
2 0 0 0
0 0

Fig. 4. An empty list.

longer required it is returned to the bottom of AVAIL, and AVSL is again altered. When it is required to delete a whole list, the symmetrical structure of SLIP is shown to advantage since this operation is independent of the length of the list to be deleted. The list is returned to AVAIL as a block, the top of the list linked to the bottom of AVAIL, and AVSL pointing to the bottom of the returned list. Hence only 3 links need be changed.

In such a dynamic storage allocation system, efficient use of space requires that space no longer needed be promptly returned to AVAIL. In some list processors (e.g. LISP) this is done entirely by system "garbage collection", a search of memory for data not referenced. In other systems reclamation is left to the user. In SLIP responsibility is shared between the system and the user. The essential concept is that of a reference count (stored in the LCNTR field of a list header). This keeps count of the number of times a given list occurs as a sub-list in a program. This count is automatically maintained by the system which increases it when the name of a list is stored on some other list and decreases it when the name is removed or such a containing list is itself erased. Hence the criterion for erasing a list is that whenever the reference count (RC) is decreased to zero, the list is not referenced and may be erased. So long as the RC remains positive, the list is in active use and must be preserved. Facilities are also provided for a user to prevent a list from being erased by giving it an initial RC of 1. Such a list may only be erased by explicit programmer intervention. A routine exists to accomplish this.

The above requires the system to maintain RC's dynamically. It is simple to decide when to increase a RC as every datum placed on a list is checked to determine if it is in fact the name of a list. In such a case the RC of the list is incremented by 1. Decrementing is more difficult. Since all routines removing or replacing the contents of a cell do so by returning the cell to AVAIL, SLIP is able to postpone decrementing until the cell containing the name is removed from AVAIL by the routine NUCELL. <u>The cell could not have been required before then.</u> NUCELL examines each cell as it is taken from AVAIL. If it is a list-name, the list referred to has its RC decremented and if the new RC is zero, the list is erased and returned to AVAIL. This deferred erasure approach is fundamental to the philosophy

University of Cape Town

of SLIP, having the intention of speeding up list processing. While it may be contended that this procedure is space wasteful (a more critical factor than speed in algebraic manipulation by computer), the argument is irrelevant as can be seen by what follows. Deferred erasure means that some list inaccessible to a program (i.e. with RC = 0) may remain unattached to AVAIL until the node referencing it (which has already been returned to AVAIL) is encountered there. But if such a node is found on AVAIL, the inaccessible list could not yet have been required anyway because there is still at least one node in AVAIL for use by the program, and since when the node refers to an inaccessible list the latter is immediately returned to AVAIL, further storage is again available.

2.3 The SLIP Routines.

2.3.1 The Primitives.

Although the primitives have been coded in assembly language, the user calls them exactly as if they were FORTRAN functions or subroutines. Those primitives which do not return a value are to be treated as subroutines only.

Note: In what follows, references to cells occur in two ways. If a cell reference is written as "A", variable A contains the address of a cell on which the function is to act. If the reference is to "CELL", the function acts on variable CELL directly. An asterisk on the left of a function-definition below indicates that it is extant in U-SLIP only.

University of Cape Town

CONT(A) INHALT(A)	These two routines are identical, the duplication existing only to avoid FORTRAN real-integer naming conventions which could cause automatic conversion of variables according to FORTRAN naming rules. For instance, V=INHALT(J) where V is automatically typed REAL (unless this rule is overwritten) should be replaced by V=CONT(J). value: The contents of the cell whose address is stored in A.
MADOV(CELL)	value: The machine address of variable CELL.
LNKL(CELL) LNKR(CELL)	value: The LNKL (LNKR) field of CELL, returned in integer form.
* LNKLP(CELL)	value: The absolute value of LNKL of CELL.
ID(A)	value: The ID field of the cell whose

	Routine	Description
		address is in A, in integer format. It accesses S1 of address (A)+1 where the brackets denote "the contents of variable A".
*	GETMRK(A)	value: The mark field of (A), i.e. S2 of (A)+1.
*	GETFLG(A)	value: The flag field of (A), i.e. S3 of (A)+1.
*	GETHAF(A)	value: The half-datum field of (A), i.e. H2 of (A)+1.
*	PUTMRK(D,A)	This function stores D in MRK of (A). value: A.
*	PUTFLG(D,A)	This stores D in FLG of (A). value: A.
*	PUTHAF(D,A)	This stores D in HAF of (A). value: A.
	STRDIR(X,Y)	This stores X in variable Y. It is equivalent to FORTRAN assignment, except that automatic type conversion is avoided. value: (X).
	STRIND(X,Y)	The contents of X are stored in address (Y). value: (X).
	SETDIR(N1,N2, N3,CELL)	This subroutine stores N1, N2, N3 in the ID, LNKL, LNKR fields of CELL respectively. If any of them are -1, the corresponding field of CELL is not changed from its previous value.
	SETIND(N1,N2, N3,A)	As SETDIR except that the cell into which the values are stored has address (A).
*	SETD3(N1,N2,..., N8,CELL)	This is an expanded version of SETDIR in which N1 to N8 are stored respectively in the LNKL, LNKR, ID, MRK, FLG, HAF, DL and LCNTR fields of CELL. If any of them are -1 the value originally in the corresponding field of the cell remains unchanged.
*	SETI3(N1,N2,..., N8,A)	An expanded version of SETIND, similar to SETD3, except that the cell referenced has address (A).
	INTGER(CELL) }	These routines exist solely to circumvent the FORTRAN naming conventions.
*	REALNO(CELL) }	

University of Cape Town

value: CELL.

Tracing Routines:

* TRCON * TRCOFF	Subroutines to switch on (off) the 1108 ASSEMBLER trace routine SNOOPY.
* WLKBCK(ARG)	A subroutine to invoke UNIVAC 1108 "walk-back" which exits a program in error mode if it is encountered. It is used when SLIP errors occur. A printout occurs of the FORTRAN program sequence numbers of references to an inner subroutine in the hierarchy of calls, all the way back to the main program, before exiting the program. Example: If the main program calls subroutine A (at sequence no. 190) which in turn calls function B (at sequence no. 212) and "CALL WLKBCK(3)" is encountered in B, the walkback printout would be:- ERROR EXIT IN B ROUTINE B CALLED AT SEQUENCE NO 212 OF A A CALLED AT SEQUENCE NO 190 OF MAIN PROGRAM In the call, argument ARG must be the number of arguments of the routine within which walkback is invoked.

Character and Bit Manipulation Routines:

LANORM(DATUM)	value: The contents of variable DATUM ring-shifted to the left to remove leading blanks.
SQOUT(MASK,S)	value: The field from S specified by MASK (1-bits set for the relevant field, 0's otherwise), shifted right once for each trailing zero bit in MASK. Shifting introduces leading zero bits.
SQIN(MASK,DATUM, DEST)	The value, (DATUM), is shifted left once for each trailing zero bit in MASK and then placed into the field of DEST specified by MASK (1-bits set for the relevant field). value: New contents of DEST.
SHIN(N,DATUM, DEST)	The contents of DEST are shifted left N bits, introducing trailing zero bits. The N rightmost bits of DATUM are inserted into

University of Cape Town

the vacated positions.
value: New contents of DEST.

2.3.2 The FORTRAN Routines.

2.3.2.1 Storage Allocation.

INITAS(SPACE,N) The initialisation routine. SPACE is an array of at least N words. AVAIL is created by forward linking words of SPACE to form N/3 3-word cells. The last cell of AVAIL always has a zero LNKR field for recognising when AVAIL is exhausted. The public lists W(1) to W(100) (the names of the lists are stored in array W: see below) are also created and left empty but with their RC = 4095 (7777_8) to ensure they are not erased. The names of W(i) as well as AVSL (see 2.2) are stored in a named COMMON, SLIP, so that any program which is to use the public lists must include the statement

```
COMMON /SLIP/ W(100)
```

The first executable statement of any program written in SLIP should be:

```
CALL INITAS(SPACE,N)
```

IRALST(L) The RC of list L is reduced by 1 and if the new value is zero, L is returned to AVAIL.
value: The new RC-value.

NUCELL(DUMMY) A cell is taken from the top of AVAIL. If none are available the program is terminated by an error message and walkback. DUMMY is a dummy argument.
value: The address of the new cell obtained.

RCELL(A) The cell the address of which is in A is placed on the bottom of AVAIL.

* LSPACE This subroutine counts the number of nodes of AVAIL and prints it. It is useful to determine core utilization at any point in a U-SLIP program, but should be used sparingly since it has to traverse every node of AVAIL individually and is thus very slow.

University of Cape Town

2.3.2.2 Data Manipulation and Testing Routines.

LIST(L) — An empty list is created. If L is the literal 9, LCNTR is initially zero. Otherwise LCNTR is 1 and the name of the created list is placed in L.
value: Name of the created list.

NEWBOT(D,L) }
NEWTOP(D,L) } — A cell containing the datum D is inserted immediately above (below) the header of list L.
value: Address of new cell taken from AVAIL.

NXTLFT(D,C) }
NXTRGT(D,C) } — A cell containing datum D is inserted to the left (right) of the cell whose address is in C.
value: Address of the new cell taken from AVAIL.

Note: Calls on these functions may be embedded because of the values they return, e. g.

ITEM = NXTRGT(D1,NXTRGT(D2,NEWBOT(LB,LA)))

SUBST(D,C) — The datum D replaces the datum in the cell whose address is in C.
value: The previous datum.

SUBSTP(D,L) }
SUBSBT(D,L) } — The datum D replaces the datum at the top (bottom) of list L.
value: The previous datum.

The above three substitution routines operate by returning the old cell to AVAIL and obtaining a new one with the desired datum. This is of no concern to a user unless a reader or sequencer is pointing to the old cell, in which case STRIND should rather be used.

LSSCPY(L) — List L is copied and left unchanged. The copy contains the same data and structure as does L.
value: The name of the new copy.

NULSTL(C,L) }
NULSTR(C,L) } — C contains the address of a cell on list L. L is split, with all the cells on the left (right) of C (inclusive) placed on a new list, while its remaining cells are left in L.
value: Name of the new list created.

INLSTL(L,C) } — The entire list (excluding its header) is

INLSTR(L,C)}	inserted to the left (right) of the cell whose address is C. L is made into an empty list. value: L.
POPBOT(L) POPTOP(L)	The bottom (top) cell on list L is returned to AVAIL. value: Datum on the removed cell.
DELETE(C)	The cell whose address is in C is returned to AVAIL. An attempt to delete a list header this way fails, resulting in an error message and return of the value 0.0. value: The datum in the removed cell C.
MTLIST(L)	List L is made into an empty list and all its cells (excluding the header) are returned to AVAIL. value: L.
BOT(L) TOP(L)	value: Datum stored in the bottom (top) cell of list L.
NAMTST(X)	value: 0 if X is a list-name, -1 otherwise.
LISTMT(L)	value: 0 if list L is empty, -1 otherwise.
EQUAL(A,B)	value: 0.0 if A=B, -1.0 otherwise.
LOCT(L)	This checks whether list L exists and if not an error message is printed and walk-back invoked. The system routines use this function constantly to ascertain whether lists are still in existence at any stage of a program. value: L.
LSTEQL(LA,LB)	value: 0 if list structures LA and LB are equal, -1 otherwise. Two list structures are equal only if all data are identical and the same sublists are referred to in the same relative positions of the structures.
LSTMRK(L)	value: The mark of list L (see 2.1).
MRKLST(M,L)	The mark of list L is set to M. value: L.
MRKLSS(M,L)	The mark of list L and all its sublists is set to M. value: L.

University of Cape Town

MADLFT(C) MADRGT(C)	value: The machine address of the cell to the left (right) of the cell whose address is in variable C.
MADNBT(L,N) MADNTP(L,N)	value: Machine address of the N-th cell from the bottom (top) of list L.

2.3.2.3 Sequencing Mechanism.

After creating a list structure one may wish to process its elements sequentially. A sequencer (sequence reader) assumes the housekeeping burden in this task. A sequencer is a pointer containing the address of some cell in a list. It occupies only one machine address (a variable in a user program) rather than a node from AVAIL. It is not necessarily something unique and several may be operative simultaneously, even on the same list.

The utility of a sequencer arises from routines designated to advance it to point to the next cell of a list, either to the left or the right. The mode of the advance may be linear (in the same list), or structural (possible descent into some referenced sublist). Information about the next cell is provided by use of a flag, a user-provided variable.

SEQRDR(L)	A sequencer is created, pointing initially to the header of list L. value: The address of the header of L.
SEQLL(S,F) SEQLR(S,F)	The sequencer S is linearly advanced to the next cell to the left (right) of the cell to which the sequencer is pointing. The flag F is set to +1 if the new cell is a header, 0 if it is a list-name and -1 otherwise. value: Datum stored in the new cell. The flag can be tested by the user to decide on a course of action.
SEQSL(S,F) SEQSR(S,F)	The sequencer S is structurally advanced to the next cell to the left (right) of the cell to which the sequencer is pointing. Thus if S initially points to a cell containing a list-name, the next cell S will point to is the bottom (top) of the list of which it encountered the name (unless that in turn contains a name). value: Datum stored in the new cell finally

University of Cape Town

encountered. The flag F is used as with linear advances. A sequencer cannot ascend back into an outer list after it has made a structural advance into a referenced inner one.

2.3.2.4 Readers and Advance Routines.

Often it is necessary to be able to trace through a list structure descending into a sublist and ascending back into the original list whenever appropriate. Sequencers, as stated in 2.3.2.3, cannot ascend back into a higher level, since they cannot retain historical information of their advances. A reader is a generalisation of the sequencer concept, consisting of a pointer with a memory which is in effect a pushdown stack of pointers with only the top one active at a given time. The others store the previous pointers and when popped up, can ascend back into an outer list at the position where the reader descended into the sublist, once the sublist has been traversed.

A reader consists of cells taken from AVAIL. The address of the top of the reader stack (the active reader cell) is known as the name of the reader. A reader has fields as indicated in figure 2. (See also 2.1). LPNTR is the address of the cell currently pointed to. LOFRDR is the address of the header of the list currently being traced and LCNTR is a level counter showing how "deep" the reader is in the list structure (i.e. how many times the reader has descended into sublists). The LNKR field links the reader stack in the forward direction, the bottom cell having an LNKR field of value 0.

When a reader is initially created for a list, LPNTR and LOFRDR are given the address of the header of the list and LCNTR and LNKR set to zero. If an advance (see below) causes the reader to descend into a sublist, a new cell is taken from AVAIL and linked to the currently active reader cell. LPNTR and LOFRDR of the active cell will now refer to the sublist and LCNTR be increased by 1. This continues until the header of the sublist is reached, which implies that all cells on the sublist have been processed. Then the new cell is returned to AVAIL and the old LPNTR, LOFRDR and LCNTR fields are restored.

The mechanism for keeping track of the reader is thus system provided. The user need only create a reader and then use the advance routines, similar to those of a sequencer, except

University of Cape Town

that the user requests searching for a target (certain kind of cell). The possible targets are:-

element (E) : ID = 0
name (N) : ID = 1
word (W) : ID = 0 or 1.

By specifying the mode (linear or structural), target (element, name or word) and the direction (left or right), there are 12 possible advances, the name of the advance routine suggesting its action. For example, ADVLER is the routine for advancing linearly element right.

LRDROV(L)
: A reader of list L is created, initially pointing to the header.
value: Name of the reader.

ADVLEL(R,F)
ADVLER(R,F)
ADVLNL(R,F)
ADVLNR(R,F)
ADVLWL(R,F)
ADVLWR(R,F)
: The reader R is advanced in the indicated direction, searching for a cell of the type specified by the target. Since these are linear advances, descent into sublists does not take place. If the header is encountered before the target is found, the search stops with flag F nonzero. If the target is found, F is set to zero.
value: If the target is found, the datum in the cell found is delivered, otherwise 0.

ADVSEL(R,F)
ADVSER(R,F)
ADVSNL(R,F)
ADVSNR(R,F)
ADVSWL(R,F)
ADVSWR(R,F)
: The reader R is advanced in the indicated direction searching for a cell of the type specified by the target. The search is structural so that until the target is found, any list-name will cause the reader to descend into the sublist and continue the search. Recognition of the header of the main list stops the search, while any other header causes the reader to ascend in the list structure and continue the search. The flag F is made non-zero if the target is not found, zero if it is.
value: As above.

LOFRDR(R)
: value: The name of the list currently being traced by the reader R.

LPNTR(R)
: value: The address of the cell to which the reader R is currently pointing.

University of Cape Town

LCNTR(R) value: The level counter of reader R.

REED(R) value: The datum in the cell to which the reader R is currently pointing.

INITRD(R) The reader R is made to point to the header of the current list.
value: R.

LVLRVT(R) Reader R ascends in the list structure until it is pointing to the cell in the main list from which it last descended. Nothing happens if R is already pointing into the main list.
value: R.

LVLRV1(R) As above, except that R ascends only one level.
value: R.

LRDRCP(R) The reader R, including its associated linking cells is copied. This gives two readers of the same list which can be advanced independently.
value: Name of the copy of R.

IRARDR(R) The reader R, including its associated linking cells is returned to AVAIL.
value: Level counter of R.

ADVLL, ADVLR, ADVSL and ADVSR are internal SLIP routines used by the advance functions and are of no concern to a user. They are merely separated from the latter routines to avoid a three-fold duplication of coding.

LSTPRO(D,R) This routine searches through a reader stack for a LOFRDR reference to datum D, a list-name, i.e. it seeks to find whether the reader has already traversed into D at some stage. This routine is used by LPURGE (see below).
value: 0 if R has traversed through list D previously, -1 otherwise.

LPURGE(L) A reader is appointed for list L. It then searches structurally through the list structure, removing all references to all sublists except the latest reference to each of them.

University of Cape Town

value: The number of cells deleted from L.

The above two routines are not documented in (1) or (2) although FORTRAN listings of them are included in (1). Their use appears not to be of general interest.

2.3.2.5 Description (Attribute-Value) Lists.

A description list (DL) may be associated with any list. It is composed of cells linked together symmetrically but with the LNKL field address stored negatively. Word 2 of each cell is said to contain an attribute, word 3 the corresponding value. Such a list is not a sublist, since its name does not appear as a datum on the host list. Instead its address is stored in the DL field of the header node of the host list. DL's cannot be manipulated by advance routines.

NEWVAL(AT,VAL,L) — The DL of list L is searched for the attribute AT. If found, the corresponding value is replaced by VAL. If not found, AT and VAL are added at the bottom of the DL. If no DL exists, one is created containing AT and VAL.
value: The old value of AT if one exists, zero otherwise.

ITSVAL(AT,L) — The DL of list L is searched for the attribute AT. If no DL exists, an error indication is given, printed out by the subroutine DERROR(L).
value: The value corresponding to AT if AT is found, zero otherwise.

NAMEDL(L) — value: The address of the header of the DL of list L, if one exists, 0 otherwise.

MAKEDL(LA,LB) — The list LA (which must already have been created) is made the DL of list LB, replacing any previous DL.
value: LB.

NOATVL(AT,L) — The DL of list L is searched for the attribute AT. If found, AT and its corresponding value are removed.
value: Old value corresponding to AT if AT was found, zero otherwise.

MADATR(AT,L) — value: Machine address of attribute AT on DL of list L if it exists, -1 otherwise.

University of Cape Town

MTDLST(L) The DL of list L, if it exists, is made into an empty list.
value: L.

LISTAV(AT,VAL,L) If a DL of list L does not exist, one is created. AT and its corresponding value VAL are stored at the bottom of the DL, without a search for AT in the DL nodes already extant. (Compare NEWVAL above.)

Examples where DL's are used are given in (2).

2.3.2.6 Recursion.

A subterfuge must be resorted to in SLIP in order to permit recursion in a FORTRAN program. Each block of programming to be entered recursively must begin with a uniquely labelled statement. In W-SLIP this label is assigned to a unique variable using the ASSIGN statement. In U-SLIP this has been avoided since the compiler objected to ASSIGN variables being used in any way except in GOTO statements. Instead the FORTRAN V technique of putting the statement label, prefixed with a dollar sign, as an argument in a function call was used.

SLIP-recursion uses a pushdown list of return addresses. The routine VISIT initiates the recursion by storing the address of the next instruction in sequence and then branching to the statement label specified as an argument. The routine TERM terminates the recursion at the end of the block of statements entered recursively by a transfer to the address stored in the top of the pushdown list.

The public lists W(1) to W(100) (see 2.3.2.1) are useful for transmitting arguments and saving local variables. In U-SLIP the list used for return addresses is in fact an additional public list W(101).

There is no difference between W(i) and user defined lists, except that the former are predefined and have LCNTR=4095 to ensure that they are not inadvertently erased during a program by the storage reclamation mechanism.

A description of VISIT and TERM, which were hand-coded in assembly language follows.

VISIT($\$$j,X) This function picks up the return address from the function reference to VISIT. The statement

DUMY=VISIT($\$$10,PARMT2(3,4))

University of Cape Town

has 1108 ASSEMBLER equivalent:-

```
        LMJ X11,PARMT2
        +   (3)
        +   (4)
        +   walkback word
        SA  A0,ARG2
(1)     LMJ X11,VISIT
(2)     J   10L
ARG2    +   0
        +   walkback word
(3)     SA  A0,DUMY
```

When statement (1) has been executed, register X11 contains the address of statement (2) in its H2 field (see fig. 1). Since the number of arguments (represented by the 3 statements after (1)) are fixed, the return address can be determined by storing H2 of X11 on the pushdown list, and upon return from the relevant recursion adding 3 to this value and jumping to the resulting return address, in this case statement (3), which corresponds to the assignment part of the FORTRAN statement above. In statement (2), 10L is the label attached to the first ASSEMBLER statement of those generated by the FORTRAN statement labelled 10. The first argument of VISIT, j, is the label attached to the first of the block of statements to be entered recursively. The second argument, X, is not used in the VISIT function, but as can be seen in the example above, is useful for saving variables before the recursion takes place. (See PARMT2 below). If PARMT2 or PARMTN are not used, the second argument serves no purpose but should still occur, e.g. as follows:-

```
QONE=VISIT($358,-1)
```

After the address of the current statement has been determined by VISIT, it is stored

University of Cape Town

on top of W(101) and control is transferred to the statement of which the label is the first argument of a call on VISIT.

The current level of recursion is ascended from by the use of the following routine:-

TERM(V,X) The address stored at the top of W(101) is popped up and retained by the routine. The first parameter V is inserted into accumulator register A0, which is used by FORTRAN V to carry over the value of a function upon return to the program calling it. See example above under VISIT. In that example TERM returns control to statement (3), and V is returned as the value of the corresponding VISIT function which transferred control to the block of coding terminated by the current call on TERM. Hence DUMY has the same value as V. The second argument, X, is dummy (cf. VISIT above). It is included for an optional call on the RESTOR function (see below) to pop up the values saved on the W(i) when recursion was entered at the corresponding VISIT function. Control is transferred to the address obtained by popping up W(101) and adding 3 to it (see description of VISIT above).

PRESRV(N) The first N public lists, W(1) to W(N) have their top values again pushed down onto them.

RESTOR(N) The first N public lists are popped up.

PARMT2(A,B) A is pushed down onto W(1), B onto W(2). value: A.

Since the FORTRAN V compiler does not allow a variable number of arguments in a function call, the W-SLIP version of PARMTN (see (2), p.407), with variable N arguments had to be revised as follows:-

* PARMTN(V,N) V is an array, dimension N which contains the N values to be pushed down on W(1) to W(N). value: V(1).

2.3.2.7 Input/Output Routines.

These are relatively undeveloped in W-SLIP, and not without good reason, since not only is FORTRAN I/O available to the user, but I/O routines are highly machine dependent. Word sizes of computers vary widely and the representation of alphanumeric data changes as well. For instance, the RDLSTA routine (see below) required a major overhaul of the W-SLIP version for inclusion in U-SLIP. Three I/O routines are available in U-SLIP.

* LSTRCE(L,NAME,N) This subroutine prints out in sequence all the nodes of a list L as they are encountered. Although it cannot print more than one linear list at a time and omits to list any sublists referred to in L, it was found to be useful as a diagnostic aid. NAME is a Hollerith name of at most six characters which is included in the heading of the printout of L. The first two words of the nodes of L are printed in octal format. The same applies to the third word of header and sublist nodes. If argument N of the routine is 0, all remaining datum words are printed in octal format, whereas if N is 1 fielddata ("A") format is used.

RDLSTA(DUMMY) This function reads a list structure punched on data cards. Each list and sublist is enclosed in parentheses with elements separated by blanks (one or more). The data are stored in fielddata form (see 2.1), 6 characters to a datum word of a node. Any datum with fewer than 6 characters is left justified and the waste space blank-filled. Only the first 72 card columns are read and if the end of the list structure (indicated by the mate of the initial left parenthesis) has not been found by then, another card is read. Cards are printed as soon as they are read.

value: Name of the list created.

PRLSTS(L,N) A subroutine to print a list structure L, one element per line. The format of the

data is I14 if N=1 (integers), A6 if N=2 (alphanumeric data) and F10.4 if N=3 (real numbers). The beginning and end of sublists are indicated by messages and by indentation.

Examples of the use of SLIP are given in (1) and (2). Further illustrations are provided by the source listing of SLIP itself, especially of the higher level routines, e.g. LSSCPY and LSTEQL, which use SLIP-recursion, and RDLSTA where bit and string manipulation primitives are used to advantage. The full source listing of the routines of U-SLIP may be found in Appendix A.

The AND and OR functions of W-SLIP are not provided in U-SLIP since they are available, together with several other logical functions, in FORTRAN V.

University of Cape Town

CHAPTER 3

DERIV, A SYMBOLIC DIFFERENTIATION PACKAGE

3.0 Introduction.

A set of routines written in U-SLIP for performing analytical differentiation by computer is described below. Input and output routines are included, having useful features such as I/O of expressions in a notation as close as possible to mathematical conventions. The system is designed to be expanded in scope, and simplification of derivatives is attempted to a certain extent.

3.1 Trees, Polish Notation and Algebraic Expressions.

A tree is defined as a finite set of nodes such that:

(i) there is one specially designated node called the root T;

(ii) the remaining nodes are partitioned into m ($\geqslant 0$) disjoint sets T_1, T_2, ..., T_m where each of these sets is in turn a tree. A single node is also a tree.

A terminal node comes at the end of a branch of a tree, whereas a branch node is one that is not a terminal node. The degree of a node is the number of branches extending from it. A binary tree has all its branch nodes of degree 2. A tree may be represented diagrammatically as in figure 5.

Any algebraic formula may be represented by a tree, terminal nodes being operands and branch nodes of degree n representing n-ary operators. Functions of n arguments may be regarded as n-ary operators, operands as nullary operators. All the elementary functions and operators commonly used in algebra and trigonometry are either unary or binary. If a null-operand (here represented by the symbol £, in the DERIV system by the cross-hatch or number sign, to use the ASCII term for it) is introduced, the unary operators can be represented as binary, so that only binary trees need be considered here for the representation of elementary algebraic expressions. P. Wegner considers operators of generalised degree ((35), p.240), and in fact a very powerful differentiation routine may be based on this concept.

Figure 6 shows the tree representations of several algebraic expressions. Note that trees are parenthesis-free and the

University of Cape Town

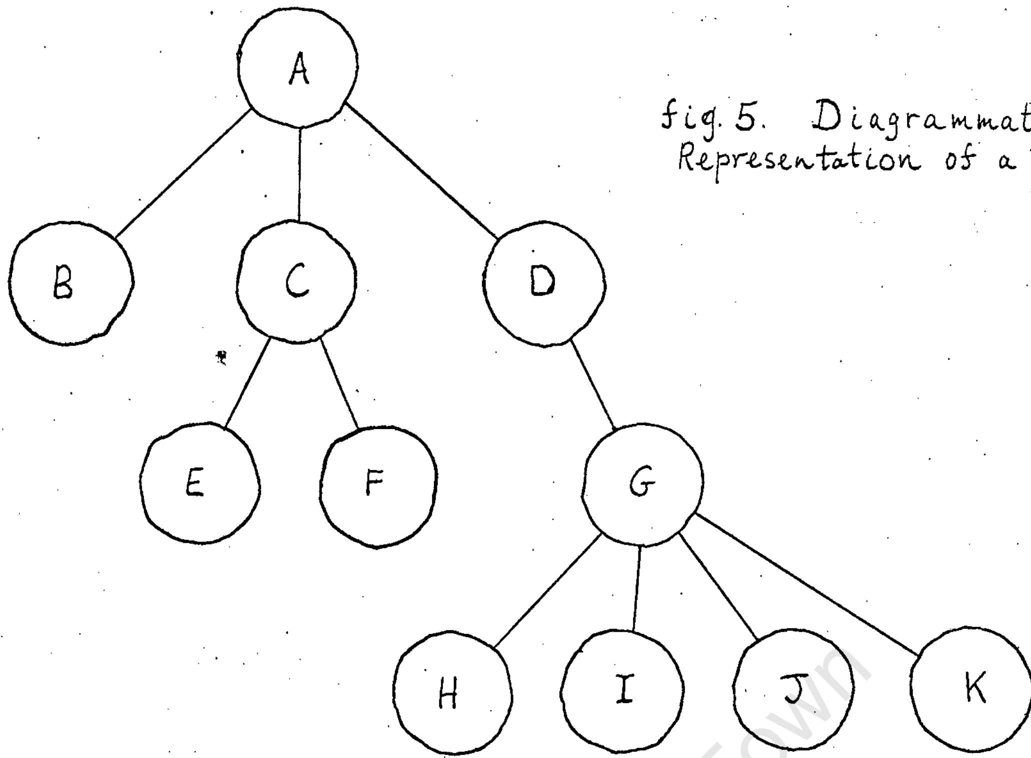
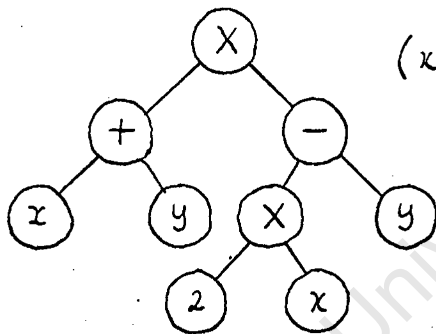
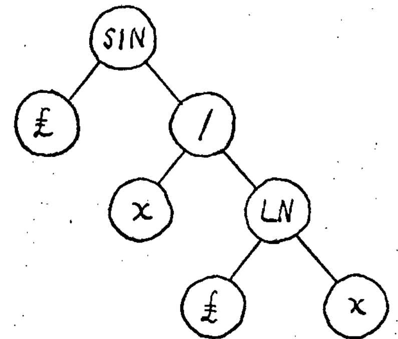


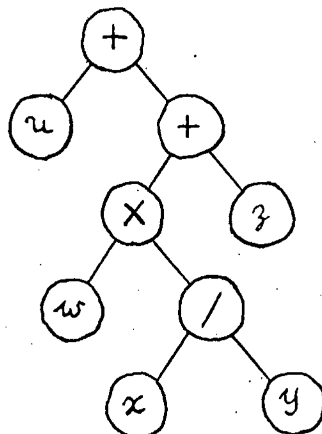
fig. 5. Diagrammatical Representation of a Tree



$(x+y) \times (2 \times x - y)$



$SIN(x / LN\,x)$



$u + w \times x / y + z$

fig. 6. Representations of Algebraic Expressions by Binary Trees.

University of Cape Town

precedence of the operators of an algebraic expression is clear from the tree structure which represents it.

By traversing binary trees in the three possible manners, three different methods of representing an algebraic expression linearly can be generated.

By traversing such a tree, visiting the root at any stage, then its left subtree and then its right subtree until all nodes have been visited (preorder traversal), Polish prefix notation is generated. The examples of figure 6 give the following results:

X + x y - X 2 x y
SIN £ / x LN £ x
+ u + X w / x y z

It may be observed that an operator always precedes its left and right operands (be they single variables or expressions).

Traversing a tree in the order left subtree - root - right subtree (postorder traversal) leads to the method usually employed for writing mathematical expressions (after necessary parentheses have been inserted to preserve the precedence of the operators of the expression). This is known as infix notation.

The third possibility for tree traversal (left subtree - right subtree - root) known as endorder traversal, leads to reverse Polish (Polish suffix, postfix) notation. The examples of fig. 6 are again used for illustration.

x y + 2 x X y - X
£ x £ x LN / SIN
u w x y / X z + +

Here an operand is always preceded by its two operands. Polish suffix notation is well known to compiler writers since it is the most convenient way of evaluating expressions, using a push-down stack. It is also simple to convert infix expressions to their Polish suffix counterpart by using a stack to push down operators while carrying across operands to the converted expressions. Operators are popped off the top of the stack if the precedence of an operator about to go into the stack is less than or equal to (see note below) those at the top of the stack, otherwise the operator is stacked immediately. The precedences are preassigned to all the possible operators. The algorithm for converting infix to Polish prefix notation is similar (see 3.3.3). This form of Polish notation, rather than reverse

Polish is used in DERIV (see 3.3). Flowcharts for performing the two conversion algorithms are included (figures 7 & 8).

Note: Consider an example of the conversion from infix to reverse Polish notation, say A + B - C. There are two equally valid reverse Polish representations of this, A B + C - and A B C - +. Which is used depends on whether the order of evaluation of the operators of an expression is to be right-to-left or the opposite. The flowchart in fig. 7 produces the first form. If the second were required, operators would be popped off the stack if the precedence of an operator about to go into the stack had a precedence strictly less than those at the top of the stack. A similar duality exists in the Polish prefix representation of algebraic expressions (see 3.3.3). Note as well that the precedences assigned the binary operators in fig. 7 are unique so that the case of equal precedences only arises when, for example, an expression has several terms or factors.

To convert an expression from Polish prefix to infix form is more difficult than the reverse. In fig. 9 a recursive method for accomplishing this is flowcharted, since it is used by the output routine of the differentiator.

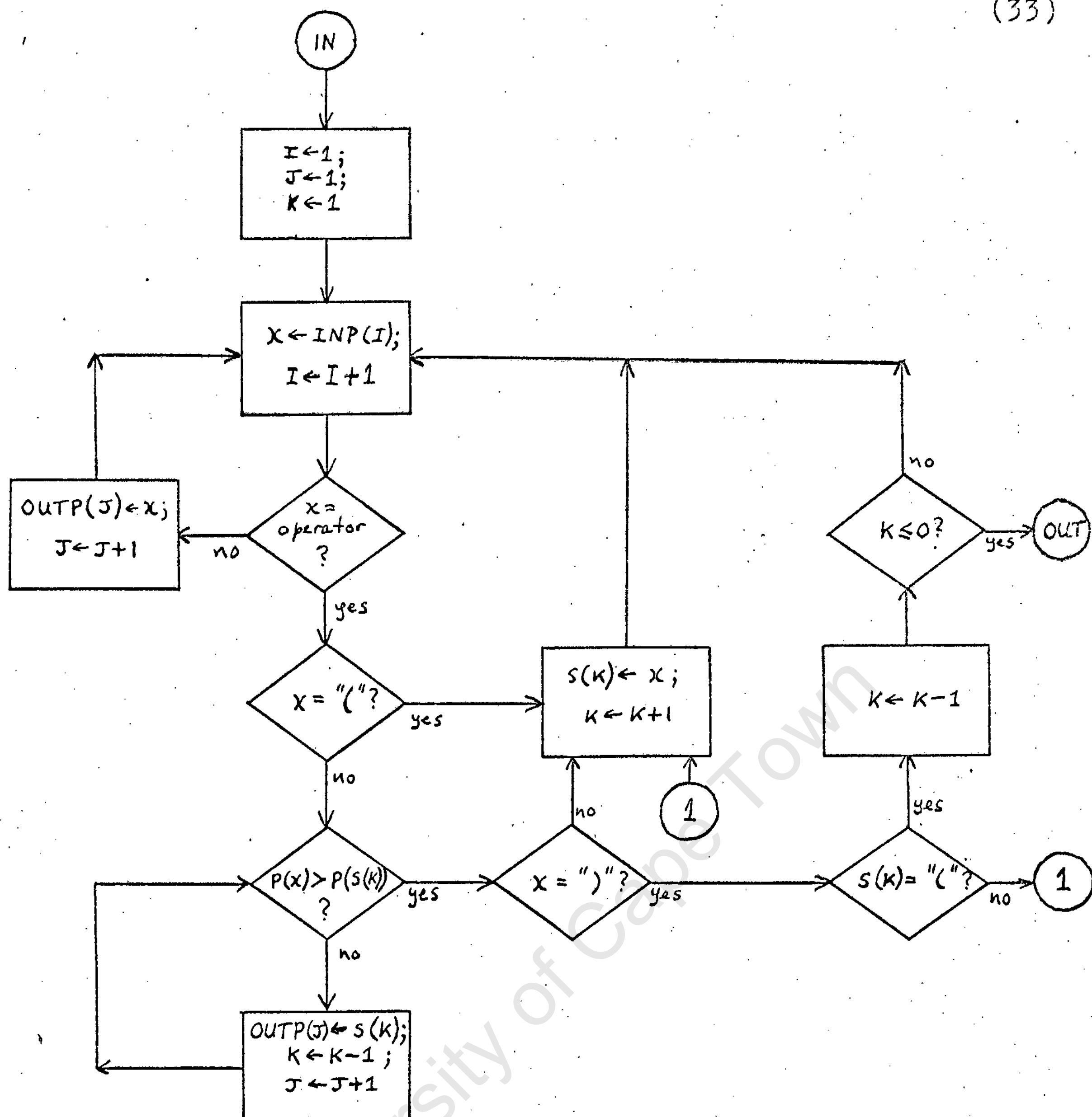
3.2 The SLIP Differentiation Package, DERIV.

As mentioned in 3.1, an input expression in infix form is translated to Polish prefix notation and then differentiated, whereupon the derivative may be translated back to infix notation and output.

The usefulness of Polish prefix notation is that the expression may be scanned from the left, and depending on whether the nodes encountered are operands or operators, appropriate action can easily be taken in building up a derivative. Since an operator is always encountered before its two operands, the differentiation routine can immediately decide what differentiation rule to apply, represented by a module in the routine. At each step in the recursive routine the left and right operands of the operator are delineated. The routine considers them as the appropriate rule requires. If they are to be copied to the derivative this is done and if differentiation is required, the operand is examined recursively to see whether it is a single variable or a subexpression. The derivative is built up linearly from the left, also in Polish prefix notation.

The allowable operators and functions are the usual unary

University of Cape Town

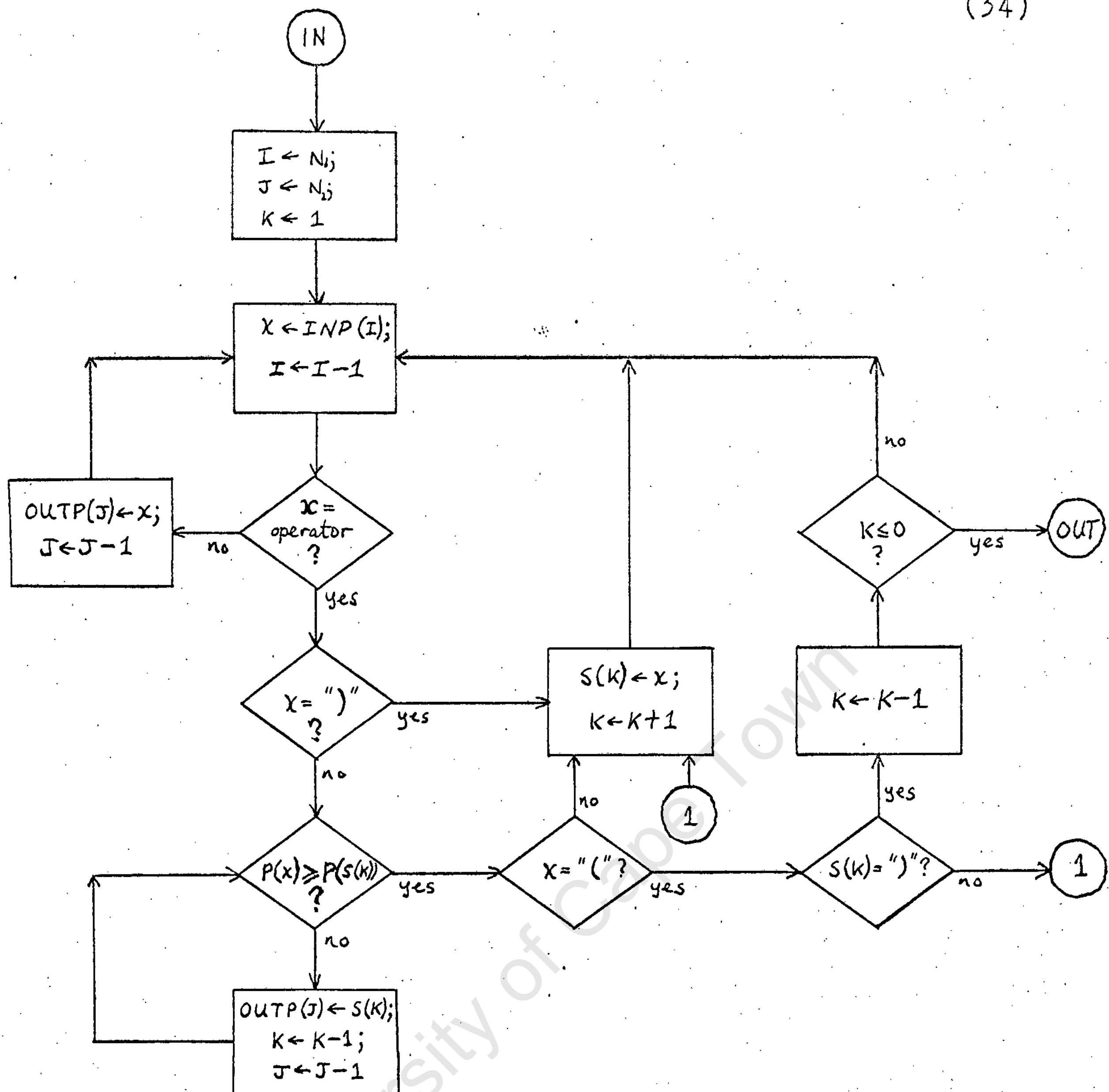


University of Cape Town

Fig. 7. Conversion of Infix to Polish Suffix Ordering.

INP is the input array holding the operators and operands of an infix ordered expression, OUTP the output Polish suffix ordered array. S is a stack to hold operators pushed down on it temporarily. The input expression is assumed to be bounded by an outermost parenthesis pair. P is an operator-precedence array in which the precedences are as follows:

x:	+	−	X	/	exponentiation	"unary" functions	(	)
$P(x)$:	3	4	5	6	7	8	1	2



University of Cape Town

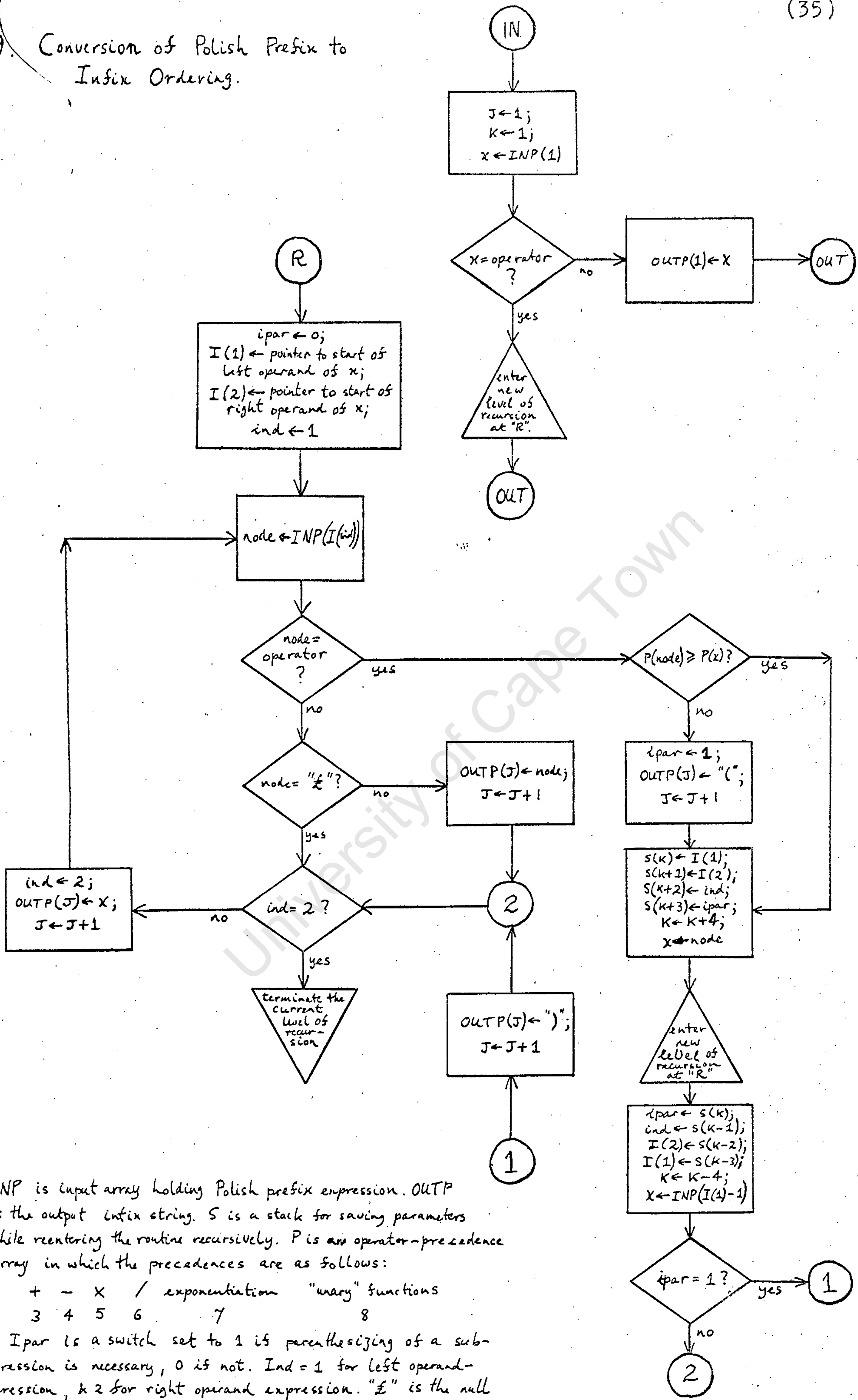
Fig 8. Conversion of Infix to Polish Prefix Ordering.

INP is the input array holding the operators and operands of an infix ordered expression, OUTP the output Polish Prefix ordered array. S is a temporary pushdown stack for operators. N_1 is the number of operators & operands in the input expression. N_2 is the length of the output expression which is equal to N_1 minus the number of parentheses in the input expression. The input expression is assumed bounded by an outermost parenthesis pair. P is an operator-precedence array in which the precedences are as follows:

x:	+	−	X	/	exponentiation	"unary functions"	(	)
P(x):	3	4	5	6	7	8	2	1

This algorithm is the same as that of Fig 7, except that the input expression is scanned, and the output expression built up, in the opposite direction to that algorithm, and the roles of left and right

.9 Conversion of Polish Prefix to Infix Ordering.



INP is input array holding Polish prefix expression. OUTP is the output infix string. S is a stack for saving parameters while reentering the routine recursively. P is an operator-precedence array in which the precedences are as follows:

x:	+	−	×	/	exponentiation	"unary" functions
P(x):	3	4	5	6	7	8

Ipar is a switch set to 1 if parenthesizing of a sub-expression is necessary, 0 if not. Ind = 1 for left operand-expression, & 2 for right operand expression. "£" is the null

and binary operators (+, -, X, / and exponentiation) as well as the natural and common logarithm functions, the trigonometric and hyperbolic functions and all their inverse functions. Functions of more than one argument are not representable in this implementation for binary operators. To cope with the derivative of the arcsecant amongst other discontinuous functions, the operator MOD, representing the absolute value of an operand, is included in the system, but any attempt to differentiate an expression which includes MOD results in error termination. The system must exclude discontinuous functions where they cannot be conveniently and consistently represented in a notational sense.

The expressions in DERIV are represented as symmetrical, linear SLIP lists, each node containing either a single operand or an operator in its datum plus necessary auxiliary information in the other fields of each node. Sublists are used but not in the usual SLIP sense (see Chap. 2). They are used for representing subexpressions of an expression, an artifice for reducing storage and time requirements in DERIV. For instance, the expression

$$A = \cos^2 x + \cos^4 x + \cos^6 x \quad .$$

is more economically represented by computer as two expressions:

$$Y = \cos x \qquad \text{and} \qquad A = Y^2 + Y^4 + Y^6$$

Description lists (see 2.3.2.5) have been used in DERIV to keep count of which derivative of an expression is with respect to which variable. Expressions may be deleted by the system if no longer accessible or by a user program if not required any more. One list common to all expressions in a DERIV program was required to correlate the addresses and internal alphabetic names (henceforth called tags) of all list-expressions currently in existence in the program. Tags are necessary because of the need to identify subexpressions (already in existence in the DERIV system) referenced by a new expression which is read into DERIV as a data card.

One SLIP feature not used to advantage is the reader and sequencer. These are unnecessary since DERIV lists are always linear and it is more convenient to use pointers in the expressions. Where subexpressions are used, they should be differentiated before the expression referencing them. The tags of the derivatives of the subexpressions then replace references to the

University of Cape Town

subexpressions in the derivative of the main expression where necessary. Thus the need for a reader traversing a list structure and differentiating as it goes along is precluded. This technique would be less economical time- and storage-wise than the method described before. Subexpression tags on a list are regarded as special operand nodes.

The public lists of SLIP are used to advantage in DERIV. Not only are they necessary for the recursive routines of the package, but W(100) is used as the tag-list address "collator list", instead of the DL mechanism. W(99) is used for saving variables in the recursive routines, and as a stack for the conversion of lists from infix to Polish notation. Thus only the first 98 public lists are available to a user program.

Higher order derivatives are easily obtained, as the derivative function, ALGDIF (see 3.3), delivers as its value the list-name of the derivative formed. Thus function calls may be nested. It was necessary to restrict the routine to only one differentiation at a time since a variable number of arguments is not allowed in FORTRAN V. Similarly, partial differentiation with respect to various variables is allowed, since variables and alphabetic constants are represented internally in the same way. For instance, if the second derivative of expression L is required, first with respect to X and then Y, the following function call may be used:

```
DLYX=ALGDIF(ALGDIF(L,'X','DLX'),'Y','DLYX')
```

Since a reference to a subexpression is represented by putting the tag of the subexpression in the relevant node, differentiation with respect to an expression is possible, provided that the subexpression be defined first. The subexpression reference is represented almost exactly as are variables and constants.

In figure 10 the layout of nodes in DERIV is indicated in detail. Note that the basic layout has not changed from U-SLIP, and only fields unused by the list processor are used by DERIV.

The description list of an expression is pointed to by the address in the DL field of the list header. A DL is only created if the expression is differentiated at least once. The attribute field of a DL node contains the variable of differentiation and the value field the address of the header of the derivative list with respect to this variable. In this way the whole his-

University of Cape Town

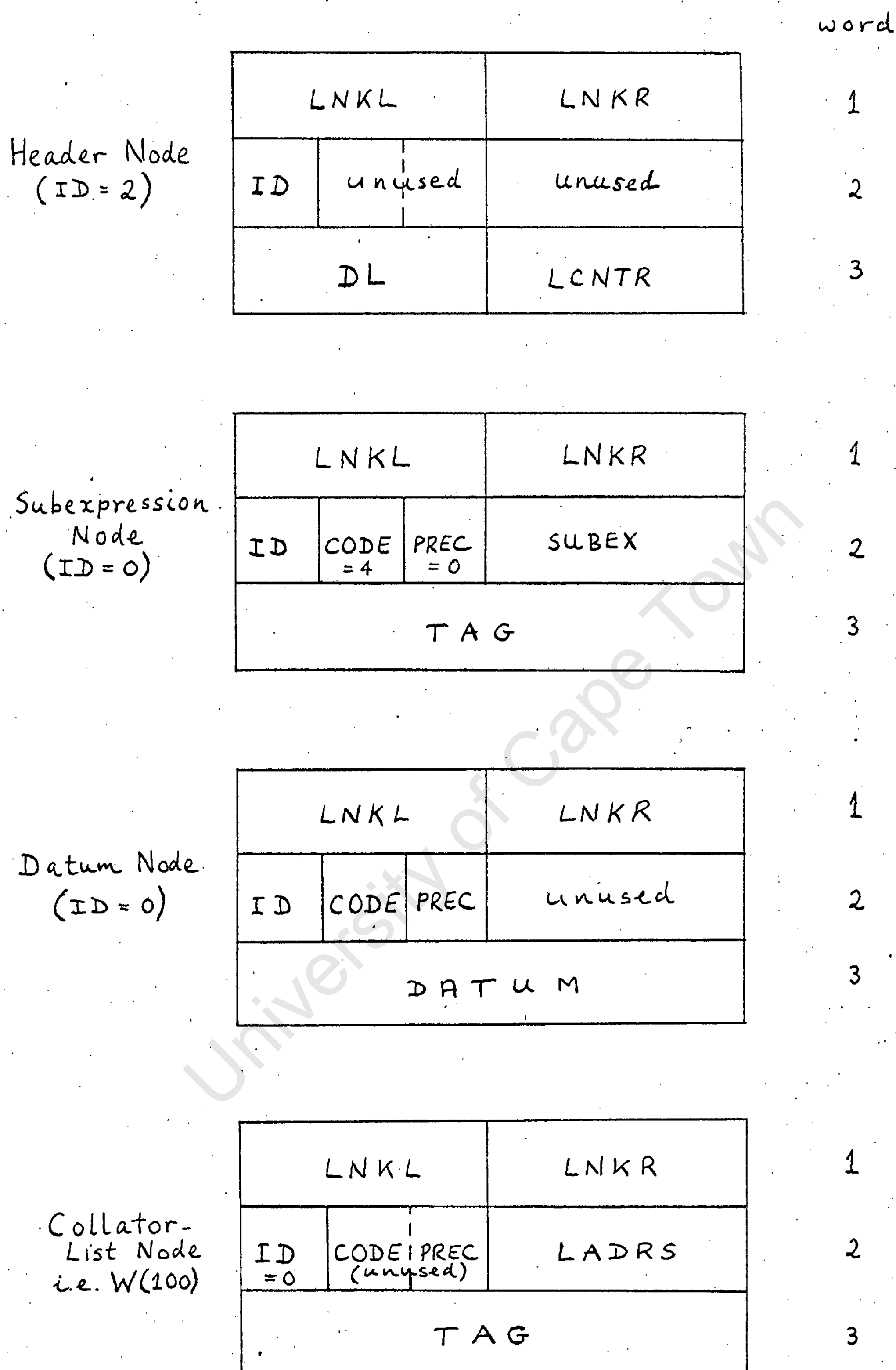


fig 10. Layout of DERIV node types

tory of differentiation of an expression is known. The collator list and DL's attached to expressions perform all the "book-keeping" that DERIV requires.

The layouts of subexpression and datum nodes are similar. In the former ID is 0, CODE is 4 (denoting an alphabetic variable), PREC is 0 and TAG has the tag associated with the subexpression referenced. SUBEX is the pointer to the header node of the subexpression (cf. description of a datum cell).

In a datum cell it is necessary to distinguish between operators and operands of four types: real and integer numbers, reserved words (e.g. PI, E, and I as used for the corresponding mathematical constants) and alphanumeric variables. The distinction is maintained by the value in the CODE field which is 0, 1, 2, 3 and 4 for operators, integers, real numbers, reserved symbols and variables respectively. User constants and variables are represented in the same way since only the datum with respect to which differentiation takes place is a variable, all the other alphanumeric variables being treated as constants. The PREC field is 0 except when CODE=0, in which case it contains the precedence ordering of the operator in the relevant datum field. Table 1 gives a list of the operators allowed in the differentiator, their mnemonics and the precedences assigned them. Note that exponentiation is denoted by "!" in DERIV. The datum field contains an integer if CODE=1, a real number if CODE=2 and in all other cases at most 6 fielddata characters, representing the datum alphanumerically. If the name has fewer than 6 characters, it is left justified and the remaining characters represented as blanks.

A collator list node uses only the LADRS and TAG fields, the former field storing the pointer to the header node of the expression identified by the tag stored in the latter field. As explained above, this is necessary for convenient I/O where expressions refer to subexpressions.

Note: In table 1 it can be seen that parentheses are treated as operators by DERIV. These only occur in the infix lists used for I/O. Three forms of parenthesis (round, square and broken) are allowed for convenience.

Further discussion is delayed until the routines have been described.

University of Cape Town

Operator	DERIV Mnemonic	Precedence level
* (,[,<	(	2
*),],>	)	1
+	+	3
-	-	4
X	*	5
division	/	6
exponentiation	!	7
natural logarithm	LN	8
common logarithm	LOG	8
sine	SIN	8
cosine	COS	8
tangent	TAN	8
cotangent	COT	8
secant	SEC	8
cosecant	CSC	8
arcsine	ASIN	8
arccosine	ACOS	8
arctangent	ATAN	8
arccotangent	ACOT	8
arcsecant	ASEC	8
arccosecant	ACSC	8
absolute value	MOD	8
hyperbolic sine	SINH	8
h. cosine	COSH	8
h. tangent	TANH	8
h. cotangent	COTH	8
h. secant	SECH	8
h. cosecant	CSCH	8
inverse h. sine	ASINH	8
inv. h. cosine	ACOSH	8
inv. h. tangent	ATANH	8
inv. h. secant	ASECH	8
inv. h. cosecant	ACSCH	8
inv. h. cotangent	ACOTH	8

Table 1. The Operators Allowed in the DERIV system.
*(See end of section 3.2).

University of Cape Town

Expression	Derivative of Expression	Polish Prefix Ordering
Constant	0	0
Variable	1	1
A + B	A' + B'	+ A' B'
+ A	+ A'	A'
A - B	A' - B'	- A' B'
- A	- A'	- £ A'
A * B	A' * B + B' * A	+ * A' B * B' A
A / B	(A' * B - B' * A) / B ! 2	/ - * A' B * B' A ! B 2
A ! B	B * A ! (B - 1) * A' + A ! B * LN A * B'	+ * B * ! A - B 1 A' * ! A B * LN £ A B'
LN A	(1 / A) * A'	* / 1 A A'
LOG A	LOG E * (1 / A) * A'	* LOG £ E * / 1 A A'
SIN A	COS A * A'	* COS £ A A'
COS A	- SIN A * A'	* - £ SIN £ A A'
TAN A	(SEC A) ! 2 * A'	* ! SEC £ A 2 A'
COT A	- (CSC A) ! 2 * A'	* - £ ! CSC £ A 2 A'
SEC A	SEC A * TAN A * A'	* SEC £ A * TAN £ A A'
CSC A	- CSC A * COT A * A'	- £ CSC £ A * COT £ A A'
ASIN A	(1 / (1 - A ! 2) ! ½) * A'	* / 1 ! - 1 ! A 2 ½ A'
ACOS A	- (1 / (1 - A ! 2) ! ½) * A'	- £ * / 1 ! - 1 ! A 2 ½ A'
ATAN A	1 / (1 + A ! 2) * A'	* / 1 + 1 ! A 2 A'
ACOT A	- 1 / (1 + A ! 2) * A'	- £ * / 1 + 1 ! A 2 A'
ASEC A	1 / (MOD A * (A ! 2 - 1) ! ½ * A'	* / 1 * MOD £ A ! - ! A 2 1 ½ A'
ACSC A	- 1 / (MOD A * (A ! 2 - 1) ! ½ * A'	* / 1 * MOD £ A ! - ! A 2 1 ½ A'

Table 2. Differentiation Rules as Used in DERIV. (See also Table 1)
(continued overleaf)

A & B represent expressions, A' & B' their derivatives. £ represents the null operand used to convert unary operators to binary.

University of Cape Town

Expression	Derivative of Expression	Polish Prefix Ordering
SINH A	COSH A * A'	* COSH £ A A'
COSH A	SINH A * A'	* SINH £ A A'
TANH A	SECH A ! 2 * A'	* ! SECH £ A 2 A'
COTH A	- CSCH A ! 2 * A'	- £ * ! CSCH £ A 2 A'
SECH A	- SECH A * TANH A * A'	- £ * SECH £ A * TANH £ A A'
CSCH A	- CSCH A * COTH A * A'	- £ * CSCH £ A * COTH £ A A'
ASINH A	1 / (A ! 2 + 1) ! ½ * A'	* / 1 ! + ! A 2 1 ½ A'
ACOSH A	+ 1 / (A ! 2 - 1) ! ½ * A'	+ £ * / 1 ! - ! A 2 1 ½ A'
ATANH A	1 / (1 - A ! 2) * A'	* / 1 - 1 ! A 2 A'
ACOTH A	- 1 / (A ! 2 - 1) * A'	- £ * / 1 - ! A 2 1 A'
ASECH A	- 1 / A * (1 - A ! 2) ! ½ * A'	- £ * / 1 * A ! - 1 ! A 2 ½ A'
ACSCH A	- 1 / (MOD A * (1 + A ! 2) ! ½) * A'	- £ * / 1 * MOD £ A ! + 1 ! A 2 ½ A'

Table 2. Differentiation Rules as Used in DERIV. (continued)

(See note on previous page).

University of Cape Town

3.3 The Routines in DERIV.

These routines fall into two groups, the differentiation routine and its ancillary functions, and the routines used for input/output. There are FORTRAN listings of the routines in Appendix B. Flowcharts of some routines are also provided (see Appendix C).

3.3.1 Differentiation.

ALGDIF(L,VAR,TAG) List L is differentiated with respect to variable VAR (in Hollerith form) and the tag TAG (also in Hollerith form) is attached to the derivative list, the name of which is delivered as the value of the function. ALGDIF is discussed in greater detail in 3.3.2.

Note: It is necessary in DERIV to distinguish carefully between a list-name (expression-name) and a tag. The former is the name of the list in SLIP name-format (see Chap. 2) whereas the tag is for alphabetic identification of the expression-lists of DERIV.

NDIF(NU,ND,VAR) This function is used by ALGDIF. NU contains the address of a node on the input list, i.e. it is a pointer to such a node. ND is a pointer to the bottom of the derivative list which is being built up. VAR is the variable of differentiation as in ALGDIF. The latter calls NDIF when the operand to be differentiated at any stage is not an expression, but of unit length. If the operand in the datum field of NU is VAR, a new cell is taken from AVAIL and added to the right of the cell pointed to by ND on the derivative list. It has a datum 1. If the datum of NU is not VAR, 0 is delivered in the datum field of the new derivative node, unless one of two other possibilities occurs. If the datum is the null operand, the derivative node contains the null operand as well. If NU contains a pointer to a subexpression (SUBEX field non-zero) the following occurs.

University of Cape Town

For simplicity consider the expression E, its derivative being DE. E references subexpression S which should already have been differentiated with respect to VAR to yield expression DS. Since NU (on E) points to S, the DL of list S is searched for the pointer to DS. If it is not found, a warning is printed and a datum of 0 returned in the derivative node. Otherwise the pointer to DS is compared with those of the collator list in search for the tag of DS. If it is not found on W(100) there is an error in the system and walkback occurs. Otherwise the tag of DS is placed in the new cell to the right of ND in DE and the SUBEX field of the new cell contains a pointer to DS.

value: The address of the cell taken from AVAIL for the new node on DE.

NCODE(A) — value: The CODE field of the cell whose address is in A.

NDAT(A) — value: The DATUM field of the cell whose address is in A.

NPREC(A) — value: The PREC field of the cell whose address is in A.

NEWNOD(N1,N2,N3,A) — A new cell to the right of the node whose address is in A is created from AVAIL, with N1 to N3 in its CODE, PREC and DATUM fields respectively. If N1 or N2 are -1, these fields remain as they were in the cell when it was taken from AVAIL.

value: Address of the new cell.

HANDLE(NB,NE,L) — This routine, given the address of the starting node of some operand within the list (NB), returns the address of the last node in the operand (NE) and the total number of nodes in the operand (L). The list on which this routine is used must be in Polish prefix notation, where an operator always precedes its two operands. If

University of Cape Town

the operator beginning at NB starts with an operand cell, it is of unit length and NE=NB. If the operand is an expression the first node of the expression must contain an operator. This in turn has two operands, possibly themselves expressions. HANDLE scans from the left starting at NB. Initially it requires one operand (for the operator whose operand is to be delineated by the routine). For every operator encountered, an additional operand is required. The scan continues until the number of operators encountered is 1 less than the number of operand cells. In this case the full operand is delineated by NB and NE, and L contains the number of nodes scanned.

ISYNTX(LST) The expression of list-name LST is tested for correct Polish prefix ordering. LST is scanned rightwards node by node, and at any stage the number of operators already encountered must be at least one less than the number of operand nodes, or the expression is misformed. When all the nodes have been scanned, the number of operators must be exactly one less than the number of operand nodes.

value: 0 if the expression LST is syntactically correct, -1 otherwise.

IOPCPY(NU,L,ND) The operand starting at the node whose address is in NU and whose length is L is copied to the right of the node whose address is in ND. Cells are taken from AVAIL for this purpose.

value: Address of the last cell inserted to the right of ND.

IEQOPD(IP,LS) Two operands whose starting addresses are stored in IP (an array dimensioned 2) and whose lengths are stored in array LS (also dimensioned 2) are compared. If they have

identical content (i.e. all the DERIV fields of corresponding nodes are identical) and the nodes of each operand occur in the same order and the operands are of the same length, then the value of this function is 0, otherwise -1.

IRALEX(L) This is similar to IRALST (see Chap. 2) except that before the list whose name is L is restored to AVAIL, the collator list is searched for the tag of list L, which is then removed as well. If a DL is attached to L, then this is deleted as well.
value: LCNTR field of list L.

Note: Since the tag of the list is removed, this list cannot be referenced any more, as lists usually can be when restored to AVAIL with a LCNTR=1 or more. Lists created for input of expressions by READEX (see 3.3.3) are thus created with an initial LCNTR of 1 to prevent erasure by the system. IRALEX should only be used when an expression can no longer be referenced by a user program, either directly or indirectly, as a subexpression of any other lists. IRALEX is useful for incrementing AVAIL when some large expression to be differentiated makes heavy demands for new nodes and there is a danger of AVAIL being exhausted.

3.3.2 The Differentiation Function ALGDIF.

3.3.2.1 Description.

On entry into ALGDIF a list is created to hold the derivative. The tag intended for this list, the third argument of the function, is placed on the collator list together with the header address of the derivative list, unless the tag is already on W(100), in which case the LADRS field of the node of W(100) of which the tag is in the TAG field is overwritten with the header address of the newly created derivative list. This address is also placed on the DL of the list, as the value of the attribute VAR (see 3.3.1). If no DL existed, one is created by the system. If VAR already occurred on the DL, the corresponding value is overwritten and replaced by the new header address. The list L input to ALGDIF is then scanned from the left. If the node is an operand, NDIF (see 3.3.1) takes care of its derivative and the process is finished since the input list

consisted of only one node. Otherwise the routine is fully recursive. For every operator encountered, one of the entry points for the differentiation rules (represented by modules in ALGDIF) is proceeded to. If an operand is in turn to be differentiated within one of the rules, the necessary parameters are saved by being pushed down on W(99), used as a stack. The beginning of the routine is then recursively reëntered. Operands of single length are dealt with by NDIF. At the end of all the modules (including that for single operands), the then current level of recursion is terminated and control returned to the outer rule which had been interrupted for deeper level differentiation. The derivative list is built up from the left in Polish prefix notation, according to the differentiation rules (see Table 2).

3.3.2.2 Simplification.

The necessity for simplification of some form or another is vital to any algebraic computer package.

In DERIV simplification could be introduced in two forms. One possibility is to scan expressions before and after differentiation to eliminate redundancies and to attempt factorization. This technique has formidable advantages in canonical form representation and the performance of arithmetic on a derivative list to shorten it.

The second form involves various contingencies in the differentiation rules themselves to prevent redundant nodes being formed in the derivative list. The ideal approach would have been to combine both forms in DERIV but since the first approach would consume large amounts of core and time because of the intricate programming involved, and many of the problems have not been solved in general to date, only contingency simplification has been introduced in ALGDIF. An added disadvantage of the first type on its own is that a derivative would grow to its full unsimplified length before the introduction of simplification.

Zero equivalence recognition occurs only in a residual form for some of the basic operators. Certainly DERIV is unable to recognise that since $\sin^2(x + y) + \cos^2(x + y) = 1$, its derivative is zero. However if any of the unary functions have operands of single length which are not the variable of differentiation, zero is immediately produced instead of the full

University of Cape Town

expansion of the derivative rule.

Fairly complex simplification is carried out for the binary operators. Flowcharts of this simplification are shown in figures 11 through 15 below. When the operands of an operator have at any stage in the procedure been determined, the category (ICAT) into which each operand falls is ascertained: 1 if it is of unit length and its datum not the variable of differentiation, 2 if it is of length 1 and this variable and ICAT=3 if the operand is an expression including further operators. This assists in determining what action to take in the differentiation modules, for instance if in the "+" rule both operands are of category 2 and the variable is X, the output to the derivative list is "* 2 X".

No attempt is made in DERIV to represent all expressions in some canonical form, except at a very basic level in some modules, as can be seen from the example above. For example the "-" module uses IEQOPD to determine if the two operands, both of category 3, are exactly equal, in which case time and storage can be saved by the derivative being 0. However, if the two operands are exactly equal except that they are rearranged slightly, e.g. operands within them interchanged, the IEQOPD test will fail and the full differentiation procedure has to be undergone for that phase of the process.

Because the contingency simplification is not very sophisticated, the derivatives produced remain to a large extent unsimplified and are not always aesthetically pleasing. Nevertheless it succeeds in reducing the core and time requirements of a DERIV program considerably. The implementation of DERIV at UCT has fairly wide capabilities because a fast-access memory of 65K words is available (see 3.6 & 3.7 below).

3.3.3 Input/Output Routines in DERIV.

The purpose of such routines is to simplify matters for a user since I/O may be performed using infix ordered expressions. Though the notation used is similar to that usually employed in mathematics, the mnemonics for some of the allowable functions necessarily differ from mathematical practice (see Table 1). Furthermore, what looks like an equation or function that is input is in fact an assignment of the expression on the right hand side of "=" to the list-tag on the left. Round, broken and square parentheses are provided for the convenience of the user,

University of Cape Town

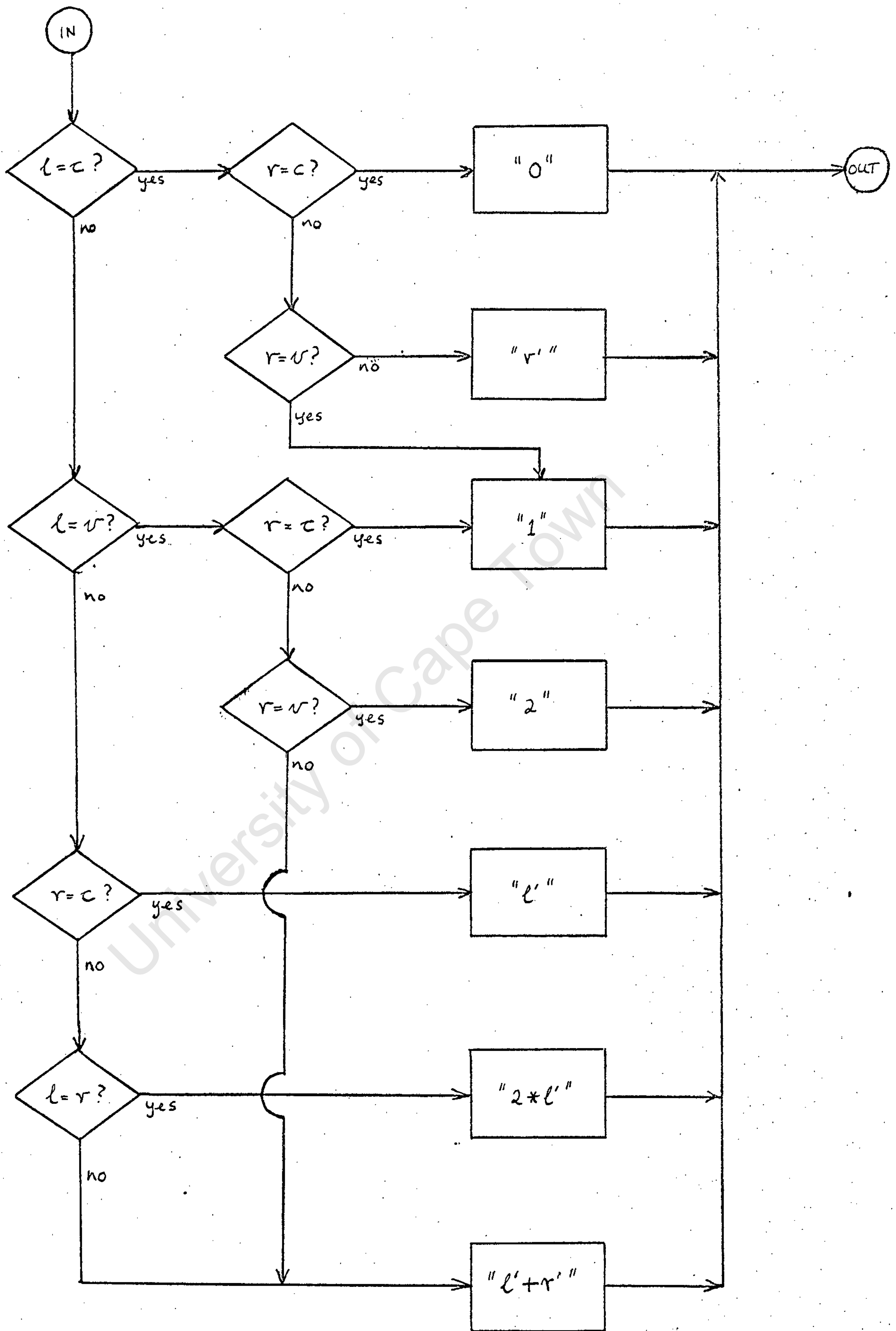


fig. 11 . Simplification of the Derivative of "l + r"
(c = constant ; v = variable)

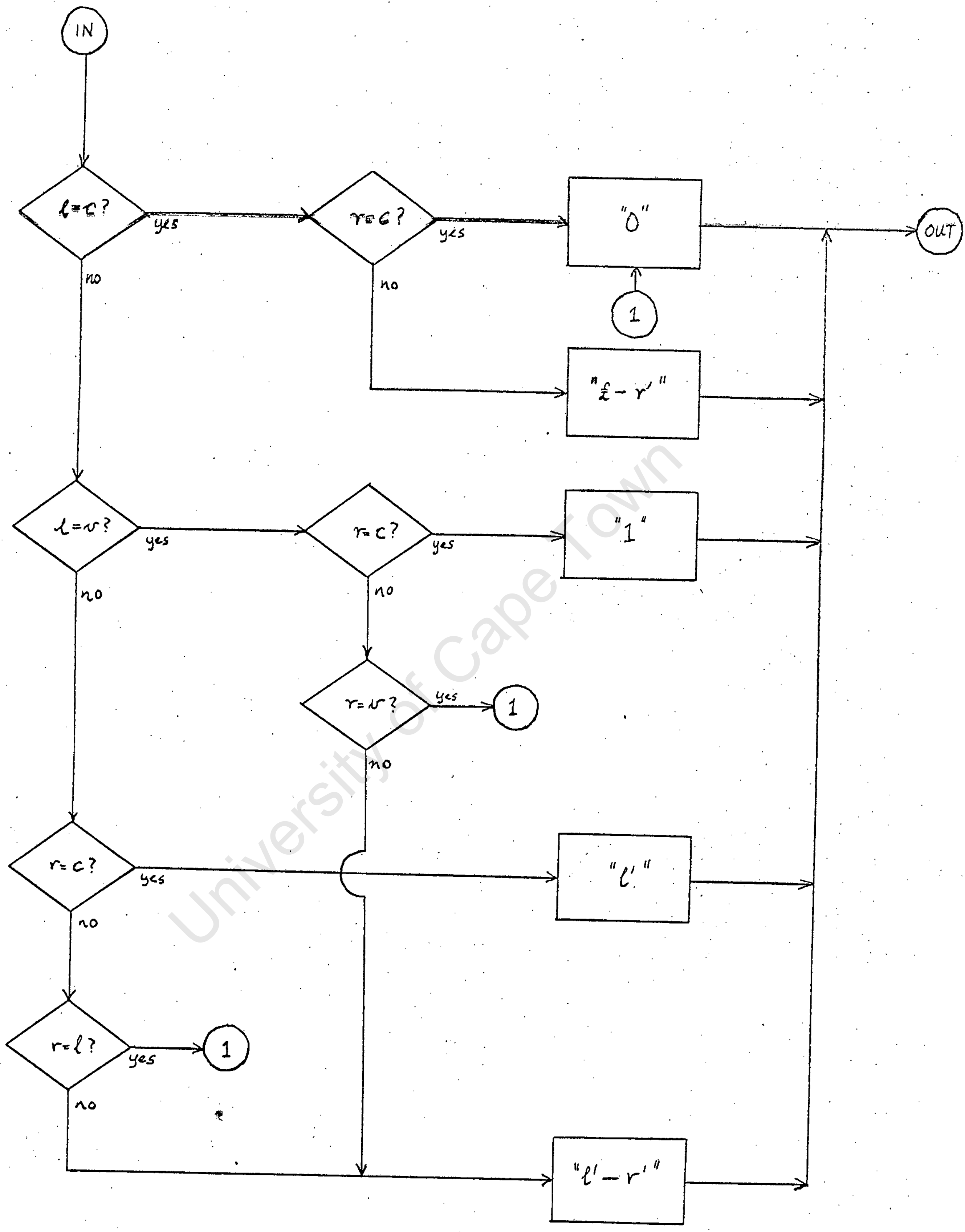
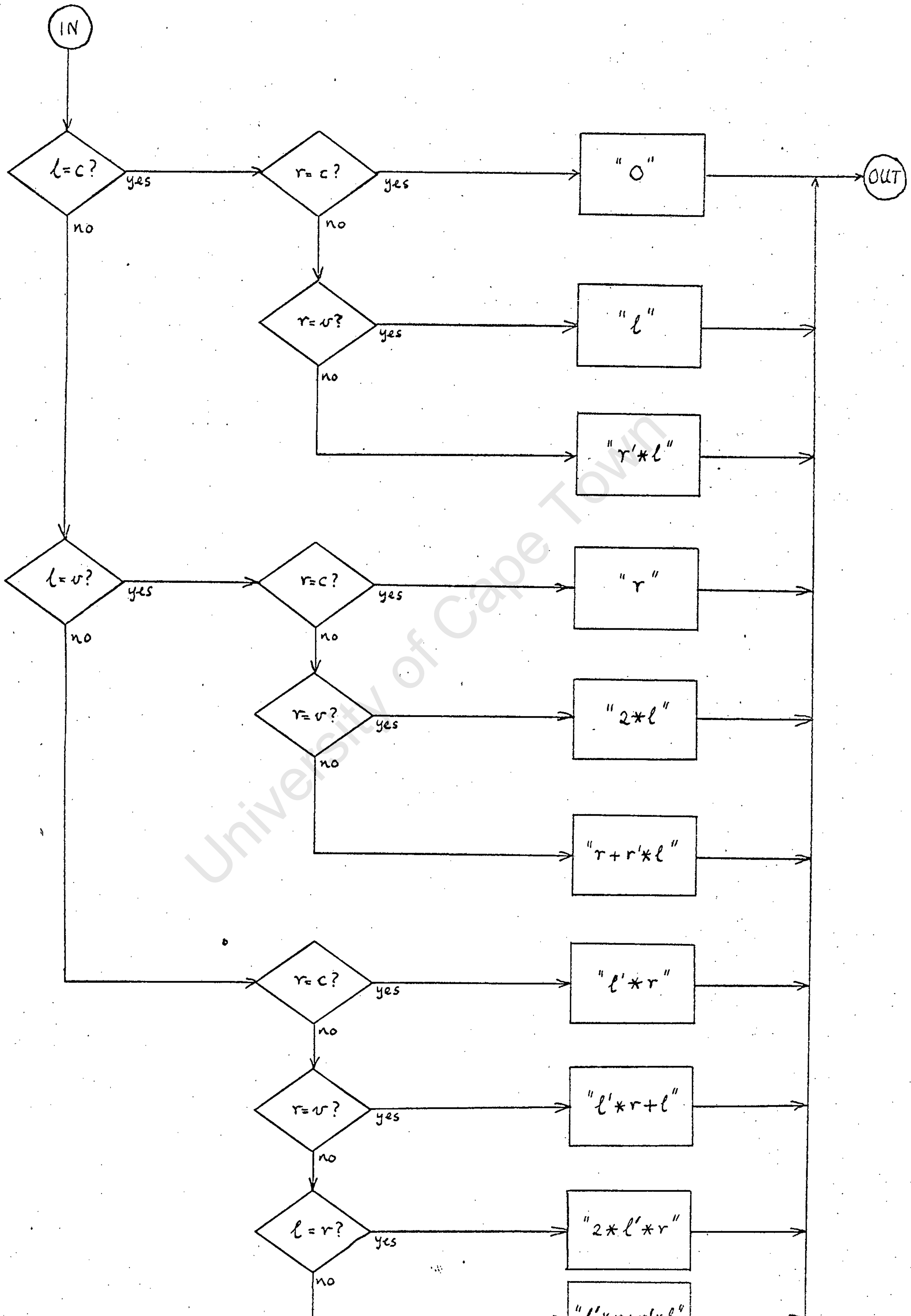


fig 12. Simplification of the Derivative of "l − r".
(c = constant; v = variable)
(l & r are expressions, l' & r' their derivatives).

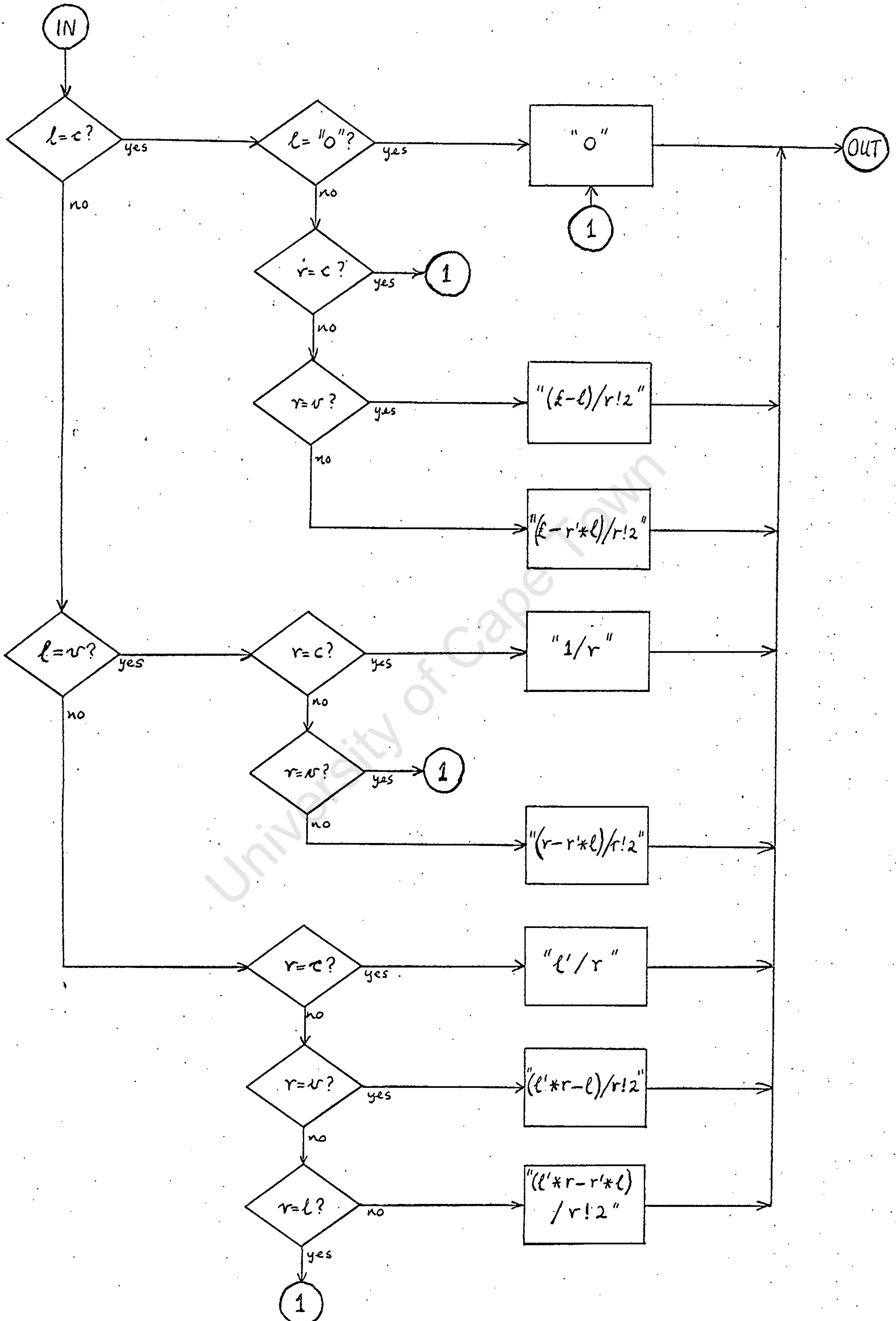
University of Cape Town

fig 13. Simplification of the Derivative of "l*r".
(c = constant; v = variable)
(l & r are expressions; l' & r' their derivatives).



University of Cape Town

Fig 14. Simplification of the Derivative of l/r.
(c = constant ; v = variable).
(l & r are expressions, l' & r' their derivatives).



University of Cape Town

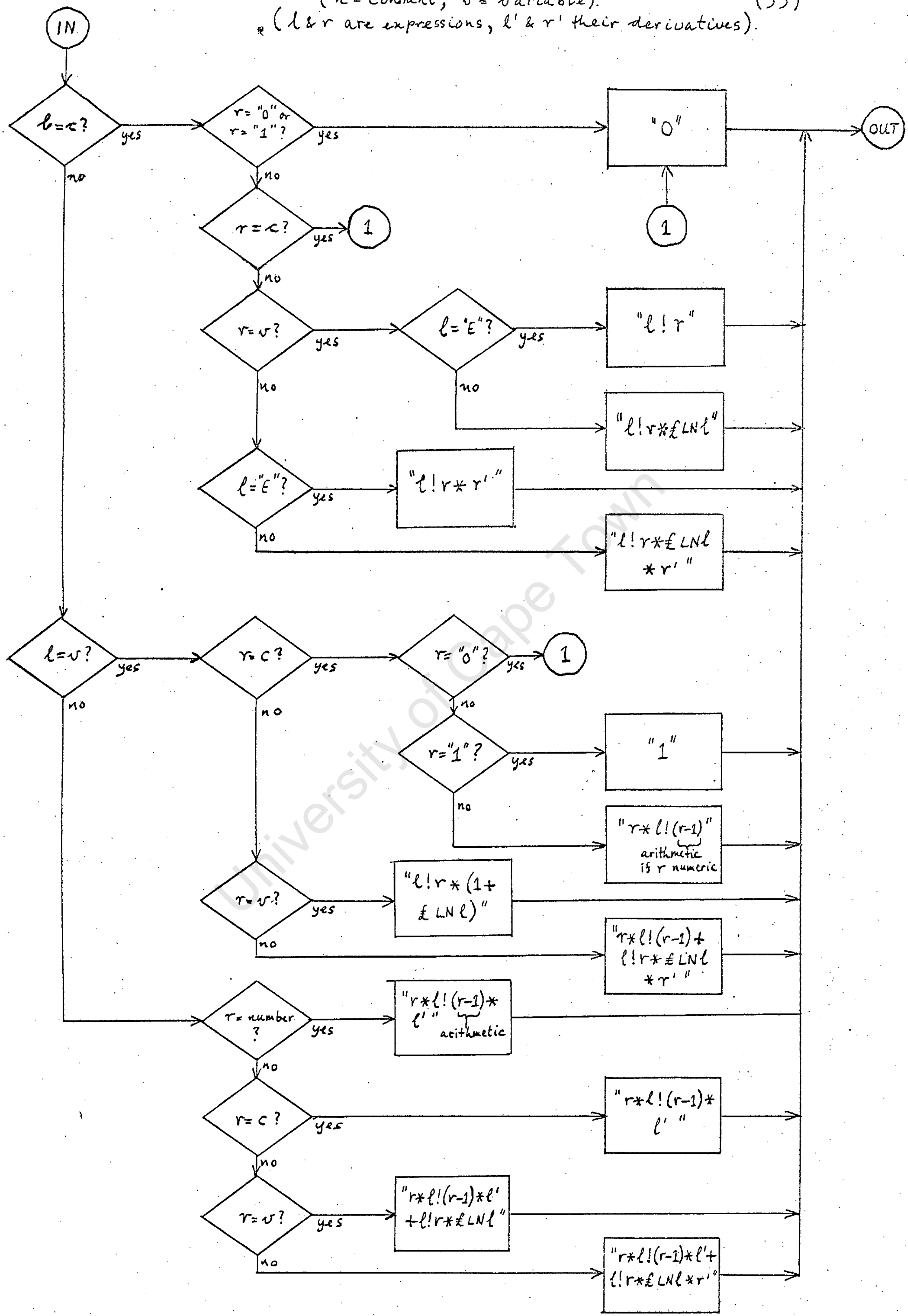


fig 15. Simplification of the Derivative of l!r
(c = constant; v = variable).
(l & r are expressions, l' & r' their derivatives).

University of Cape Town

though their internal representation is the same. Since square brackets do not occur in the EBCDIC input code, their use is limited to input devices using ASCII or Hollerith codes.

Expressions to be output are also printed in semi-mathematical notation, parenthesizing of subexpressions to any depth being alternately broken and round in an attempt to simplify reading the expressions.

References to subexpressions separately input are by their tags, preceded by a dollar sign. This is applicable to input and output.

The I/O routines are as follows:

PREORD(LIN,LOUT) This subroutine is used by READEX (see below) to convert an infix expression represented by list LIN to a Polish prefix ordered one in a list LOUT created by the routine. The algorithm has been adapted from that shown in the flowchart of fig. 8 (see 3.1), except that instead of contiguous addresses, lists are used. LIN is first bounded at the top and bottom by opening and closing parenthesis nodes respectively and is then scanned from the <u>right</u> leftwards. Operator nodes (including closing parentheses) are pushed onto a stack while operand nodes are inserted at the top of LOUT (i.e. the prefix list is built up from the <u>right</u>). Before any operator node is pushed onto the temporary stack, W(99), any operator nodes of lower or equal precedence (see note below) to it at the top of W(99) are popped up and inserted at the front of LOUT. Right parentheses are not popped up onto LOUT and the only way of ridding W(99) of them is by encountering a left parenthesis. Both are then discarded and the process continued. Termination of the process occurs when the outermost parenthesis pair, inserted into the list, meet. LIN is not deleted and only the parenthesis nodes inserted initially

at its top and bottom are removed before return from the routine.

Note: Because the precedences of operators in the algorithm have been assigned uniquely except to the unary functions (see fig. 8 and Table 1), the case of equal precedences in popping up W(99) only arises where for instance, the input expression was $x +_1 y +_2 z$ or $x -_1 y -_2 z$. As stated in 3.1, there are two possible Polish representations for such an expression. In the case of prefix notation, the first expression above yields either $+_2 +_1 x\ y\ z$ or $+_1 x +_2 y\ z$. Including the equality case in popping up the stack leads to the first form and excluding it to the second form, the opposite being true in the reverse Polish algorithm. In DERIV the first form is more convenient since the routine for conversion of prefix expressions back to infix notation (INFORD) in effect scans rightwards rather than leftwards because of its recursive nature (see below). If the second form of prefix representation of an expression were used, INFORD would parenthesize the infix expressions it produces incorrectly.

INFORD(LIN,LOUT) A list LIN representing an expression in Polish prefix ordering is converted to a list LOUT, created in the subroutine, which contains an infix ordered expression with any necessary parenthesis nodes inserted. The algorithm used is that of figure 9 (see 3.1), except that lists rather than contiguous addresses are manipulated here. The algorithm is recursive since the only other method of conversion was found to be difficult to program in general. This subroutine always inserts new nodes on LOUT at the back, thus building up the infix expression from left to right. The algorithm also recognises when it is necessary to insert parentheses around some operand on the output list, thus avoiding much extraneous parenthesizing. The structure of the Polish prefix list is such that if the current operator has a precedence less than that of the outer operator of which this operator (together with its

University of Cape Town

operands) is an operand, parenthesizing is required for this operand. For example consider * z + x y, prefix for z * (x + y). The left operand of * is z, the right operand + x y. Because * has higher precedence than +, it is necessary in the infix expression that the right operand of * be parenthesized. Since INFORD is recursive, it is wasteful of storage(see 3.6).

READEX(IPRT) A mathematical expression is read off the first 78 columns of a data card. If the expression has to be continued to a further data card, a non-blank character should appear in column 80. Any number of data cards may thus be used to represent an expression. The first 78 columns of each card are read into a Hollerith buffer of 13 words of 6 characters each. The buffer is then scanned character by character from the left, until either the end of the card is reached (in which case if the continuation character is set another card is read and if not the scan is finished) or the terminator character "@" is encountered, in which case the expression has been completely scanned and anything else on the card including a continuation character is ignored. The @ was included in READEX to eliminate the necessity of scanning redundant blanks following the end of an expression. If the argument IPRT is literal 9 each card image read to represent the expression being input is printed in the same format. If IPRT is anything else, the card images are not printed. Variable names (including those of tags of subexpressions referenced and that of the expression being input) may contain up to 6 characters, either alphabetic or numeric, as long as the first character is one of

University of Cape Town

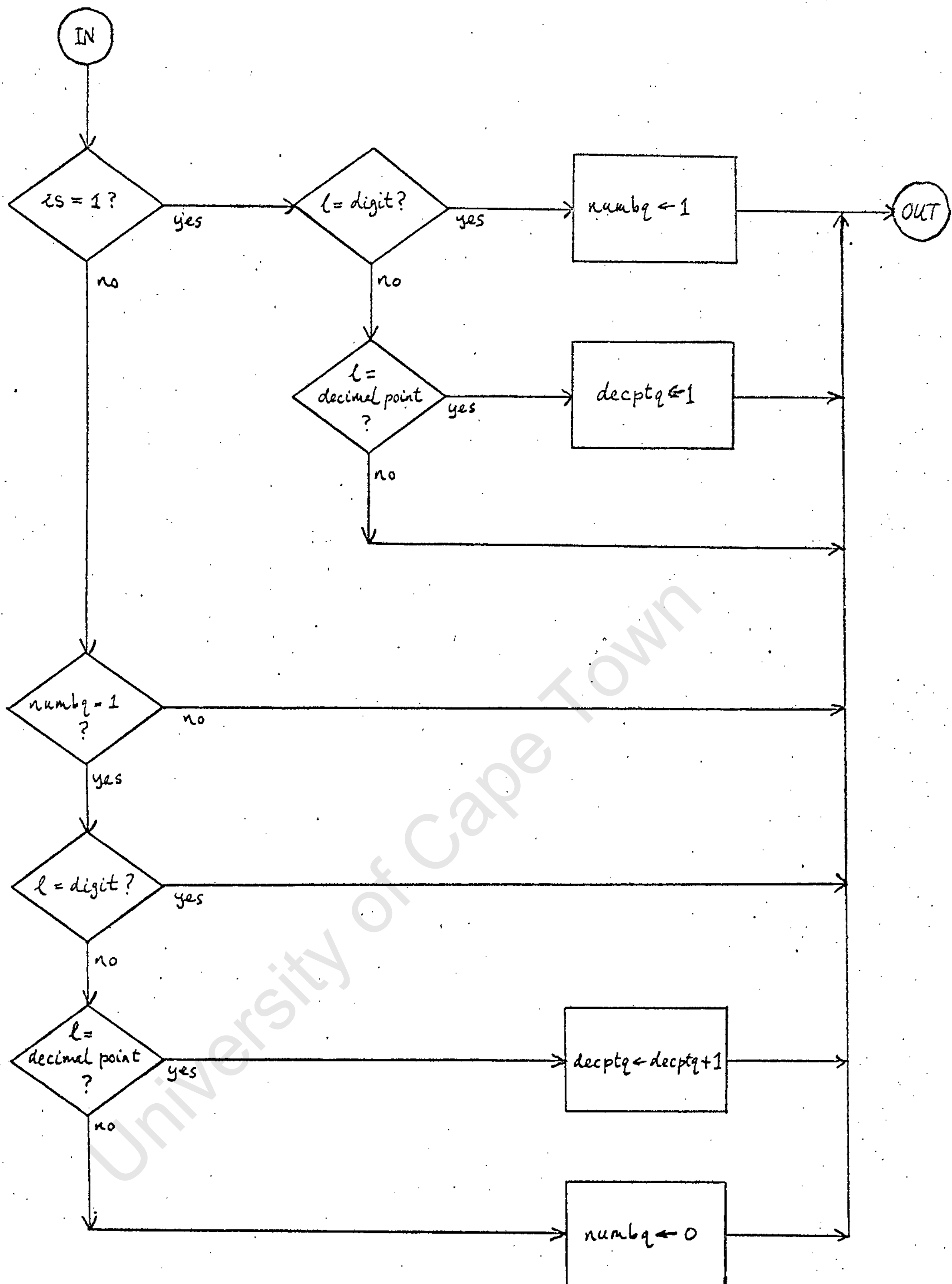
the former. Variables of length more than 6 characters are truncated to 6 and a warning message is printed. This is valuable in case a mistake was made by the user and some operator omitted from the expression (see 3.4). Integers and real numbers are recognised and must have the same syntax as in FORTRAN. In other words if a number contains a decimal point it is represented as real, otherwise as an integer. Numbers may have up to 18 characters in them. If this number is exceeded an error message results. Figure 16 shows a flowchart indicating how variables are recognised. Two examples of typical input expressions follow:

```
FNX=A * X!2 + B*X +C@
T=2./(ØONE - 3.0*VAR! 2)-SIN<.2*PI*ØZERO>.
```

Until an "=" is encountered, a variable (read in character by character) is regarded as the tag of the list being read in. When the "=" is read the tag is left justified and stored in a temporary variable. The scan then continues to the right of the "=". The delimiters of variables recognised by READEX are the first 7 entries in Table 1 (see 3.2), as well as a blank, Ø and @. When a delimiter is encountered any variable which has been built up in the temporary variable buffer (TVB) as the scan proceeded is interpreted and output to a temporary list which will eventually hold the infix expression. The interpretation phase is as follows:

If Ø had been encountered immediately before the variable, a subexpression node is created, referring to the list of which the variable is a tag. If no such tag is found on the collator list, an error termination results. If the variable is recognised as one of the system operators, an

University of Cape Town



University of Cape Town

Fig. 16. Recognition of Input Variables and Constants — Algorithm Used in READEX.

Numbq & decptq initially both zero. Algorithm entered for each letter of word being scanned. When delimiter encountered, word recognised as alphanumeric if final value of numbq is 0, as an integer if numbq = 1, decptq = 0 & as a real number if numbq = 1, decptq = 1. Decptq > 1 gives an error as does the combination decptq = 1, numbq = 0.

operator node is added to the infix list, preceded by a node with the null operand, "£", if the operator is unary. If it is recognised as a reserved constant (e.g. PI) then the appropriate node is created. If the variable is a number it is converted to the relevant internal representation (integer or real) and a node added to the list. In all other cases an ordinary variable node is created.

When the variable has been added to the infix list the delimiter is dealt with. Parentheses are added to the list as operators, operator-delimiters result in the creation of operator nodes and blanks result in immediate continuation of the scan. If an @ is encountered, the scan is terminated and anything still in the TVB is added to the infix list. A $ results in the setting of a switch to indicate a following tag, the TVB is emptied to the list and the following tag variable scanned. During the scan counters total up all the variables, operators and left and right parentheses found. Upon termination of the scan, there are syntactical checks that the left and right parentheses balance one another and that the number of operators in the list is exactly one greater than the number of operands. PREORD is then called to convert the infix list to Polish prefix ordering, after which the input list is deleted. The tag of the input expression is then compared with those already on the collator list. If it is there, the new list's header address replaces that in the relevant cell of W(100). Otherwise a new node is added to W(100).

value: The name of the Polish prefix list with the input expression.

University of Cape Town

PRNTEX(L,IDEL) The expression in list L is checked for Polish prefix ordering using ISYNTX (see above) and then converted to infix order and printed in "mathematical" notation. If IDEL is 9, L is retained by the system, otherwise it is deleted after conversion to infix ordering (using INFORD). The collator list is searched for the tag corresponding to the address of list L. If not found, an error message is printed and walkback invoked. Otherwise the tag and "=" are added as two operand nodes at the top of the infix list. This list is then read node by node and the datum of each node is added to the print buffer of 21 Hollerith words of 6 characters each (126 print positions). All unnecessary blanks in a datum are compressed out of the print image and extra blanks are only inserted where absolutely necessary. Null operands of unary operators are omitted from the print buffer. Numbers are translated into Hollerith characters before insertion in the print buffer. Nested parenthesis pairs are alternately broken and round to simplify reading the output expression. When the print buffer is full, it is printed, re-initialised to blanks and the node-scan continued. If a variable or operator is not able to fit onto the remainder of a line, the line is output with blanks in the vacant positions at the end of the print buffer. The datum is then inserted at the beginning of the fresh print buffer.

3.4 Design Philosophy of DERIV.

DERIV was designed with general use in mind. Provision has been made for extension of the system so as to differentiate additional functions. Furthermore, extensive error checking is undertaken within DERIV to prevent input errors.

Further differentiation rules may be added to ALGDIF with

University of Cape Town

ease, since not more than five statements in ALGDIF and READEX have to be changed. The operator table has been dimensioned to allow 40 functions. More reserved symbols may occur in the reserved word table as well. This requires two changes in the programming of the routines. Functions for more specialised purposes may be added to DERIV with ease since it is based on FORTRAN and SLIP.

The error checks of READEX are fairly comprehensive and should guard against misformed expressions entering the system. There are checks that illegal symbols are not input, that more than one decimal point does not occur in a number, that a variable or number is not too long for the temporary variable buffer, that the left and right parentheses balance in an expression and that operands or operators are not missing. There is also a check that a subexpression not yet defined is not entered in an expression.

It is not possible to ensure that the error messages are always comprehensible to the user. A situation may arise where an expression has a mistake which is detected by the system but misinterpreted. For instance, if the user has omitted the tag name and "=" of an input expression, the first 6 characters will be read, after which a warning that a variable has more than 6 characters is printed. The remainder of the expression until the terminator is ignored. It has been attempted to make the error messages more understandable by printing the offending variable, number, character or tag with the message. Not all the errors lead to walkback termination of the program. This is done only where absolutely necessary, e.g. if the expression is misformed. Too many diagnostic messages were thought to be unnecessary and a nuisance, so their number was kept to a minimum. The walkback termination is of great use as it locates the exact place in the program where the error occurred. This is an improvement on the use of a FORTRAN STOP statement which leads to the normal termination of a program without the exact location of a fault.

Perhaps the most difficult decision which had to be made was how much responsibility to leave to the user and how much to DERIV. The choice of deleting expressions is left entirely to users since they may have to cope with core shortage in attempting to differentiate large expressions. However, retrieval

University of Cape Town

of temporary lists used to store input and output infix expressions and of nodes used by pushdown stacks is part of the SLIP garbage reclamation scheme.

Storage limitations are the reason why READEX, ALGDIF and PRNTEX were programmed separately as well. The user has control over when he performs the various operations rather than automatic I/O by ALGDIF. The user also has the option of inputting expressions in sections using the subexpression mechanism to save storage and time. The dynamic use of storage may mean that a program which runs out of core may execute successfully if rearranged slightly. The bookkeeping required to correlate list names and expression tags as well as to identify derivatives of an expression with respect to different variables is done entirely by DERIV. However, the user has the responsibility of identifying expressions adequately by means of tags. If they are not assigned uniquely to expressions, overwriting of the collator list and DL nodes may cause trouble, particularly if complicated partial differentiation is performed.

Care also has to be taken that expressions are not deleted until they are definitely no longer required by a program. In addition, any sublists referenced by an expression should be differentiated with respect to the variable of differentiation before the expression itself is, otherwise the sublists are treated as constants after a warning message has been printed.

Because more responsibility is placed in the hands of the user than in most packages similar to DERIV, it may be more difficult to get a DERIV program working properly. However, it is hoped that DERIV has been designed in a versatile enough manner to be of considerable use. Examples of programs written in DERIV are illustrated in 3.6.

3.5 Testing and Debugging of DERIV.

This task was found difficult to accomplish thoroughly since SLIP and DERIV together have about 150 different entry points. It was desired to test all these routines one by one, but this is virtually impossible because they use one another constantly. In order to test SLIP, the tutorial examples given in (1) and (2) were used. The primitives and recursion routines were tested and debugged individually. Since the examples which were used for testing purposes covered the routines of SLIP most frequently used, only some of the high-level functions

remained to be checked. This could also be done by the use of examples, together with the LSTRCE routine and walkback.

The reason for most errors was the conversion to a different node size in U-SLIP. In addition, there were a few typographical errors in the FORTRAN listings of SLIP in (1), from which U-SLIP was copied and modified. The grouping into subroutines with multiple entry points also caused some bugs which were difficult to detect.

Ensuring that DERIV was error-free was more complicated. Large amounts of testing were required to discover programming errors and incorrect logic in ALGDIF and the I/O functions. The modular nature of the routines assisted considerably in their debugging since each module of the routines has a specific task. Walkback and the DERIV error messages (even while the routines were still logically incorrect) were helpful too. The assembler trace (37) was useful on occasion.

In order to test all aspects of the differentiation rules and simplification procedures, a large number of expressions were differentiated using the program shown in 3.6 (example 1). Their execution times and core usage were also noted.

3.6 The Capabilities of DERIV.

3.6.1 Examples and Estimates of Their Efficiency.

The examples mentioned below are listed, together with the printouts obtained, in Appendix D.

Example 1 was used for extensive checking of the differentiation rules in ALGDIF, as well as to obtain a rough estimate of execution times in DERIV. A total number of 122 expressions were differentiated for testing purposes. They were mostly very short and simple, their aim being to illustrate the correct working of the rules and any simplification performed. The total execution time was 66.79 seconds. Hence the average time per expression was 0.55 seconds. A realistic estimate of core and time requirements is difficult to make as their usage is completely dependent not only on the length of an expression but on its form. Roughly, using Ex. 1, it was found that the maximum number of nodes needed for an expression varied from $2\frac{1}{2}$ to 5 times its length.

Example 2 illustrates how an expression may be differentiated with respect to a subexpression, rather than a variable.

The purpose of Example 3 is to indicate how core and time

University of Cape Town

may be saved by breaking up a large expression into subexpressions. The input expression is that which was differentiated by Hanson et al. ((10), p.355). The DERIV output does not resemble that of (10) because of a typographical error in that article. The order of outputting factors and their derivatives when multiplied together is reversed as well, but careful checking established the equivalence of the derivatives. Unfortunately (10) does not give an estimate of the time and core used by the derivative. In Table 3 examples 3(a) to 3(c) are compared. From there, it is seen that dramatic decreases in core and time requirements occur when an expression is broken up into subexpressions. This is obtained without much increase in the complexity of the program. An additional advantage is that the expression is more readable and very likely of more significance when broken up in this way. Exs 3(b) and 3(c) are similar and while they take the same execution time, the former uses considerably less storage as the print routine is called at the end of the program for all the expressions and they are deleted immediately after they are printed. In Ex. 3(c) each subexpression is printed as soon as it has been differentiated and thus has still to be retained since it is required as a subexpression of the last expression input. Although (c) is slightly more convenient to program, (b) is certainly more efficient.

Example 4 was included to illustrate the partial differentiation capabilities of DERIV. It has been programmed in two ways in order to show that a big saving in core requirements is possible if expressions not needed any more by the program are deleted as soon as possible (see Table 4 and Appendix D).

The program shown in Ex. 5 was written to solve a problem encountered by E. Cairncross in the Chemical Engineering Department at UCT (see also (39)). In order to obtain an expression for the velocity at any point in a fluid agitated by a flat disc spinning about its axis, the stream function has to be differentiated with respect to the two coordinates in the plane being considered. The stream function for this problem, χ_1, is a function of the orthogonal oblate spheroidal coordinates, ξ and η. It was differentiated by hand but the velocities obtained did not vanish a long distance from the disc. Since a mistake in the differentiation was suspected, DERIV was used to

University of Cape Town

Exercise:	3(a)	3(b)	3(c)
Average Execution Time (secs)	34.52	9.08	9.08
Max. Core Used by AVAIL (words)	6822	1022	1686
Total Core Requirements (words)	29494	24524	24516

Table 3. Comparison of Three Different Approaches to Example 3.

Exercise:	4(a)	4(b)
Average Execution Time (secs)	10.33	9.83
Max. Core Used by AVAIL (words)	2385	1521
Total Core Requirements (words)	24546	24547

Table 4. Comparison of Two Methods of Programming Example 4.

University of Cape Town

differentiate the function automatically. The stream function was broken up into 13 subexpressions since this was how the problem was presented. The derivative was obtained in 33 seconds. The maximum number of words of AVAIL used was 4482. If the expression had not been broken up but had been assembled by hand from all the functions specified in (39), it is certain that the maximum of 14K nodes possible in AVAIL (see 3.6.2) would have been insufficient. Even assuming there were enough storage available, it is unlikely that the solution would have been intelligible.

3.6.2 Inefficiencies in Core and Time Utilization.

As illustrated in Ex. 1 of 3.6.1, for a simple DERIV program to read in an expression, differentiate it and print the derivative, a rough estimate of the core requirements for the program and subroutines is 22K, omitting the list of available space. Assuming that this leaves 42K words for AVAIL (core capacity on the 1106 is 65K), 14K nodes are available. This very likely means that for a successful run the length of a large derivative should not exceed 4K nodes in length, if the input expression is deleted before the derivative is printed. Breaking up an expression into sublists has the effect of reducing storage requirements about five-fold. The only disadvantage is that it has to be done manually before input to DERIV.

One way of reducing heavy storage needs is to change ALGDIF so as to be nonrecursive. In this case, an expression would probably require little more storage than its own length in the differentiation process. However, the programming complexity would have increased greatly, especially with the use of a list processor. An approach such as that described in (15) may just as well have been used then, precluding the use of a list processor. The simplification techniques introduced in ALGDIF may also be a mixed blessing since they require the temporary storage of several parameters at each recursive entry into the program. The modules in ALGDIF have been written in such a way as to delay the necessity of saving parameters as much as possible. Every possibility is explored to avoid such waste of storage by examining the operands at each stage in the recursion carefully. Storage is also needed by the pushdown stack of return addresses.

If INFORD had been programmed non-recursively, the storage

University of Cape Town

requirements of the output routine would have been considerably reduced. If an algorithm which is not difficult to program is found to convert prefix to infix ordering in a non-recursive manner, this routine may be rewritten. A lot of nodes should then be saved for use by expressions rather than as temporary storage.

Another possibility for increasing core capacity if it is critical is to rearrange the grouping into subroutines of the SLIP "library", which presently occupies about 19K. If this can be effected in such a way that some of the subroutines are omitted from an executable program, as many as a thousand more nodes will be available for use by DERIV. This task is difficult since no entry point within a grouping of functions may call another in the same group. Furthermore, most of the routines of SLIP are used in some way or another within DERIV.

A third way of lowering core requirements is if the size of a U-SLIP node can be reduced to two words. This is impossible at present. If DERIV had been based on a list processor such as LISP 1.5 (38), it is possible that core requirements would have been less. However, LISP functions are difficult to debug and the dynamic storage retrieval in that language is completely out of the hands of the user, whereas SLIP gives a programmer more alternatives in handling heavy core usage.

Time-wise SLIP is a relatively slow list processor. Attempts have been made to cut down on this time waste in DERIV. However, the modular nature of the routines, simplifying their debugging, has not been sacrificed. Where possible, extra subroutines have been avoided and the necessary coding inserted directly in the routines, but the desirability of extra routines in order to avoid duplication of coding and to enhance the simplicity of the routines had to be taken into account.

SLIP has some advantageous features. Since lists are symmetric, their deletion is fast and independent of their length. The dynamic storage retrieval scheme of SLIP has advantages over LISP garbage collection. Whereas the latter occurs at regular intervals, SLIP garbage retrieval only takes place when absolutely necessary.

Searching is a factor which increases execution times in DERIV. There are frequent searches in both READEX and ALGDIF (e.g. to identify operators). Although this is unavoidable,

these tables have been arranged in such a way that operators occurring more frequently in common algebraic expressions are placed before others in the operator table. This is difficult to assess, but +, -, *, and ! appear first in the table. An example of the searching necessary: if there is a variable in the temporary variable buffer of READEX, it has to be checked against the entire operator and reserved word tables before it is recognised as an ordinary variable. Furthermore, every symbol read in is checked against a set of symbols not allowed in DERIV. This is necessary in case an input expression has been mispunched.

The sublist mechanism reduces the time taken for a DERIV program, as may be seen in Ex. 3 (see 3.6.1). The execution time in that example was reduced by a factor of four when the expression to be differentiated was divided into subexpressions.

Although DERIV has inefficiencies and is unsophisticated in the simplification introduced, it is hoped that it will be of use in practice. Since it is based on FORTRAN, it is easy to improve, and if the demand at UCT for analytical differentiation by computer increases, such extensions and improvements as are necessary may be implemented.

University of Cape Town

BIBLIOGRAPHY

(1) Weizenbaum, J. Symmetric List Processor. Comm. ACM 6, 9 (1963), p.524.

(2) Smith, D. K. An Introduction to the List-Processing Language, SLIP. In "Programming Systems and Languages", ed. S. Rosen. McGraw-Hill (1967), p.393.

(3) Duquet, R. T. The 1108 SLIP Package. UNIVAC Supported. UPLI Reference No. 800085 (June, 1967).

(4) Ball, W. E. & Berns, R. I. AUTOMAST - Automatic Mathematical Analysis and Symbolic Translation. Presented at the SICSAM Symposium, Washington D.C. (March, 1966).

(5) Gotlieb, C. C. & Novak, R. J. ALGEM - An Algebraic Manipulator. Presented at SICSAM Symposium (1966).

(6) Lapidus, A. & Goldstein, M. Some Experiments in Algebraic Manipulation by Computer. Comm. ACM 8, 8 (1965), p.501.

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

(8) Nolan, J. F. Analytical Differentiation on a Digital Computer. M.A. Thesis. MIT (May, 1953).

(9) Kahrimanian, H. G. Analytical Differentiation by a Digital Computer. M.A. Thesis. Temple Univ. (May, 1953).

(10) Hanson, J. W. et al. Analytical Differentiation by Computer. Comm. ACM 5, 6 (1962), p.349.

(11) Slagle, J. R. A Heuristic Program that Solves Symbolic Integration Problems in Freshman Calculus: SAINT. Report 5 G - 0001. MIT Lincoln Laboratory. (1961).

(13) Hearn, A. C. REDUCE 2: A System and Language for Algebraic Manipulation. Proc. 2nd SICSAM Symposium. ACM, New York. (1971), p.128.

(14) Schorr, H. Analytical Differentiation Using a Syntax-Directed Compiler. Computer J. 7, 4 (1965), p. 290.

(15) Smith, P. J. Symbolic Derivatives without List-Processing, Subroutines or Recursion. Comm. ACM 8, 8 (1965), p.494.

(16) Wengert, R. R. A Simple Automatic Derivative Evaluation Program. Comm. ACM 7, 8 (1964), p.463.

(17) Wilkins, R. D. Investigation of a New Analytical Method for

University of Cape Town

Derivative Evaluation. Comm. ACM 7, 8 (1964), p.465.

(18) Sammet, J. E. & Bond, E. R. Introduction to FORMAC. IEEE Trans. Electron. Comput. EC-13, 4 (1964), p.386.

(19) Tobey, R. G. et al. Automatic Simplification in FORMAC. Proc. AFIPS FJCC 27, part 1 (1965), p.37.

(20) - . FORMAC (Operating and Users Preliminary Reference Manual). IBM Program Inform. Dept. No. 7090 R2IBM 0016. Hawthorne, NY. (1965).

(21) Engelman, C. MATHLAB: A Program for On-Line Assistance in Symbolic Computations. The MITRE Corp. MTP-18 (1965).

(22) D'Inverno, R. A. ALAM - Atlas LISP Algebraic Manipulator. Computer J. 12, 2 (1969), p. 124.

(23) Barton, D. et al. An Algebra System. Computer J. 13, 1 (1970), p.32.

(23a) Hearn, A. C. Computation of Algebraic Properties of Elementary Particle Reactions using a Digital Computer. Comm. ACM 9, 8 (1966), p.573.

(23b) Hearn, A. C. Applications of Symbol Manipulation in Theoretical Physics. Comm. ACM 14, 8 (1971), p.511.

(24) Moses, J. Algebraic Simplification: A Guide for the Perplexed. Comm. ACM 14, 8 (1971), p.527.

(25) Hall, A. D. The ALTRAN System for Rational Function Manipulation - A Survey. Comm. ACM 14, 8 (1971), p.517.

(26) Collins, G. E. PM - A System for Polynomial Manipulation. Comm. ACM 9, 8 (1966), p.578.

(27) Collins, G. E. & Griesmer, J. H. Comparison of Computing Times in Several Algebraic Manipulation Systems. IBM Watson Research Centre. Yorktown Heights, NY. No. 641.

(28) Johnson, S. C. On the Problem of Recognising Zero. J. ACM 18, 4 (1971), p.559.

(29) Caviness, B. F. On Canonical Forms and Simplification. J. ACM 17, 2 (1970), p.385.

(30) Collins, G. E. Polynomial Remainder Sequences and Determinants. IBM Watson Research Centre. Yorktown Heights, NY. Research Report No. RC-1209 (1964).

(31) Collins, G. E. Subresultants and Reduced Polynomial Remainder Sequences. J. ACM 14, 1 (1967), p.128.

(32) Brown, W. S. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors. J. ACM 18, 4 (1971) p.478.

(33) Brown, W. S. & Traub, J. F. On Euclid's Algorithm and the

University of Cape Town

Theory of Subresultants. J. ACM 18, 4 (1971), p.505.

(34) Martin, W. A. Determining the Equivalence of Algebraic Expressions by Hash Coding. J. ACM 18, 4 (1971), p.549.

(35) Wegner, P. Programming Languages, Information Structures and Machine Organisation. McGraw-Hill (1968).

(36) - . UNIVAC 1100 Series FORTRAN V Programmer's Reference Manual. UP-4060 Rev. 2 (1971).

(37) - . UNIVAC 1100 Series EXEC II and EXEC 8 ASSEMBLER Programmer's Reference Manual. UP-4040 Rev. 3 (1971).

(38) McCarthy, J. LISP 1.5 Programmer's Manual. MIT Press. Cambridge, Mass. (Dec., 1969).

(39) Waters, N. D. & King, M. J. The Steady Flow of an Elastico-Viscous Liquid Induced by a Rotating Spheroid. Quart. J. Mechanics & Appl. Maths XXIV, 3 (1971), p.331.

University of Cape Town

APPENDIX A

SLIP: FORTRAN AND ASSEMBLER LISTINGS

University of Cape Town

```
      FUNCTION FNA(DUMMY)
      RETURN
C-----------------------------------------------------------------------
      ENTRY     BOT(K)
C   OBTAIN CONTENTS OF BOTTOM NODE OF LIST K
      FNA=CONT(LNKL(CONT(LNKR(LOCT(K))))+2)
      RETURN
C-----------------------------------------------------------------------
      ENTRY POPBOT(K)
C   REMOVE BOTTOM NODE OF LIST K & RETURN ITS DATUM
      FNA=DELETE(LNKL(CONT(LNKR(LOCT(K)))))
      RETURN
C-----------------------------------------------------------------------
      ENTRY POPTOP(K)
C   REMOVE TOP NODE OF LIST K & RETURN ITS DATUM
      FNA=DELETE(LNKR(CONT(LNKR(LOCT(K)))))
      RETURN
C-----------------------------------------------------------------------
      ENTRY REED(K)
C   RETURN DATUM OF NODE THAT READER K IS CURRENTLY POINTING TO
C   ADDRESS OF NODE HELD IN LNKL FIELD OF WORD 1 OF READER NODE
      FNA =CONT(LNKL(CONT(K))+2)
      RETURN
C-----------------------------------------------------------------------
      ENTRY SEQLL(Z,K)
C   Z IS A SEQUENCER APPOINTED FOR A LIST & K A FLAG
C   MOVE TO NODE TO LEFT OF THAT POINTED TO BY Z
      L=LNKL(Z)
C   ADJUST Z TO POINT TO NEW NODE
      Z=CONT(L)
C   RETURN CONTENTS OF NEW CELL TRAVERSED
      FNA  =CONT(L+2)
C   SET VALUE OF FLAG K
      K=ID(L)-1
      RETURN
C-----------------------------------------------------------------------
      ENTRY SEQLR(Z,K)
C   COMMENTS AS FOR SEQLL ABOVE EXCEPT THAT TRAVERSAL IS TO THE RIGHT
      L=LNKR(Z)
      Z=CONT(L)
      FNA  =CONT(L+2)
      K=ID(L)-1
      RETURN
C-----------------------------------------------------------------------
      ENTRY SEQRDR(K)
C   APPOINT A SEQUENCER FOR LIST K
C   VALUE IS THE NAME OF K
      FNA=CONT(LNKR(LOCT(K)))
      RETURN
C-----------------------------------------------------------------------
      ENTRY     SEQSL(Z,K)
C   Z IS A SEQUENCER APPOINTED FOR A LIST & K A FLAG
C   IS NODE ON LEFT OF THAT POINTED TO BY Z A SUBLIST NODE?
      IF(ID(LNKL(Z)).NE.1) GO TO 14
C   MOVE TO BOTTOM OF SUBLIST REFERENCED BY LIST  NODE
   13 L=LNKL(CONT(LNKL(CONT(LNKL(Z)+2))))
      GO TO 15
```

University of Cape Town

```
C   MOVE TO NODE ON LEFT OF THAT POINTED TO BY Z
   14 L=LNKL(Z)
C    IS NODE ENCOUNTERED A POINTER TO A SUBLIST?
   15 IF(ID(L).EQ.1) GO TO 16
C   VALUE IS CONTENTS OF DATUM CELL FINALLY ENCOUNTERED
      FNA   =CONT(L+2)
C   NEW VALUE OF SEQUENCER Z TO BE RETURNED
      Z=CONT(L)
C   WORK OUT VALUE OF FLAG K TO BE RETURNED
      K=ID(L)-1
      RETURN
   16 Z=CONT(L)
C   MOVE STILL DEEPER INTO LIST STRUCTURE IN SEARCH OF A DATUM CELL
      GO TO 13
C-----------------------------------------------------------------------
      ENTRY     SEQSR(Z,K)
C   COMMENTS AS FOR SEQSL ABOVE EXCEPT THAT TRAVERSAL IS TO THE RIGHT
      IF(ID(LNKR(Z)).NE.1) GO TO 18
   17 L=LNKR(CONT(LNKR(CONT(LNKR(Z)+2))))
      GO TO 19
   18 L=LNKR(Z)
   19 IF(ID(L).EQ.1) GO TO 20
      FNA   =CONT(L+2)
      Z=CONT(L)
      K=ID(L)-1
      RETURN
   20 Z=CONT(L)
      GO TO 17
C-----------------------------------------------------------------------
      ENTRY SUBST(K,N)
C   LBACK IS POINTER TO NODE BEFORE N
      LBACK=LNKL(CONT(N))
C   DELETE NODE N
      FNA   =DELETE(N)
C   INSERT NEW NODE IN ITS PLACE WITH DATUM K
      IDUM=NXTRGT(K,LBACK)
      RETURN
C-----------------------------------------------------------------------
      ENTRY     TOP(K)
C   CHECK THAT K IS A LIST. THEN OBTAIN CONTENTS OF ITS TOP NODE
      FNA=CONT(LNKR(CONT(LNKR(LOCT(K))))+2)
      RETURN
      END
```

University of Cape Town

```
      FUNCTION FNB(DUMMY)
      COMMON /SLIP/ W(101),AVSL
      INTEGER FNB
      RETURN
C-----------------------------------------------------------------------
      ENTRY INITRD(K)
C   INITIALISE READER K BY MAKING IT POINT TO CURRENT LIST'S
C   HEADER HELD IN LNKL FIELD OF WORD 3 OF K
      CALL SETIND(-1,LNKL(K+2),-1,K)
      FNB    =K
      RETURN
C-----------------------------------------------------------------------
      ENTRY INLSTL(M,N)
C   INSERT LIST M AFTER NODE N TO ITS LEFT
C   CHECK THAT M EXISTS
      L=LOCT(M)
C   TOP & BOTTOM NODES OF LIST M
      ITOP=LNKR(CONT(L))
      IBOT=LNKL(CONT(L))
      FNB    =L
C   REINITIALISE HEADER OF M SO THAT IT APPEARS TO BE EMPTY
      CALL SETIND(-1,L,L,L)
C   POINTER TO NODE BELOW N BEFORE NEW NODES INSERTED
      IPRE=LNKR(CONT(N))
C   LINK NODE N TO BOTTOM OF OLD LIST M
      CALL SETIND(-1,IBOT,-1,N)
C   LINK TOP OF OLD LIST M TO NODE BELOW N AS WAS BEFORE
      CALL SETIND(-1,-1,ITOP,IPRE)
C   COMPLETE LINKAGES IN OPPOSITE DIRECTION
      CALL SETIND(-1,IPRE,-1,ITOP)
      CALL SETIND(-1,-1,N,IBOT)
      RETURN
C-----------------------------------------------------------------------
      ENTRY INLSTR(M,N)
C   AS INLSTL ABOVE EXCEPT THAT LIST M INSERTED BELOW NODE N
      L=LOCT(M)
      ITOP=LNKR(CONT(L))
      IBOT=LNKL(CONT(L))
      FNB    =L
      CALL SETIND(-1,L,L,L)
      ISUC=LNKR(CONT(N))
      CALL SETIND(-1,-1,ITOP,N)
      CALL SETIND(-1,IBOT,-1,ISUC)
      CALL SETIND(-1,N,-1,ITOP)
      CALL SETIND(-1,-1,ISUC,IBOT)
      RETURN
C-----------------------------------------------------------------------
      ENTRY LCNTR(K)
C   ACCESS LCNTR FIELD OF READER K
      FNB=LNKR(CONT(LNKR(K+2)))
      RETURN
C-----------------------------------------------------------------------
      ENTRY LISTMT(K)
C   CHECK THAT LIST K EXISTS
      L=LOCT(K)
C   LNKL OF LIST HEADER = LNKR?
      IF(INHALT(LNKR(L)).NE.INHALT(LNKR(CONT(LNKR(L))))) GO TO 2
```

University of Cape Town

```
C   LNKL=LNKR IN LIST HEADER HENCE LIST EMPTY
      FNB    =0
      RETURN
C   LIST NOT EMPTY
    2 FNB    =-1
      RETURN
C-----------------------------------------------------------------------
      ENTRY LOFRDR(K)
C   RETURN IN NAME FORMAT THE NAME OF LIST CURRENTLY BEING
C   TRACED BY READER K
C   OBTAIN LOFRDR FIELD OF READER
      L=LNKL(CONT(K+2))
C   PUT LIST NAME OBTAINED IN NAME FORMAT
      CALL SETDIR(-1,L,L,L)
      FNB    =L
      RETURN
C-----------------------------------------------------------------------
      ENTRY LPNTR(K)
C   RETURN LPNTR FIELD OF READER K - ADDRESS OF CELL POINTED TO
C   CURRENTLY BY THE READER
      FNB=LNKL(CONT(LNKR(K)))
      RETURN
C-----------------------------------------------------------------------
      ENTRY LSTMRK(K)
C   OBTAIN THE MARK OF LIST K
      FNB    =LOCT(K)
      DUMY=GETMRK(LNKR(K))
      RETURN
C-----------------------------------------------------------------------
      ENTRY LSTPRO(M,N)
C   N IS A READER, M A DATUM WHICH IS A LIST NAME
      NEXT=M
C   IS LOFRDR OF READER CELL A LIST NAME?
    3 IF(LNKL(NEXT+2).EQ.LNKR(M)) GO TO 4
C   CONSIDER NEXT LOWER CELL OF READER STACK
      NEXT=LNKR(NEXT)
C   HAVE REACHED BOTTOM OF READER STACK YET?
      IF(NEXT.NE.0) GO TO 3
C   HAVE REACHED BOTTOM OF READER STACK WITHOUT A REFERENCE TO M
      FNB    =-1
      RETURN
C   HAVE FOUND A REFERENCE IN READER STACK TO LIST M, I.E.
C   READER HAS TRAVERSED THROUGH THAT LIST TO GET TO HERE
    4 FNB    =0
      RETURN
C-----------------------------------------------------------------------
      ENTRY LVLRVT(K)
C   K IS THE READER OF A LIST
      FNB    =K
C   IF LNKR OF READER CELL ZERO, ARE AT BOTTOM OF READER STACK AND
C   HENCE IN MAIN LIST SO RETURN
    5 IF(LNKR(CONT(FNB+2)).EQ.0) RETURN
C   MOVE DOWN READER STACK
      L=LNKR(CONT(FNB    ))
C   POP UP THE READER STACK
      DUMY=STRIND(CONT(L),FNB    )
      DUMY=STRIND(CONT(L+2),FNB    +2)
```

University of Cape Town

```
      CALL RCELL(L)
C   CONTINUE DELETING THE READER CELLS
      GO TO 5
C---------------------------------------------------------------------
      ENTRY LVLRV1(K)
C   K IS THE READER OF A LIST
      FNB    =K
C   IF AT BOTTOM OF READER STACK CANNOT REVERSE BACK A LEVEL IN THE LIST
C   STRUCTURE AS ALREADY IN MAIN LIST & HENCE RETURN
      IF(LNKR(CONT(FNB+2)).EQ.0) RETURN
C   MOVE DOWN READER STACK
      L=LNKR(CONT(FNB    ))
C   POP UP THE READER STACK
      DUMY=STRIND(CONT(L),FNB    )
      DUMY=STRIND(CONT(L+2),FNB    +2)
      CALL RCELL(L)
      RETURN
C---------------------------------------------------------------------
      ENTRY MADATR(AT,LST)
C   OBTAIN THE DL FIELD OF LIST LST
      LSTDES=LNKL(CONT(LNKL(LST)+2))
C   DOES LIST LST HAVE A DL? IF NOT DL FIELD OF HEADER IS ZERO
      IF(LSTDES.EQ.0) GO TO 7
C   PROCEED TO TOP OF DL
      FNB    =LNKR(CONT(LSTDES))
C   IS LNKL POSITIVE? THEN ARE AT HEADER OF DL AGAIN.
    6 IF(CONT(FNB    ).GE.0) GO TO 7
C   IF ATTRIBUTE OF NODE CURRENTLY BEING INSPECTED = AT, RETURN
C   MACHINE ADDRESS OF THIS NODE
      IF(INHALT(FNB    +1).EQ.INTGER(AT)) RETURN
C   CONTINUE SCAN OF DL NODES
      FNB    =LNKR(CONT(FNB    ))
      GO TO 6
C   NO DL ATTACHED TO LST
    7 FNB    =-1
      RETURN
C---------------------------------------------------------------------
      ENTRY MADLFT(K)
C   VALUE IS ADDRESS OF NODE TO LEFT OF K
      FNB=LNKL(CONT(LNKR(K)))
C   IS NODE A HEADER?
      IF(ID(FNB    ).NE.2) RETURN
C   IF NODE A HEADER RETURN LIST NAME IN NAME FORMAT
      CALL SETDIR(-1,FNB,FNB,FNB)
      RETURN
C---------------------------------------------------------------------
      ENTRY MADNBT(K,N)
C   CHECK THAT LIST K EXISTS
      L=LOCT(K)
C   FIND N-TH NODE FROM BOTTOM OF K
      DO 8 I=1,N
    8 L=LNKL(CONT(L))
C   IF NODE NOT A HEADER RETURN MACHINE ADDRESS
      IF(ID(L).NE.2) GO TO 9
C   IF NODE A HEADER RETURN LIST NAME
      CALL SETDIR(-1,L,L,L)
C   RETURN ADDRESS OF NODE L IF ABOVE INSTRUCTION SKIPPED
```

University of Cape Town

```
    9 FNB    =L
      RETURN
C-----------------------------------------------------------------------
      ENTRY MADNTP(K,N)
C   AS MADNBT ABOVE BUT WANT N-TH NODE FROM TOP OF LIST K
      L=LOCT(K)
      DO 10 I=1,N
   10 L=LNKR(CONT(L))
      IF(ID(L).NE.2) GO TO 11
      CALL SETDIR(-1,L,L,L)
   11 FNB    =L
      RETURN
C-----------------------------------------------------------------------
      ENTRY MADRGT(K)
C   AS MADLFT ABOVE BUT REQUIRE MACHINE ADDRESS OF NODE TO RIGHT OF K
      FNB=LNKR(CONT(LNKR(K)))
      IF(ID(FNB    ).NE.2) RETURN
      CALL SETDIR(-1,FNB,FNB,FNB)
      RETURN
C-----------------------------------------------------------------------
      ENTRY MRKLST(M,N)
C   SET MARK OF LIST N TO M
      FNB    =LOCT(N)
      DUMY=PUTMRK(M,LNKR(N))
      RETURN
C-----------------------------------------------------------------------
      ENTRY MTLIST(K)
C   EMPTY ALL NODES OF LIST K TO AVAIL EXCEPT LIST HEADER.
C   CHECK THAT K EXISTS
      L=LOCT(K)
C   IS THE LIST K EMPTY ALREADY?
      IF(INHALT(K).EQ.INHALT(LNKR(CONT(K)))) GO TO 12
C   LIST NOT EMPTY. LR & LL POINT TO FIRST & LAST OF CELLS TO BE REMOVED
      LR=LNKR(CONT(L))
      LL=LNKL(CONT(L))
C   MAKE HEADER NODE POINT TO ITSELF
      CALL SETIND(-1,L,L,L)
C   ADJUST AVSL (QUASINAME OF AVAIL) SO IT INCLUDES CELLS BEING RETURNED
C   TO AVAIL & LINK IN THE CELLS TO AVAIL
      CALL SETIND(-1,-1,LR,LNKL(AVSL))
      CALL SETDIR(-1,LL,-1,AVSL)
      CALL SETIND(-1,-1,0,LNKL(AVSL))
C   RETURN LIST NAME
   12 FNB=L
      RETURN
C-----------------------------------------------------------------------
      ENTRY NAMEDL(K)
C   ACCESS CONTENTS OF DL FIELD OF HEADER OF LIST K
      FNB    =LNKL(LOCT(K)+2)
      RETURN
      END
```

University of Cape Town

```
      FUNCTION FNC(DUMMY)
C   THE 4 ROUTINES FOLLOWING USED BY THE ADVANCE ROUTINES LISTED BELOW
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVLL(LR,J,K)
C   LR A READER, J & K THE ALTERNATIVE ID'S OBJECTIVE NODE MAY HAVE
C   CLR IS ADDRESS OF TOP NODE OF READER STACK
      CLR=CONT(LR)
C   LK IS ADDRESS OF CELL TO LEFT OF THAT READER IS CURRENTLY
C   POINTING TO
    1 LK=LNKL(CONT(LNKL(CLR)))
      CAND=REALNO(LK)
C   ADJUST CLR TO POINT TO LK
      CALL SETDIR(-1,LK,-1,CLR)
C   IS LK THE HEADER NODE OF THE LIST?
      IF(ID(CAND).EQ.2) GO TO 3
C   IS ID OF LK 1-ST ALTERNATIVE REQUIRED?
      IF(ID(CAND).EQ.J) GO TO 2
C   IS ID OF LK 2-ND ALTERNATIVE REQUIRED. IF NOT CONTINUE SCAN
      IF(ID(CAND).NE.K) GO TO 1
C   HAVE FOUND OBJECTIVE NODE
    2 FNC  =0.0
      GO TO 4
C   TEST HAS FAILED AS ENCOUNTERED HEADER OF LIST BEFORE FINDING
C   OBJECTIVE NODE
    3 FNC  =-1.0
C   ADJUST TOP CELL OF READER TO POINT TO CELL FOUND
    4 DUMY=STRIND(CLR,LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVLR(LR,J,K)
C   COMMENTS AS FOR ADVLL ABOVE BUT READER TRAVERSES DOWN LIST HERE
      CLR=CONT(LR)
    5 LK=LNKR(CONT(LNKL(CLR)))
      CAND=REALNO(LK)
      CALL SETDIR(-1,LK,-1,CLR)
      IF(ID(CAND).EQ.2) GO TO 7
      IF(ID(CAND).EQ.J) GO TO 6
      IF(ID(CAND).NE.K) GO TO 5
    6 FNC  =0.0
      GO TO 8
    7 FNC  =-1.0
    8 DUMY=STRIND(CLR,LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVSL(L,J,K)
C   L IS A READER, J & K ALTERNATIVE TYPES OF OBJECTIVE NODE
C   TO SEARCH FOR
C   R HAS ADDRESS OF TOP NODE OF READER STACK
      R=CONT(L)
C   CAND IS ADDRESS OF NODE THAT READER CURRENTLY POINTING TO
      CAND=REALNO(LNKL(R))
C   IS THIS CELL A SUBLIST NODE?
      IF(ID(CAND).EQ.1) GO TO 10
C   EXAMINE NEXT NODE TO THE LEFT ON LIST
    9 LCP=LNKL(CONT(CAND))
C   PUT THIS NODE ADDRESS INTO LNKL FIELD OF R SO R FORMATTED
```

University of Cape Town

```
C   AS WORD 1 OF TOP CELL OF READER STACK
      CALL SETDIR(-1,LCP,-1,R)
      CAND=REALNO(LCP)
C   IS THE NEW CELL THE LIST'S HEADER?
      IF(ID(CAND).EQ.2) GO TO 11
C   IS NEW CELL ONE OF ALTERNATIVE OBJECTIVE CELLS BEING
C   SOUGHT?
      IF((ID(CAND).EQ.J).OR.(ID(CAND).EQ.K)) GO TO 13
C   IS NEW CELL A SUBLIST NODE? IF NOT CONTINUE LEFT SCAN
      IF(ID(CAND).NE.1) GO TO 9
C   CELL EXAMINED WAS A SUBLIST NODE. TAKE NEW CELL FROM AVAIL
   10 M=NUCELL(DUMY)
C   PUSH CELL ON TOP OF READER STACK WITH CONTENTS OF ADDRESS IN
C   R IN WORD 1 OF THE CELL
      DUMY=STRIND(R,M)
C   WORDS 2 & 3 OF NEW READER CELL COPIED FROM OLD CELL
      DUMY=STRIND(CONT(L+1),M+1)
      DUMY=STRIND(CONT(L+2),M+2)
C   OLD TOP OF READER STACK HAS LCNTR INCREASED & LOFROR NOW
C   POINTS TO HEADER OF SUBLIST ENTERED
      CALL SETIND(-1,INHALT(LCP+2),LCNTR(L)+1,L+2)
C   ADJUST R TO POINT TO NEW TOP OF READER STACK
      CALL SETDIR(-1,-1,M,R)
C   CAND IS NAME OF SUBLIST TO BE ENTERED SO POINTS TO HEADER OF SUBLIST
      CAND=CONT(LNKL(R)+2)
C   CONTINUE SCAN OF NODES
      GO TO 9
C   HAVE REACHED HEADER OF LIST BEING TRAVERSED.IS IT MAIN LIST?
   11 IF(LCNTR(L).NE.0) GO TO 12
C   LIST IS MAIN LIST SO TERMINATE ADVANCE
      FNC  =-1.0
      GO TO 14
C   HAVE REACHED END OR SUBLIST WITHOUT FINDING OBJECTIVE,
C   SO POP UP READER STACK
   12 LK=LNKR(R)
      R=CONT(LK)
      DUMY=STRIND(CONT(LK+2),L+2)
C   CAND IS ADDRESS OF NODE WHERE DESCENT INTO SUBLIST STARTED
      CAND=REALNO(LNKL(R))
C   REMOVE READER CELL
      CALL RCELL(LK)
C   CONTINUE SCAN THROUGH LIST STRUCTURE
      GO TO 9
C   HAVE FOUND OBJECTIVE NODE
   13 FNC  =0.0
C   ADJUST TOP OF READER TO POINT TO CELL FOUND
   14 DUMY=STRIND(R,L)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVSR(L,J,K)
C   COMMENTS AS FOR ADVSL ABOVE BUT READER TRAVERSES RIGHTWARDS
      R=CONT(L)
      CAND=REALNO(LNKL(R))
      IF(ID(CAND).EQ.1) GO TO 16
   15 LCP=LNKR(CONT(CAND))
      CALL SETDIR(-1,LCP,-1,R)
      CAND=REALNO(LCP)
```

University of Cape Town

```
      IF(ID(CAND).EQ.2) GO TO 17
      IF((ID(CAND).EQ.J) .OR.(ID(CAND).EQ.K)) GO TO 19
      IF(ID(CAND).NE.1) GO TO 15
   16 M=NUCELL(DUMY)
      DUMY=STRIND(R,M)
      DUMY=STRIND(CONT(L+1),M+1)
      DUMY=STRIND(CONT(L+2),M+2)
      CALL SETIND(-1,INHALT(LCP+2),LCNTR(L)+1,L+2)
      CALL SETDIR(-1,-1,M,R)
      CAND=CONT(LNKL(R)+2)
      GO TO 15
   17 IF(LCNTR(L).NE.0) GO TO 18
      FNC  =-1.0
      GO TO 20
   18 LK=LNKR(R)
      R=CONT(LK)
      DUMY=STRIND(CONT(LK+2),L+2)
      CAND=REALNO(LNKL(R))
      CALL RCELL(LK)
      GO TO 15
   19 FNC  =0.0
   20 DUMY=STRIND(R,L)
      RETURN
C-----------------------------------------------------------------------------
      ENTRY DELETE(K)
C   IS NODE K A LIST HEADER?
      IF(ID(K).NE.2) GO TO 21
C  CANNOT DELETE HEADER NODE THIS WAY
      CALL TRCOFF
      PRINT 901
  901 FORMAT('1ATTEMPT TO DELETE LIST HEADER DISREGARDED')
      FNC   =0.0
      RETURN
C  RETURN DATUM OF CELL TO BE DELETED AS VALUE OF FUNCTION
   21 FNC=CONT(K+2)
C  POINTERS TO CELLS ON EITHER SIDE OF K
      LL=LNKL(CONT(K))
      LM=LNKR(CONT(K))
C  LINK TOGETHER CELLS ON EITHER SIDE OF REMOVED NODE K
      CALL SETIND(-1,-1,LM,LL)
      CALL SETIND(-1,LL,-1,LM)
C  RETURN CELL K TO AVAIL
      CALL RCELL(K)
      RETURN
C-----------------------------------------------------------------------------
      ENTRY EQUAL(A,B)
      IF(A-B) 22,23,22
C   A NOT =B
   22 FNC  =-1.0
      RETURN
C   A=B
   23 FNC  =0.0
      RETURN
      END
```

University of Cape Town

```
      FUNCTION FND(DUMMY)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVLEL(LR,A)
C   ADVANCE LINEAR ELEMENT LEFT USING READER LR, FLAG A
C   LEFT LINEAR ADVANCE ROUTINE CALLED TO SEARCH FOR OBJECTIVE
C   CELL WITH ID=0
      A=ADVLL(LR,0,0)
C   WAS ADVANCE SUCCESSFUL?
      IF(ABS(A).GE.0.1) RETURN
C   ADVANCE SUCCESSFUL - RETURN DATUM OF CELL THAT LR NOW POINTS TO
      FND    =REED(LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVLNL(LR,A)
C   ADVANCE LINEAR NAME LEFT: SEARCH FOR OBJECTIVE CELL WITH ID=1
      A=ADVLL(LR,1,1)
      IF(ABS(A).GE.0.1) RETURN
      FND    =REED(LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVLWL(LR,A)
C   ADVANCE LINEAR WORD LEFT: SEARCH FOR OBJECTIVE CELL WITH ID 0 OR 1
      A=ADVLL(LR,1,0)
      IF(ABS(A).GE.0.1) RETURN
      FND    =REED(LR)
      RETURN
C-----------------------------------------------------------------------
C   THE FOLLOWING 3 ROUTINES AS THOSE ABOVE BUT TRAVERSAL IS TO THE
C   RIGHT
      ENTRY ADVLER(LR,A)
      A=ADVLR(LR,0,0)
      IF(ABS(A).GE.0.1) RETURN
      FND    =REED(LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVLNR(LR,A)
      A=ADVLR(LR,1,1)
      IF(ABS(A).GE.0.1) RETURN
      FND    =REED(LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVLWR(LR,A)
      A=ADVLR(LR,1,0)
      IF(ABS(A).GE.0.1) RETURN
      FND    =REED(LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVSEL(LR,A)
C   ADVANCE STRUCTURAL ELEMENT LEFT: NEED OBJECTIVE CELL OF ID 0
      A=ADVSL(LR,0,0)
      IF(ABS(A).GE.0.1) RETURN
      FND    =REED(LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVSNL(LR,A)
C   ADVANCE STRUCTURAL NAME LEFT: SEARCH FOR OBJECTIVE CELL OF ID 1
```

University of Cape Town

```
      A=ADVSL(LR,1,1)
      IF(ABS(A).GE.0.1) RETURN
      END    =REED(LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVSWL(LR,A)
C   ADVANCE STRUCTURAL WORD LEFT: SEARCH FOR OBJECTIVE CELL WITH
C   ID=0 OR 1
      A=ADVSL(LR,1,0)
      IF(ABS(A).GE.0.1) RETURN
      END    =REED(LR)
      RETURN
C-----------------------------------------------------------------------
C   THE FOLLOWING 3 ROUTINES AS THOSE ABOVE BUT TRAVERSAL IS TO
C   THE RIGHT
      ENTRY ADVSER(LR,A)
      A=ADVSR(LR,0,0)
      IF(ABS(A).GE.0.1) RETURN
      END    =REED(LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVSNR(LR,A)
      A=ADVSR(LR,1,1)
      IF(ABS(A).GE.0.1) RETURN
      END    =REED(LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY ADVSWR(LR,A)
      A=ADVSR(LR,1,0)
      IF(ABS(A).GE.0.1) RETURN
      END    =REED(LR)
      RETURN
      END
```

University of Cape Town

```
      FUNCTION FNE(DUMMY)
      INTEGER FNE
      RETURN
C-----------------------------------------------------------------------
      ENTRY IRARDR(K)
C   K IS A READER TO BE ERASED
C   RETURN VALUE OF LEVEL COUNTER OF K
      FNE    =LCNTR(K)
      M=K
C   N IS A POINTER TO NEXT LOWER ELEMENT OF READER STACK
    1 N=LNKR(CONT(M))
C   POP UP READER STACK
      CALL RCELL(M)
C   HAS BOTTOM OF READER STACK BEEN REACHED?
      IF(N.EQ.0) RETURN
C   CONTINUE DELETING NODES OF READER STACK
      M=N
      GO TO 1
C-----------------------------------------------------------------------
      ENTRY ITSVAL(AT,LST)
C   IS THE DL FIELD OF LIST LST ZERO?
      IF(LNKL(CONT(LST+2)).EQ.0) GO TO 2
C   FIND MACHINE ADDRESS OF ATTRIBUTE AT ON DL OF LST
      M=MADATR(AT,LST)
C   DOES THE ATTRIBUTE NOT EXIST ON THE DL
      IF(M.EQ.-1) GO TO 3
C   RETURN VALUE CORRESPONDING TO AT
      FNE    =INHALT(M+2)
      RETURN
C   NO DL ATTACHED TO LST SO PRINT ERROR MESSAGE
    2 CALL DERROR(LST)
    3 FNE    =0
      RETURN
C-----------------------------------------------------------------------
      ENTRY LIST(K)
C   OBTAIN NEW CELL FROM AVAIL - ITS ADDRESS RETURNED AS VALUE
      FNE =NUCELL(DUMY)
C   CHANGE VALUE OF FUNCTION TO LIST NAME FORMAT
      CALL SETDIR(-1,FNE,FNE,FNE)
C   SET UP HEADER NODE IN CELL
      CALL SETIND(2,FNE,FNE,FNE)
      IF(K.EQ.9) RETURN
C   IF K NOT EQUAL 9, LCNTR OF NEW EMPTY LIST SET TO 1
      CALL SETIND(-1,-1,1,FNE+2)
C   RETURN LIST NAME IN FUNCTION ARGUMENT
      K=FNE
      RETURN
C-----------------------------------------------------------------------
      ENTRY LRDRCP(K)
C   COPY READER K INTO A NEW STACK
C   NEW CELL TAKEN FROM AVAIL FOR TOP CELL OF READER & THIS RETURNED
C   AS READER NAME
      FNE    =NUCELL(DUMY)
C   'NEWR' POINTS TO INITIALISED POSITION IN COPIED READER
      NEWR=FNE
C   'NOW' POINTS TO INITIALISED POSITION IN K STACK
      NOW=K
```

University of Cape Town

```
C   COPY 'NEW' INTO 'NEWR'
  4 DUMY=STRIND(CONT(NOW),NEWR)
    DUMY=STRIND(CONT(NOW+1),NEWR+1)
    DUMY=STRIND(CONT(NOW+2),NEWR+2)
C   OBTAIN POINTER TO NEXT ELEMENT DOWN K STACK
    NOW=LNKR(CONT(NOW))
C   HAS LAST CELL OF K STACK BEEN COPIED ALREADY?
    IF(NOW.EQ.0) RETURN
C   NEW CELL OBTAINED FOR READER-COPY
    NEW=NUCELL(DUMY)
C   LINK CURRENT CELL IN COPIED STACK TO NEW CELL OBTAINED
    CALL SETIND(-1,-1,NEW,NEWR)
C   CONTINUE COPYING PROCESS
    NEWR=NEW
    GO TO 4
C-----------------------------------------------------------------------
    ENTRY LRDROV(K)
C   APPOINT READER FOR LIST K
C   CREATE NEW CELL FOR READER & RETURN ITS ADDRESS AS READER NAME
    FNE    =NUCELL(DUMY)
C   SET UP FIELDS OF READER CELL SO THAT IT POINTS TO HEADER OF K
    CALL SETIND(3,LOCT(K),0,FNE)
    CALL SETIND(-1,K,0,FNE+2)
    RETURN
C-----------------------------------------------------------------------
    ENTRY MTDLST(K)
C   EMPTY DL OF LIST K
    FNE    =K
C   OBTAIN DL-FIELD OF HEADER OF LIST K
    M=LNKL(CONT(LOCT(K)+2))
C   IF DL-FIELD ZERO, DL DOES NOT EXIST ANYWAY
    IF(M.EQ.0) RETURN
C   CREATE NAME OF DL IN NAME FORMAT
    CALL SETDIR(-1,M,M,X)
C   EMPTY DL
    IDUM=MTLIST(X)
    RETURN
C-----------------------------------------------------------------------
    ENTRY NEWBOT(K,L)
C   PLACE DATUM K ON BOTTOM OF LIST L
    FNE    =NXTLFT(K,LOCT(L))
    RETURN
C-----------------------------------------------------------------------
    ENTRY NEWTOP(K,L)
C   PLACE DATUM K ON TOP OF LIST L
    FNE    =NXTRGT(K,LOCT(L))
    RETURN
C-----------------------------------------------------------------------
    ENTRY NOATVL(AT,LST)
C   OBTAIN MACHINE ADDRESS OF ATTRIBUTE AT ON LIST LST
    M=MADATR(AT,LST)
C   HAS THE ATTRIBUTE NOT BEEN FOUND?
    IF(M.EQ.-1) GO TO 5
C   RETURN WITH VALUE CORRESPONDING TO 'AT'
    FNE    =INHALT(M+2)
C   DELETE CELL WHICH CONTAINED ATTRIBUTE & CORRESPONDING VALUE
    DUMY=DELETE(M)
```

University of Cape Town

```
      RETURN
C   ATTRIBUTE NOT FOUND
    5 FNE    =0
      RETURN
      END
```

University of Cape Town

```
      FUNCTION FNG(DUMMY)
      INTEGER FNG
      RETURN
C-----------------------------------------------------------------------
      ENTRY LISTAV(M)
C   CREATE A LIST WITH LCNTR=0 & RETURN ITS NAME
      FNG    =LIST(9)
C   PUT ADDRESS OF THIS LIST IN DL FIELD OF LIST M
      CALL SETIND(-1,LNKR(FNG    ),-1,M+2)
      RETURN
C-----------------------------------------------------------------------
      ENTRY MAKEDL(L,M)
C   EMPTY OLD DL OF LIST M & IF IT DOES NOT EXIST, ZERO RETURNED
      FNG    =MTDLST(M)
C   CHECK THAT THE LISTS L & M EXIST
      N=LOCT(M)
      K=LOCT(L)
C   MAKE LIST L A DL OF LIST M
      CALL SETIND(-1,K,-1,N+2)
C   INCREASE THE LCNTR OF DL L
      CALL SETIND(-1,-1,LCNTR(L)+1,K+2)
      RETURN
C-----------------------------------------------------------------------
      ENTRY MRKLSS(L,M)
C   GIVE LIST M & ALL ITS SUBLISTS THE MARK L
      FNG    =M
C   MARK LIST M & APPOINT A READER LR FOR M
      LR=LRDROV(MRKLST(L,M))
C   TRAVERSE LIST STRUCTURE M SEARCHING FOR REFERENCES TO SUBLISTS
    1 X=ADVSNR(LR,K)
C   HAS THE SEARCH BEEN COMPLETED?
      IF(K.NE.0) GO TO 2
C   MARK THE SUBLIST REFERRED TO IN THE SUBLIST NODE ENCOUNTERED BY LR
      IDUM=MRKLST(L,LNKR(X))
C   CONTINUE THE TRAVERSAL
      GO TO 1
C   SEARCH COMPLETED - HENCE DELETE THE READER LR
    2 CALL RCELL(LR)
      RETURN
C-----------------------------------------------------------------------
      ENTRY NULSTL(L,M)
C   ALL CELLS TO LEFT OF L REMOVED FROM LIST M & PLACED ON NEW LIST
C   CREATE LIST WITH LCNTR OF 0
      FNG    =LIST(9)
C   IS CELL FROM WHICH SPLITTING IS TO TAKE PLACE THE LIST HEADER?
      IF(ID(L).NE.2) GO TO 3
C   HEADER NODE OF EMPTY LIST FORMED
      CALL SETIND(2,FNG,FNG,FNG)
      RETURN
C   POINTER TO TOP OF LIST M (NODE TO RIGHT OF HEADER)
    3 LTOP=LNKR(CONT(M))
C   POINTER TO CELL TO RIGHT OF SPLITTING OBTAINED
      LSUC=LNKR(CONT(L))
C   JOIN TOGETHER CELLS OMITTED FROM THE SPLIT
      CALL SETIND(-1,-1,LSUC,M)
      CALL SETIND(-1,M    ,-1,LSUC)
C   CREATE HEADER NODE OF NODES TO BE REMOVED FROM M
```

University of Cape Town

```
      CALL SETIND(2,L    ,LTOP,FNG)
C   JOIN REMOVED NODES TO NEW HEADER CELL
      CALL SETIND(-1,-1,FNG,L)
      CALL SETIND(-1,FNG,-1,LTOP)
      RETURN
C-----------------------------------------------------------------------
      ENTRY NULSTR(L,M)
C   AS NULSTL ABOVE BUT SPLITTING IS TO RIGHT OF L
      FNG    =LIST(9)
      IF(ID(L    ).NE.2) GO TO 4
      CALL SETIND(2,FNG,FNG,FNG)
      RETURN
    4 LBOT=LNKL(CONT(M))
      LPRE=LNKL(CONT(L))
      CALL SETIND(-1,LPRE,-1,M)
      CALL SETIND(-1,-1,M    ,LPRE)
      CALL SETIND(2,LBOT,L    ,FNG)
      CALL SETIND(-1,FNG,-1,L)
      CALL SETIND(-1,-1,FNG,LBOT)
      RETURN
C-----------------------------------------------------------------------
      ENTRY NXTLFT(M,L)
C   TAKE NEW CELL FROM AVAIL & INSERT TO LEFT OF CELL L WITH
C   A DATUM VALUE OF M
      IL=NUCELL(DUMY)
C   RETURN ADDRESS OF NEW CELL CREATED
      FNG    =IL
C   LL IS POINTER TO CELL CURRENTLY AT RIGHT OF L
      LL=LNKL(CONT(LNKR(L)))
C   CHANGE POINTERS OF CELLS SO THAT NEW CELL LINKED INBETWEEN L & LL
      CALL SETIND(-1,-1,IL,LL)
      CALL SETIND(-1,IL,-1,LNKR(L))
      CALL SETIND(0,LL,LNKR(L),IL)
C   IS THE DATUM BEING INSERTED A LIST NAME?
      IF(NAMTST(M).NE.0) GO TO 5
C   DATUM IS A LIST NAME SO GIVE NEW CELL ADDED IN ID=1
      CALL SETIND(1,-1,-1,IL)
C   INCREMENT LCNTR OF LIST WHOSE LIST NAME IS DATUM OF NEW CELL
      CALL SETIND(-1,-1,LCNTR(M)+1,M+2)
C   INSERT DATUM INTO THE NEW CELL TO LEFT OF L
    5 DUMY=STRIND(M,IL+2)
      RETURN
C-----------------------------------------------------------------------
      ENTRY NXTRGT(M,L)
C   AS NXTLFT ABOVE EXCEPT THAT NEW CELL IS ADDED TO RIGHT OF CELL L
      IR=NUCELL(DUMY)
      FNG    =IR
      LR=LNKR(CONT(LNKR(L)))
      CALL SETIND(-1,IR,-1,LR)
      CALL SETIND(-1,-1,IR,LNKR(L))
      CALL SETIND(0,LNKR(L),LR,IR)
      IF(NAMTST(M).NE.0) GO TO 6
      CALL SETIND(1,-1,-1,IR)
      CALL SETIND(-1,-1,LCNTR(M)+1,M+2)
    6 DUMY=STRIND(M,IR+2)
      RETURN
      END
```

University of Cape Town

```
      FUNCTION FNH(DUMMY)
      COMMON /SLIP/ W(101),AVSL
      RETURN
C-----------------------------------------------------------------------
      ENTRY PARMT2(A,B)
C  PUSH A & B DOWN ON FIRST TWO PUBLIC LISTS
      IDUM=NEWTOP(A,W(1))
      IDUM=NEWTOP(B,W(2))
      FNH   =A
      RETURN
C-----------------------------------------------------------------------
      ENTRY PARMTN(VAL,NUM)
      DIMENSION VAL(1)
C  CHECK THAT MORE THAN 100 PUBLIC LISTS NOT BEING USED
      IF(NUM.LT.1.OR.NUM.GT.100) GO TO 2
      DO 1 I=1,NUM
C  PUSH VALUES IN ARRAY VAL DOWN ON FIRST NUM PUBLIC LISTS
    1 IDUM=NEWTOP(VAL(I),W(I))
      FNH   =VAL(1)
      RETURN
C  NUM OUT OF RANGE 1 TO 100 : ERROR MESSAGE
    2 CALL TRCOFF
      PRINT 899
  899 FORMAT(' SECOND ARGUMENT OF CALL TO PARMTN NOT AN INTEGER IN THE R
     -ANGE 1 TO 100'/'  - NO STORAGE OF VALUES ON W(I) & PARMTN SET TO Z
     -ERO')
      FNH   =0.0
      RETURN
C-----------------------------------------------------------------------
      ENTRY RDLSTA(Z)
C  CRDBUF IS BUFFER INTO WHICH CARD IMAGES ARE INSERTED
      DIMENSION CRDBUF(12)
C  CP IS AN ARRAY HOLDING MASKS TO ACCESS ALL SIXTHS OF A COMPUTER WORD
      DIMENSION CP(6) /0770000000000,07700000000,077000000,0770000,
     - 07700,077/
      INTEGER WORD,BLANK,SYMBOL,LP,RP,PLACE
C  LP, RP, BLANK ARE RIGHT JUSTIFIED FIELDDATA CONSTANTS
      DATA BLANK ,LP,RP/'     ','     (','     )'/
  900 FORMAT(12A6)
  901 FORMAT(1X,12A6)
C  CREATE A STACK LIST TO KEEP TRACK OF LIST STRUCTURE TO BE INPUT
      IDUM=LIST(STACK)
C  INITIALISE CHARACTER COUNT. WHEN IT REACHES 6 A NODE HAS BEEN
C  FILLED WITH SIX FIELDDATA CHARACTERS.
      IS=1
C  INITIALISE VARIABLE WORD TO BLANKS
      WORD=BLANK
      KOUNT=0
      PLACE=BLANK
C.......................................................................
C  READ & PRINT A CARD IMAGE
    5 READ 900,CRDBUF
      PRINT 901,CRDBUF
C  IW IS CARD BUFFER WORD COUNT, IC THE CHARACTER COUNT FOR EACH
C  BUFFER WORD
C  INITIALISE IW,IC
      IW=1
```

University of Cape Town

```
    6 IC=1
C   EXTRACT CHARACTER FROM IC-TH POSITION OF WORD IW OF CARD BUFFER
C   & INSERT IT RIGHT JUSTIFIED IN A BLANK WORD
    7 SYMBOL=INTGER(SQIN(CP(6),SQOUT(CP(IC),CRDBUF(IW)),PLACE))
C   IS THE LETTER A BLANK?
      IF(SYMBOL.EQ.BLANK) GO TO 17
C   IS THE LETTER A LEFT PARENTHESIS?
      IF(SYMBOL.EQ.LP) GO TO 10
C   IS THE LETTER A RIGHT PARENTHESIS?
      IF(SYMBOL.EQ.RP) GO TO 18
      GO TO 14
C.....................................................................................
C   HAS BUFFER CHARACTER COUNT REACHED END OF A WORD?
    8 IF(IC.EQ.6) GO TO 9
C   INCREMENT CHARACTER COUNT
      IC=IC+1
C   CONTINUE CHARACTER SCAN FROM CARD BUFFER
      GO TO 7
C   HAS WHOLE BUFFER BEEN SCANNED - IF SO READ ANOTHER CARD
    9 IF(IW.EQ.12) GO TO 5
C   GO TO NEXT WORD OF CARD BUFFER
      IW=IW+1
C   CONTINUE CHARACTER SCAN
      GO TO 6
C.....................................................................................
C   IS LEFT PARENTHESIS WHICH WAS ENCOUNTERED THE FIRST OF THE LIST?
C   IF NOT GO TO STATEMENT 11
   10 IF(KOUNT.NE.0) GO TO 11
C   CREATE NEW LIST & PUT REFERENCE NODE TO IT ON STACK
      IDUM=NXTRGT(LIST(NEW),STACK)
      KOUNT=1
C   REENTER ROUTINE RECURSIVELY
      DUMY=VISIT($8,-1)
C   ON RETURN TO THIS LEVEL OF RECURSION REMOVE TOP REFERENCE ON STACK
C   & RETURN DATUM STORED THERE (WHICH POINTS DIRECTLY TO LIST STRUCTURE
C   CREATED).
      FNH    =POPTOP(STACK)
C   DELETE THE STACK
      IDUM=MTLIST(STACK)
      CALL RCELL(STACK)
      RETURN
C.....................................................................................
C   LEFT PARENTHESIS ENCOUNTERED SO NEW SUBLIST REQUIRED
   11 IF(WORD.EQ.BLANK) GO TO 12
C   LEFT JUSTIFY CHARACTERS ALREADY IN WORD BEFORE L.P. ENCOUNTERED
C   & INSERT AT TOP OF LIST REFERRED TO AT TOP OF STACK
      IDUM=NXTLFT(LANORM(WORD),TOP(STACK))
C   REINITIALISE WORD & IS.
      WORD=BLANK
      IS=1
C   RECURSIVELY REENTER ROUTINE AT STATEMENT 13
   12 DUMY=VISIT($13,-1)
C   ON RETURN FROM DEEPER RECURSION LEVEL POP UP STACK, REVERTING A LEVEL
C   IN THE LIST STRUCTURE
      IDUM=INTGER(POPTOP(STACK))
C   CONTINUE SCAN
      GO TO 8
```

University of Cape Town

```
C   CREATE NEW LIST & INSERT ITS NAME AT TOP OF LIST REFERRED TO AT
C   TOP OF STACK
 13 IDUM=NXTLFT(LIST(NEW),TOP(STACK))
C   INSERT NEW LIST NAME ON TOP OF STACK
    IDUM=NXTRGT(NEW,STACK)
C   CONTINUE SCAN
    GO TO 8
C.......................................................................
C   SYMBOL ENCOUNTERED SO PLACE IT IN WORD ON LEFT SHIFTING
C   CHARACTERS ALREADY THERE LEFTWARDS
 14 WORD=INTGER(SHIN(6,SYMBOL,WORD))
C   IS WORD FULL YET?
    IF(IS.EQ.6) GO TO 15
C   CONTINUE SCAN
    IS=IS+1
    GO TO 8
C   WORD IS FULL SO PUT IT ON LIST STRUCTURE & REINITIALISE IT
 15 IS=1
 16 IDUM=NXTLFT(LANORM(WORD),TOP(STACK))
    WORD=BLANK
C   CONTINUE SCAN
    IS=1
    GO TO 8
C.......................................................................
C   BLANK ENCOUNTERED AS DELIMITER SO CONTINUE SCAN IF WORD EMPTY
 17 IF(WORD.EQ.BLANK) GO TO 8
C   WORD NOT EMPTY SO INSERT IN LIST STRUCTURE
    GO TO 16
C.......................................................................
C   RIGHT PARENTHESIS ENCOUNTERED
 18 IF(WORD.EQ.BLANK) GO TO 19
C   INSERT CONTENTS OF WORD (LEFT JUSTIFIED) ON TOP OF LISTREFERRED
C   TO AT TOP OF STACK
    IDUM=NXTLFT(LANORM(WORD),TOP(STACK))
C   REINITIALISE 'WORD' & 'IS'
    WORD=BLANK
    IS=1
C   ASCEND A LEVEL IN THE RECURSION
 19 CALL TERM(Z,-1)
C-----------------------------------------------------------------------
    ENTRY SUBSBT(DAT,LST)
C   SUBSTITUTE DAT FOR DATUM IN NODE AT BOTTOM OF LIST LST
    FNH    =SUBST(DAT,LNKL(CONT(LST)))
    RETURN
C-----------------------------------------------------------------------
    ENTRY SUBSTP(DAT,LST)
C   SUBSTITUTE DAT FOR DATUM IN NODE AT TOP OF LIST LST
    FNH    =SUBST(DAT,LNKR(CONT(LST)))
    RETURN
    END
```

University of Cape Town

```
      FUNCTION FNI(DUMMY)
      COMMON /SLIP/ W(101),AVSL
      INTEGER FNI
      RETURN
C-----------------------------------------------------------------------
      ENTRY LSSCPY(LA)
C   APPOINT READER FOR LIST LA, CREATE NEW LIST, SAVE THE NAMES ON PUBLIC
C   LISTS & ENTER DEEPER LEVEL OF RECURSION
      FNI=  INTGER(VISIT($100,PARMT2(LRDROV(LA),LIST(9))))
C   ON RETURN TO OUTERMOST LEVEL OF RECURSION TERMINATE PROCESS
      RETURN
C   LC IS NAME OF LIST STORED ON TOP OF W(2)
  100 LC=INTGER(TOP(W(2)))
C   LR IS NAME OF READER STORED ON TOP OF W(1)
      LR=INTGER(TOP(W(1)))
C   ADVANCE LINEARLY TO RIGHT SEARCHING FOR NON-HEADER NODE
    1 X=ADVLWR(LR,K)
C   WAS TARGET OF SEARCH FOUND?
      IF(K.EQ.0) GO TO 2
C   TARGET NOT FOUND SO POP UP READER
      CALL RCELL(LR)
C   TERMINATE THIS LEVEL OF RECURSION, POP UP PUBLIC LISTS & RETURN NAME
C   OF LIST LC TO CORRESPONDING VISIT FUNCTION
      CALL TERM(LC,RESTOR(2))
C   TARGET FOUND. IS IT A SUBLIST NODE?
    2 IF(NAMTST(X).EQ.0) GO TO 3
C   COPY TARGET FOUND TO BOTTOM OF NEW COPY OF LA
      IDUM=NEWBOT(X,LC)
C   CONTINUE READER SCAN OF LIST STRUCTURE LA
      GO TO 1
C   HAVE SUBLIST CELL. CREATE READER FOR SUBLIST & CREATE A NEW LIST
C   FOR COPY OF SUBLIST. SAVE VALUES ON PUBLIC LISTS & REENTER ROUTINE
C   RECURSIVELY. ON RETURN TO THIS LEVEL OF RECURSION PLACE NAME OF
C   COPY OF SUBLIST AT BOTTOM OF COPY OF OUTER LIST
    3 IDUM=NEWBOT(VISIT($100,PARMT2(LRDROV(X),LIST(9))),TOP(W(2)))
C   CONTINUE SCAN OF LIST STRUCTURE
      GO TO 100
C-----------------------------------------------------------------------
      ENTRY LSTEQL(LA,LB)
C   APPOINT READERS FOR LA, LB LISTS , SAVE THEM ON PUBLIC LISTS
C   & ENTER DEEPER LEVEL OF RECURSION
      FNI   =INTGER(VISIT($200,PARMT2(LRDROV(LA),LRDROV(LB))))
C   ON RETURN TO OUTERMOST LEVEL OF RECURSION TERMINATE PROCESS
      RETURN
C   LRA & LRB ARE NAMES OF THE READERS OF LISTS LA & LB
  200 LRA=INTGER(TOP(W(1)))
      LRB=INTGER(TOP(W(2)))
C   ADVANCE BOTH READERS LINEARLY TO RIGHT SEARCHING FOR NONHEADER NODE
    4 XA=ADVLWR(LRA,KA)
      XB=ADVLWR(LRB,KB)
C   WAS TARGET ON LA FOUND?
      IF(KA.EQ.0) GO TO 5
C   WAS TARGET ON LB FOUND?
      IF(KB) 6,7,6
C   TARGET ON LA FOUND. WAS TARGET ON LB FOUND?
    5 IF(KB.NE.0) GO TO 7
C   TARGET ON LB FOUND. ARE CONTENTS OF FOUND CELLS ON LA, LB EQUAL?
```

University of Cape Town

```
      IF(INTGER(XA).EQ.INTGER(XB)) GO TO 4
C   IS TARGET ON LA A SUBLIST CELL?
      IF(NAMTST(XA).NE.0) GO TO 7
C   IS TARGET ON LB A SUBLIST CELL?
      IF(NAMTST(XB).NE.0) GO TO 7
C   SUBLIST CELL FOUND SO APPOINT READERS FOR SUBLISTS , SAVE THEM &
C   ENTER DEEPER LEVEL OF RECURSION
      FNI    =INTGER(VISIT($200,PARMT2(LRDROV(XA),LRDROV(XB))))
C   IF TEST WAS SUCCESSFUL CONTINUE SCAN OF THE LISTS
      IF(FNI    )7,200,7
C   POP UP BOTH READERS
    6 CALL RCELL(LRA)
      CALL RCELL(LRB)
C   TERMINATE CURRENT LEVEL OF RECURSION & RETURN 0 TO CORRESPONDING
C   VISIT FUNCTION INDICATING SUCCESSFUL TEST ON LISTS
      CALL TERM(0,RESTOR(2))
C   POP UP BOTH READERS
    7 CALL RCELL(LRA)
      CALL RCELL(LRB)
C   TERMINATE CURRENT LEVEL OF RECURSION & RETURN -1 TO CORRESPONDING
C   VISIT FUNCTION INDICATING THAT LIST STRUCTURES NOT EQUAL
      CALL TERM(-1,RESTOR(2))
C-----------------------------------------------------------------------
      ENTRY NEWVAL(AT,VAL,LST)
C   FIND MACHINE ADDRESS OF ATTRIBUTE AT ON DL OF LIST LST
      M=MADATR(AT,LST)
C   WAS ATTRIBUTE FOUND?
      IF(M.EQ.-1) GO TO 8
C   IF FOUND SUBSTITUTE VAL FOR VALUE CORRESPONDING TO AT ON DL
C   RETURN OLD VALUE AS VALUE OF FUNCTION
      FNI    =INTGER(SUBST(VAL,M+2))
      RETURN
C   AT NOT FOUND ON DL SO AT & CORRESPONDING VAL LOADED ON DL
    8 IDUM=LDATVL(AT,VAL,LST)
      FNI    =0
      RETURN
      END
```

University of Cape Town

```
      FUNCTION FNJ(DUMMY)
      INTEGER FNJ
      RETURN
C-----------------------------------------------------------------------
      ENTRY LDATVL(AT,VAL,LST)
C   IS DL FIELD OF HEADER OF LIST LST ZERO?
      IF(LNKL(CONT(LST+2)).NE.0) GO TO 1
C   DL FIELD ZERO SO CREATE DL OF LIST LST & RETURN ITS NAME
      FNJ    =LISTAV(LST)
C   PUT A NEW CELL ON TOP OF THE DL
    1 MADR=NXTLFT(0,LNKL(CONT(LST+2)))
C   ALTER LNKL FIELD OF CELL TO BE NEGATIVE
      ITEMP=-LNKL(CONT(MADR))
      CALL SETIND(-1,ITEMP,-1,MADR)
C   PUT AT & VAL IN FIELDS OF NEW NODE OF DL
      DUMY=STRIND(AT,MADR+1)
      DUMY=STRIND(VAL,MADR+2)
      RETURN
C-----------------------------------------------------------------------
      ENTRY LPURGE(LST)
C   APPOINT READER K FOR LIST LST
      K=LRDROV(LST)
C   INITIALISE COUNT
      FNJ    =0
C   ADVANCE DOWN LST STRUCTURALLY SEARCHING FOR NON-HEADER NODE
    2 Y=ADVSWR(K,J)
C   HAS TARGET BEEN FOUND?
    3 IF(J.NE.0) GO TO 4
C   IS TARGET NODE SUBLIST CELL? IF NOT CONTINUE SCAN
      IF(NAMTST(Y).NE.0) GO TO 2
C   HAS LIST POINTED TO BY CELL ALREADY BEEN TRAVERSED TO GET TO THIS
C   CELL? IF NOT CONTINUE SCAN
      IF(LSTPRO(Y,K).NE.0) GO TO 2
C   LIST HAS ALREADY BEEN TRAVERSED THROUGH
C   FIND ADDRESS OF NODE POINTED TO BY READER
      L=LPNTR(K)
C   ADVANCE ALONG LIST LINEARLY
      Y=ADVLWR(K,J)
C   DELETF SUBLIST CELL POINTED TO BY K
      DUMY=DELETE(L)
C   INCREASE DELETION COUNT (FINAL VALUE RETURNED AS VALUE OF FUNCTION
      FNJ    =FNJ    +1
C   CONTINUE SCAN OF SUBLIST
      GO TO 3
C   TARGET NOT FOUND SO DELETE READER & EXIT FROM ROUTINE
    4 IDUM=IRARDR(K)
      RETURN
      END
```

University of Cape Town

```
      FUNCTION IRALST(K)
C   ERASE LIST K & RETURN ITS NODES TO AVAIL
C   CHECK THAT K EXISTS
      L=LOCT(K)
C   REDUCE LCNTR OF K BY ONE
      CALL SETIND(-1,-1,LCNTR(L)-1,L+2)
C   RETURN LCNTR AS VALUE OF FUNCTION
      IRALST=LCNTR(L)
C   IS LEVEL COUNTER ZERO? IF NOT LIST IS STILL REFERENCED AS A SUBLIST
C   & HENCE CANNOT BE DELETED IMMEDIATELY
      IF(IRALST.NE.0) RETURN
C   EMPTY LIST K
      IDUM=MTLIST(K)
C   N HAS CONTENTS OF DL FIELD OF K
      N=LNKL(CONT(LNKR(L)+2))
C   IS THERE A DL ATTACHED TO K?
      IF(N.EQ.0) GO TO 1
C   OBTAIN NEW CELL FROM AVAIL
      NEW=NUCELL(DUMY)
C   SET UP NEW CELL AS SUBLIST CELL REFERENCING DL
      CALL SETI3(-1,-1,1,-1,-1,-1,N,N,NEW)
C   PUT SUBLIST CELL REFERRING TO DL ON AVAIL SO THAT NODES OF DL CAN
C   BE USED BY STORAGE RECLAMATION WHEN NECESSARY (LCNTR OF DL ZERO)
      CALL RCELL(NEW)
C   DELETE HEADER OF K
    1 CALL RCELL(L)
      RETURN
      END
```

University of Cape Town

```
      FUNCTION NUCELL(DUMMY)
      COMMON /SLIP/ W(101),AVSL
C   M IS POINTER TO TOP OF AVAIL
      M=LNKR(AVSL)
      IF(M.NE.0) GO TO 1
      CALL TRCOFF
C  LIST OF AVAILABLE SPACE (AVAIL) EXHAUSTED IF NEXT CELL OF AVAIL
C  HAS ZERO LNKR FIELD
      PRINT 901
901   FORMAT('1    LIST OF AVAILABLE SPACE EXHAUSTED.')
      CALL WLKBCK(1)
C  IS CELL TAKEN OFF AVAIL A REFERENCE TO A SUBLIST?
    1 IF(ID(M).NE.1) GO TO 2
C  CELL WAS SUBLIST REFERENCE SO APPLY IRALST TO DELETE LIST REFERENCED
C  OR TO DECREASE NUMBER OF REFERENCES TO IT AS THIS SUBLIST NODE HAD
C  BEEN RESTORED TO AVAIL
      IDUM=IRALST(CONT(M+2))
C  READJUST LIMITS OF AVAIL TO PRECLUDE CELL TAKEN FROM IT
    2 CALL SETDIR(-1,-1,LNKR(CONT(M)),AVSL)
C  ZEROISE NEW NODE BEING DELIVERED FROM AVAIL
      DUMY=STRIND(STRIND(STRIND(0,M),M+1),M+2)
C  RETURN ADDRESS OF NEW NODE
      NUCELL=M
      RETURN
      END
```

University of Cape Town

```
.                  LISTING OF ASSEMBLER-CODED PRIMITIVE ROUTINES
.                  ---------------------------------------------
. NOTE: INDEX REGISTER X11 IS USED TO CARRY OVER THE ADDRESSES OF
.          OF THE ARGUMENTS OF A FUNCTION CALL FROM FORTRAN & THE
.          RETURN ADDRESS, SINCE THE ASSEMBLER EQUIVALENT OF A FUNCTION
.          OR SUBROUTINE CALL IN FORTRAN IS:
.                        LMJ   X11,XXXXXX
.                        +     ADDRESS OF ARG1
.                        +     ADDRESS OF ARG2
.                                .
.                                .
.                                .
.                        +     ADDRESS OF ARGN
.                        +     WALKBACK WORD
.          UPON ENTRY INTO XXXXXX, X11 HAS IN ITS H2 FIELD
.          THE ADDRESS OF THE INSTRUCTION FOLLOWING THE LMJ, I.E. IT
.          CONTAINS THE ADDRESS OF THE FIRST ARGUMENT OF THE FUNCTION
.          CALL. THE RETURN ADDRESS FROM THE CALL IS THAT OF THE
.          INSTRUCTION FOLLOWING THE WALKBACK WORD, I.E. THE ADDRESS
.          IN THE MODIFIER (H2) FIELD OF X11 PLUS M+1, WHERE M IS
.          THE NUMBER OF ARGUMENTS OF THE FUNCTION CALL. HENCE A JUMP
.          TO M+1 AS MODIFIED BY X11 IS REQUIRED TO RETURN FROM THE
.          FUNCTION.
.          ACCUMULATOR REGISTER A0 (WHICH OVERLAPS WITH X9) IS USED
.          TO CARRY BACK THE VALUE OF A FUNCTION CALL TO THE
.          CALLING PROGRAM, SINCE IF THE LMJ HAD BEEN EQUIVALENT
.          TO A FUNCTION RATHER THAN A SUBROUTINE CALL, THE INSTRUCTION
.          FOLLOWING THE WALKBACK WORD WOULD BE A STORE INSTRUCTION
.          TO STORE THE CONTENTS OF A0 SOMEWHERE IN THE CALLING
.          PROGRAM
.
. ---------------------------------------------------------------------------
           AXR$                            .
           INFO        4 2                 . LINKAGE TO
$(2),BLNK  RES         1                   .         BLANK COMMON
$(0),REGS  RES         4                   . TEMPORARY REGISTER STORAGE
WB         +           'PRMTVS'            . WALKBACK
           +           0,$-$               .         PACKAGE
TEMP       +           0                   . TEMPORARY STORAGE
. ---------------------------------------------------------------------------
$(1),CONT* NOP         A0,0                . CONT, INHALT IDENTICAL
INHALT*    LA,H2       A0,0,X11            . LOAD ADDRESS CONTAINED
           LA,H2       A0,0,A0             .    AS 1ST ARGUMENT INTO A0
           LA          A0,0,A0             . LOAD CONTENTS OF ADDRESS
           J           2,X11               . RETURN
. ---------------------------------------------------------------------------
GETFLG*    LA,H2       A0,0,X11            .
           LA,H2       A0,0,A0             .
           LA,S3       A0,1,A0             . LOAD 'FLAG' FIELD OF ADDRESS
           J           2,X11               .
. ---------------------------------------------------------------------------
GETHAF*    LA,H2       A0,0,X11            .
           LA,H2       A0,0,A0             .
           LA,H2       A0,1,A0             . LOAD 'HALF' FIELD OF ADDRESS
           J           2,X11               .
. ---------------------------------------------------------------------------
GETMRK*    LA,H2       A0,0,X11            .
```

University of Cape Town

```
          LA,H2      A0,0,A0                  .
          LA,S2      A0,1,A0                  . LOAD 'MARK' FIELD OF ADDRESS
          J          2,X11                    .
. ------------------------------------------------------------------------------
ID*       LA,H2      A0,0,X11                 .
          LA,H2      A0,0,A0                  .
          LA,S1      A0,1,A0                  . LOAD 'ID' FIELD OF ADDRESS
          J          2,X11                    .
. ------------------------------------------------------------------------------
INTGER*   NOP        A0,0                     . INTGER, REALNO ARE IDENTICAL
REALNO*   LA,H2      A0,0,X11                 . LOAD THE PARAMETER
          LA         A0,0,A0                  .    INTO REGISTER A0
          J          2,X11                    . RETURN WITH VALUE IN A0
. ------------------------------------------------------------------------------
LANORM*   LA,H2      A0,0,X11                 . LOAD ALPHAMERIC WORD
          LA         A0,0,A0                  .    INTO REGISTER A0
          TNE        A0,('         ')         . IF WORD IS FILLED
          J          2,X11                    .    WITH BLANKS, RETURN
          SA          A1,REGS                 . SAVE REGISTER A1
          SA         A0,TEMP                  . LOOP: STORE WORD
          LA,S1      A1,TEMP                  . LOAD RIGHTMOST CHARACTER
          TE         A1,(05)                  . IF THIS A BLANK
          J          $+3                      .    EXIT FROM LOOP
          LSSC       A0,6                     . CIRCULAR SHIFT CHARS. LEFTWARD
          J          $-5                      . CONTINUE LOOP
          LA          A1,REGS                 . OUT OF LOOP: RESTORE A1
          J           2,X11                   . RETURN
. ------------------------------------------------------------------------------
LNKL*     LA,H2      A0,0,X11                 . LOAD 1ST PARAMETER INTO A0
          LA,H1      A0,0,A0                  . OBTAIN LNKL FIELD OF A0
          J          2,X11                    .
. ------------------------------------------------------------------------------
LNKLP*    LA,H2      A0,0,X11                 .
          LMA,H1     A0,0,A0                  . OBTAIN MOD(LNKL) FIELD OF A0
          J          2,X11                    .
. ------------------------------------------------------------------------------
LNKR*     LA,H2      A0,0,X11                 .
          LA,H2      A0,0,A0                  . OBTAIN LNKR FIELD OF A0
          J          2,X11                    .
. ------------------------------------------------------------------------------
MADOV*    LA,H2      A0,0,X11                 . OBTAIN ADDRESS OF PARAMETER
          J          2,X11                    . RETURN WITH ADDRESS IN A0
. ------------------------------------------------------------------------------
PUTFLG*   LA,H2      A0,1,X11                 . LOAD ADDRESS CONTAINED
          LA,H2      A0,0,A0                  .    AS 2ND ARGUMENT INTO A0
          SA          A1,REGS                 . SAVE CONTENT OF A1
          LA,H2      A1,0,X11                 . LOAD 1ST ARGUMENT
          LA         A1,0,A1                  .    INTO REGISTER A1
          SA,S3      A1,1,A0                  . STORE IN 'FLAG' FIELD
          LA          A1,REGS                 . RESTORE A1
          J          3,X11                    . RETURN
. ------------------------------------------------------------------------------
PUTHAF*   LA,H2      A0,1,X11                 . SIMILAR TO 'PUTFLG'
          LA,H2      A0,0,A0                  .
          SA          A1,REGS                 .
          LA,H2      A1,0,X11                 .
          LA         A1,0,A1                  .
```

University of Cape Town

```
         SA,H2      A1,1,A0          . STORE IN 'HALF' FIELD
         LA          A1,REGS         .
         J          3,X11            .
. ------------------------------------------------------------------
PUTMRK*  LA,H2      A0,1,X11         . SIMILAR TO 'PUTFLG'
         LA,H2      A0,0,A0          .
         SA          A1,REGS         .
         LA,H2      A1,0,X11         .
         LA         A1,0,A1          .
         SA,S2      A1,1,A0          . STORE IN 'MARK' FIELD
         LA          A1,REGS         .
         J          3,X11            .
. ------------------------------------------------------------------
SETDIR*  LA,H2      A0,3,X11         . LOAD 4TH PARAMETER (ADDRESS)
ENT1     SA          A1,REGS         . SAVE REGISTER A1
         LA,H2      A1,1,X11         . LOAD INTO REGISTER A1
         LA         A1,0,A1          .     SECOND PARAMETER
         TE         A1,(-1)          . IF -1 DO NOT STORE IT
         SA,H1      A1,0,A0          . STORE IN 'LNKL' OF ADDRESS
         LA,H2      A1,2,X11         . THIRD
         LA         A1,0,A1          .     PARAMETER
         TE         A1,(-1)          .
         SA,H2      A1,0,A0          . STORE IN 'LNKR' OF ADDRESS
         LA,H2      A1,0,X11         . FIRST
         LA         A1,0,A1          .     PARAMETER
         TE         A1,(-1)          .
         SA,S1      A1,1,A0          . STORE IN 'ID' OF ADDRESS
         LA          A1,REGS         . RESTORE REGISTER A1
         J          5,X11            . RETURN
. ------------------------------------------------------------------
SETIND*  LA,H2      A0,3,X11         . LOAD 4TH PARAMETER
         LA,H2      A0,0,A0          .   (INDIRECT ADDRESS)
         J           ENT1            . PROCEED TO CODING AT ENT1
. ------------------------------------------------------------------
SETD3*   LA,H2      A0,8,X11         . LOAD 9TH PARAMETER (ADDRESS)
ENT2     SA          A1,REGS         . SAVE REGISTER A1
         LA,H2      A1,0,X11         . LOAD INTO REGISTER A1
         LA         A1,0,A1          .     FIRST PARAMETER
         TE         A1,(-1)          . IF -1 DO NOT STORE IT
         SA,H1      A1,0,A0          . STORE IN 'LNKL' OF ADDRESS
         LA,H2      A1,1,X11         . SECOND
         LA         A1,0,A1          .     PARAMETER
         TE         A1,(-1)          .
         SA,H2      A1,0,A0          . STORE IN 'LNKR' OF ADDRESS
         LA,H2      A1,2,X11         . THIRD
         LA         A1,0,A1          .     PARAMETER
         TE         A1,(-1)          .
         SA,S1      A1,1,A0          . STORE IN 'ID' OF ADDRESS
         LA,H2      A1,3,X11         . FOURTH
         LA         A1,0,A1          .     PARAMETER
         TE         A1,(-1)          .
         SA,S2      A1,1,A0          . STORE IN 'MARK' OF ADDRESS
         LA,H2      A1,4,X11         . FIFTH
         LA         A1,0,A1          .     PARAMETER
         TE         A1,(-1)          .
         SA,S3      A1,1,A0          . STORE IN 'FLAG' OF ADDRESS
         LA         A1,5,X11         . SIXTH
```

University of Cape Town

```
         LA          A1,0,A1          .    PARAMETER
         TE          A1,(-1)          .
         SA,H2       A1,1,A0          . STORE IN 'HALF' OF ADDRESS
         LA,H2       A1,6,X11         . SEVENTH
         LA          A1,0,A1          .    PARAMETER
         TE          A1,(-1)          .
         SA,H1       A1,2,A0          . STORE IN 'LPNTR' OF ADDRESS
         LA,H2       A1,7,X11         . EIGHTH
         LA          A1,0,A1          .    PARAMETER
         TE          A1,(-1)          .
         SA,H2       A1,2,A0          . STORE IN 'LCNTR' OF ADDRESS
         LA           A1,REGS         . RESTORE REGISTER A1
         J           10,X11           . RETURN
. ---------------------------------------------------------------------
SETI3*   LA,H2       A0,8,X11         . LOAD 9TH PARAMETER
         LA,H2       A0,0,A0          .    (INDIRECT ADDRESS)
         J            ENT2            . PROCEED TO CODING AT ENT2
. ---------------------------------------------------------------------
SHIN*    LA,H2       A0,0,X11         . LOAD 1ST ARGUMENT
         LA          A0,0,A0          .    INTO REGISTER A0
         SA          A0,TEMP          . SAVE ARGUMENT
         LA,H2       A0,2,X11         . LOAD 3RD ARGUMENT
         LA          A0,0,A0          .    INTO REGISTER A0
         LSSL        A0,*TEMP         . SHIFT 3RD ARG. LEFT 'N' BITS
         SA           A1,REGS         . SAVE
         SA           A2,REGS+1       .     REGISTERS
         SR           R2,REGS+2       .              USED
         LA,H2       A1,1,X11         . LOAD 2ND ARGUMENT
         LA          A1,0,A1          .    INTO REGISTER A1
         SZ          A2               . OBTAIN BITS
         DSL         A1,*TEMP         .    TO BE INSERTED
         SA          A2,A1            .       IN THIRD
         LSSC        A1,*TEMP         .          ARGUMENT
         LA          A2,(-0)          . OBTAIN MASK
         LSSL        A2,*TEMP         .    FOR
         SNA         A2,TEMP          .       INSERTION
         LA          A2,TEMP          .
         LR          R2,A2            . INSERTION
         MLU         A0,A1            .    PROCESS
         SA          A1,A0            .
         LA           A1,REGS         . RESTORE
         LA           A2,REGS+2       .        REGISTERS
         LR           R2,REGS+2       .                 USED
         J           4,X11            . RETURN
. ---------------------------------------------------------------------
SQIN*    LA,H2       A0,0,X11         . LOAD 1ST ARGUMENT
         LA          A0,0,A0          .    INTO REGISTER A0
         SA           A1,REGS         . SAVE
         SA           A2,REGS+1       .     REGISTERS
         SR           R2,REGS+2       .              USED
         SX           X6,REGS+3       .
         LX,U        X6,0             . SHIFTING OF 2ND ARGUMENT
         LA,U        A1,1             .    DETERMINED FROM MASK:
         AND         A1,A0            .
         TE,U        A2,0             . TEST FOR TRAILING ZERO BIT
         J           $+4              . EXIT FROM LOOP
         AX          X6,(1)           .
```

```
         SSC       A0,1                .
         J         $-5                 . CONTINUE LOOPING
         SX        X6,TEMP             . COUNT OF NO. OF BITS TO SHIFT
         LSSC      A0,*TEMP            .
         LA,H2     A1,1,X11            . LOAD & LEFTSHIFT
         LA        A1,0,A1             .    SECOND
         LSSC      A1,*TEMP            .       ARGUMENT
         LR        R2,A0               . INSERT 2ND ARGUMENT
         LA,H2     A0,2,X11            .    INTO CORRECT
         LA        A0,0,A0             .       FIELD OF
         MLU       A0,A1               .          THIRD ARGUMENT
         SA        A1,A0               . RESULT IS VALUE OF FUNCTION
         LA         A1,REGS            . RESTORE
         LA         A2,REGS+1          .       REGISTERS
         LR         R2,REGS+2          .                  USED
         LX         X6,REGS+3          .
         J         4,X11               . RETURN
. -----------------------------------------------------------------------
SQOUT*   LA,H2     A0,0,X11            . LOAD MASK
         LA        A0,0,A0             .    INTO REGISTER A0
         SA         A1,REGS            . SAVE
         SA         A2,REGS+1          .     REGISTERS
         SX         X6,REGS+2          .              USED
         LX,U      X6,0                . COUNT TRAILING ZEROS
         LA,U      A1,1                .       OF MASK
         AND       A1,A0               .
         TE,U      A2,0                . TEST FOR TRAILING ZERO BIT
         J         $+4                 . EXIT FROM LOOP
         AX,U      X6,1                .
         SSC       A0,1                .
         J         $-5                 . CONTINUE LOOPING
         SX        X6,TEMP             . COUNT OF NO. OF BITS TO SHIFT
         LSSC      A0,*TEMP            . LEFTSHIFT MASK BACK AGAIN
         LA,H2     A1,1,X11            . LOAD 2ND ARGUMENT
         LA        A1,0,A1             .    INTO REGISTER A1
         AND       A1,A0               . OBTAIN FIELD SPECIF'D BY MASK
         SSL       A2,*TEMP            . REMOVE TRAILING ZERO BITS
         SA        A2,A0               . RESULT IS VALUE OF FUNCTION.
         LA         A1,REGS            . RESTORE
         LA         A2,REGS+1          .       REGISTERS
         LX         X6,REGS+2          .                USED
         J         3,X11               . RETURN
. -----------------------------------------------------------------------
STRDIR*  SA         A1,REGS            . SAVE REGISTER A1
         LA,H2     A1,1,X11            . LOAD PAR. 2 (ADDRESS) INTO A1
ENT3     LA,H2     A0,0,X11            . LOAD FIRST
         LA        A0,0,A0             .    PARAMETER INTO A0
         SA        A0,0,A1             . STORE IN ADDRESS.
         LA         A1,REGS            . RESTORE REGISTER A1
         J         3,X11               . RETURN
. -----------------------------------------------------------------------
STRIND*  SA         A1,REGS            . SAVE REGISTER A1
         LA,H2     A1,1,X11            . LOAD PAR. 2 (ADDRESS)
         LA,H2     A1,0,A1             .    INTO A1 (INDIRECT)
         J          ENT3               . PROCEED TO CODING AT ENT3
. -----------------------------------------------------------------------
TRCON*   SLJ       TON$                . SWITCH ON
```

```
          J          1,X11                 .   ASSEMBLER TRACE
. -----------------------------------------------------------------------
TRCOFF*   SLJ        TOFF$                 . SWITCH OFF
          J          1,X11                 .   ASSEMBLER TRACE
. -----------------------------------------------------------------------
WLKBCK*   LA,H2      A0,1,X11              .   INVOKE
          LA,H2      A1,0,X11              .
          LA         A1,0,A1               .
          LR         R3,0,A0               .
          LX         X11,1,A0              .        WALKBACK
          SA         A1,$+2                .
          SLJ        NERR$                 .
          +          0                     .              PRINTOUT
. -----------------------------------------------------------------------
          END                              .
```

University of Cape Town

```
.                               RECURSION ROUTINES
.                               ------------------
. SEE DESCRIPTION OF RECURSION ROUTINES IN SECTION 2.3.2.6 OF TEXT
.
. ------------------------------------------------------------------------
           AXR$                                  .
           INFO          4 2                     . LINKAGE TO
$(2),BLNK  RES           1                       .    BLANK COMMON
SLIP       INFO          2 3                     . LINKAGE TO NAMED COMMON SLIP
$(3),X     RES           102                     .    WITH PUBLIC LISTS & AVSL
$(0),WTERM +             'TERM  '                . WALKBACK PACKAGE
           +             0,$-$                   .    FOR SUBROUTINE TERM
WVISIT     +             'VISIT '                . WALKBACK PACKAGE
           +             0,$-$                   .    FOR FUNCTION VISIT
TEMP       +             0                       . TEMPORARY STORAGE TO SAVE A1
JUMP       +             0                       . STORAGE FOR JUMP INSTRUCTION
. ------------------------------------------------------------------------
$(1),TERM* NOP           A0,0                    . TERMINATE RECURSIVE CODING
           SX,H2         X11,WTERM+1             . WALKBACK REGISTER-SAVE
           LMJ           X11,POPTOP              . EQUIVALENT
           +             X+100                   .    TO FUNCTION CALL
           +             $-2,WTERM               .       POPTOP(X(101))
           SA,H2         A0,A1                   . SAVE VALUE OF POPTOP CALL
           LA,H2         A0,0,X11                . LOAD 1ST ARGUMENT WHICH IS
           LA            A0,0,A0                 .    VALUE OF CORR. VISIT CALL
           LX            X11,A1                  . LOAD RETURN ADDRESS INTO X11
           LA            A1,TEMP                 . RESTORE FROM CORR. VISIT CALL
           J             3,X11                   . RETURN TO CORR. VISIT CALL
. ------------------------------------------------------------------------
VISIT*     SA            A1,TEMP                 . SAVE REGISTER A1
           SX,H2         X11,WVISIT+1            . WALKBACK REGISTER-SAVE
           LA            A1,0,X11                . STORE LABEL OF RECURSIVE BLOCK
           SA            A1,JUMP                 .    OF CODING TO BE ENTERED
           LA,U          A1,WVISIT+1             . LOAD RETURN ADDRESS AND
           SA,H2         A1,$+2                  .    STORE IN CALL ON NEWTOP
           LMJ           X11,NEWTOP              . EQUIVALENT TO
           +             0                       .    THE FUNCTION CALL
           +             X+100                   .       NEWTOP(RETURN ADDRESS,
           +             $-3,WVISIT.             .              X(101))
           EX            JUMP                    . JUMP TO THE RECURSIVE CODING
. ------------------------------------------------------------------------
           END                                   .
```

University of Cape Town

```
      SUBROUTINE SRB
      COMMON /SLIP/ X(101),AVSL
      RETURN
C-----------------------------------------------------------------------
      ENTRY DERROR(N)
C  ERROR MESSAGE ROUTINE FOR DL ROUTINES
      CALL TRCOFF
      PRINT 900,N
      PRINT 901
      RETURN
  900 FORMAT(///' LIST-NAME = ',O12/)
  901 FORMAT(' ATTRIBUTE-VALUE LIST REQUIRED BUT NOT FOUND')
C-----------------------------------------------------------------------
      ENTRY INITAS(M,N)
C  INITIALISATION ROUTINE OF SLIP
C  X ARRAY HOLDS NAMES OF PUBLIC LISTS, AVSL THE STORAGE LIMITS OF
C  AVAIL. W ARRAY HOLDS HEADER NODES OF PUBLIC LISTS & M IS THE ARRAY
C  DIMENSIONED N TO HOLD THE LIST OF AVAILABLE SPACE, AVAIL
      DIMENSION W(303),M(1)
C  CREATE THE PUBLIC LISTS & THEIR NAMES
      DO 1 I=1,101
      J=3*I-2
      DUM=STRDIR(STRDIR(STRDIR(STRDIR(0,X(I)),W(J)),W(J+1)),W(J+2))
      CALL SETDIR(-1,MADOV(W(J)),MADOV(W(J)),X(I))
    1 CALL SETD3(MADOV(W(J)),MADOV(W(J)),2,-1,-1,-1,-1,4095,W(J))
C  ZEROISE THE M ARRAY
      DO 2 I=1,N
    2 M(I)=0
C  ROUND OFF N SO THAT DIVISIBLE BY 3
      N=N/3
      N=3*N
      K=N-3
C  LINK TOGETHER AVAIL FROM M ARRAY. LAST 3 WORDS OF M REMAIN ZERO
C  SO THAT HAVE END MARKER TO INDICATE THAT AVAIL IS EXHAUSTED
      DO 3 I=1,K,3
    3 CALL SETDIR(-1,-1,MADOV(M(I+3)),M(I))
C  SET UP AVAIL LIMITS IN AVSL, THE QUASI-NAME OF AVAIL
      CALL SETDIR(-1,MADOV(M(N-2)),MADOV(M(1)),AVSL)
      RETURN
C-----------------------------------------------------------------------
      ENTRY RCELL(CELL)
C  RETURN NODE OF WHICH CELL IS THE ADDRESS TO AVAIL
C  LINK CELL TO BOTTOM OF AVAIL
      CALL SETIND(-1,-1,CELL,LNKL(AVSL))
C  CHANGE AVSL TO INCORPORATE CELL IN AVAIL LIMITS
      CALL SETDIR(-1,CELL,-1,AVSL)
C  MAKE CELL THE END MARKER FOR AVAIL
      CALL SETIND(-1,-1,0,CELL)
      RETURN
      END
```

University of Cape Town

```
      SUBROUTINE SRF
      COMMON /SLIP/ W(101),AVSL
      RETURN
C-----------------------------------------------------------------------
      ENTRY LSPACE
      K=0
      I=LNKR(AVSL)
    1 IF(LNKR(I).EQ.0) GO TO 2
      I=MADRGT(I)
      K=K+1
      GO TO 1
    2 PRINT 799,K
  799 FORMAT('0 LIST OF AVAILABLE SPACE CONTAINS ',I5,' NODES')
      RETURN
C-----------------------------------------------------------------------
      ENTRY LSTRCE(LST,LSTNAM,N)
C  PRINT OUT NODES OF LIST LST IN OCTAL & ALPHAMERIC FORMAT
C   IF N=0 OCTAL FORMAT ONLY
C   IF N=1 HEADER & SUBLIST NODES IN OCTAL & REMAINDER IN ALPHA FORMAT
      DIMENSION ICONT(3)
  800 FORMAT(/1H ,60('-'))
  801 FORMAT(/10X,'NAME OF LIST IS ',A6,', = ',O12)
  802 FORMAT(//' NODE ADDRESS',21X,'CONTENTS OF NODE'/)
  803 FORMAT(1H0,3X,O6,11X,2(O12,3X),3X,A6)
  804 FORMAT(1H0,3X,O6,8X,3(3X,O12))
      L=LOCT(LST)
      CALL TRCOFF
      PRINT 800
      PRINT 801,LSTNAM,L
      PRINT 802
      J=LNKR(L)
      DO 3 K=1,3
    3 ICONT(K)=INHALT(J+K-1)
      PRINT 804,J,ICONT
    4 J=MADRGT(J)
C  HAVE REACHED LIST HEADER YET?
      IF(ID(J).EQ.2) GO TO 7
      DO 5 K=1,3
C  PUT CONTENTS OF NODE (3 WORDS) INTO ARRAY ICONT
    5 ICONT(K)=INHALT(J+K-1)
C  IS NODE TO BE PRINTED IN ALPHAMERIC FORMAT?
      IF(N.EQ.1.AND.ID(J).EQ.0) GO TO 6
      PRINT 804,J,ICONT
      GO TO 4
    6 PRINT 803,J,ICONT
      GO TO 4
    7 PRINT 800
      RETURN
C-----------------------------------------------------------------------
      ENTRY PRESRV(N)
C  PUSH THE VALUES STORED ON TOP OF FIRST N PUBLIC LISTS AGAIN ON
C  TOP OF THEM
      DO 10 I=1,N
   10 IDUM=NEWTOP(TOP(W(I)),W(I))
      RETURN
C-----------------------------------------------------------------------
      ENTRY PRLSTS(OUTLST,II)
```

University of Cape Town

```
C  OUTLST IS LIST TO BE PRINTED. FORMAT OF OUTPUT IS SPECIFIED BY II
      EQUIVALENCE (KOUT,OUT)
      CALL TRCOFF
  900 FORMAT(21X,10HBEGIN LIST)
  901 FORMAT(21X,8HEND LIST)
  902 FORMAT(21X,I14)
  903 FORMAT(21X,13HBEGIN SUBLIST)
  904 FORMAT(21X,11HEND SUBLIST)
  905 FORMAT(21X,A6)
  906 FORMAT(21X,F10.4)
  907 FORMAT(21X,13HEMPTY SUBLIST)
      PRINT 900
C  APPOINT READER LR FOR OUTLST
      LR=LRDROV(OUTLST)
      LEVEL=0
C  ADVANCE STRUCTURALLY DOWN OUTLST
   20 X=ADVSWR(LR,K)
C  HAVE REACHED HEADER NODE OF MAIN LIST YET?
      IF(K.NE.0) GO TO 110
C  COMPARE 'LEVEL' & THE LEVEL IN STRUCTURE AT WHICH LR IS
   30 IF(LEVEL-LCNTR(LR)) 140,40,100
C  IS THE NODE POINTED TO BY LR A SUBLIST NODE?
   40 IF(NAMTST(X).NE.0) GO TO 60
C  IS THE LIST REFERENCED BY SUBLIST NODE EMPTY?
      IF(LISTMT(X).NE.0) GO TO 50
C  EMPTY SUBLIST
      PRINT 907
C  CONTINUE TRAVERSAL OF OUTLST
      GO TO 20
   50 PRINT 903
C  DESCEND A LEVEL IN THE OUTLST LIST
      LEVEL=LEVEL+1
C  CONTINUE SCAN
      GO TO 20
C  NODE POINTED TO BY LR A DATUM NODE. PRINT OUT DATUM IN APPROPRIATE
C  FORMAT & THEN CONTINUE SCAN
   60 IF(II.LT.1.OR.II.GT.3) II=3
      GO TO (70,80,90),II
   70 OUT=X
      PRINT 902,KOUT
      GO TO 20
   80 OUT=X
      PRINT 905,KOUT
      GO TO 20
   90 PRINT 906,X
      GO TO 20
C  READER HAS ASCENDED OUT OF A SUBLIST
  100 PRINT 904
C  PRINT END OF SUBLIST MESSAGE & DECREMENT LEVEL
      LEVEL=LEVEL-1
C  CHECK IF BACK IN MAIN LIST YET (STATEMENT 30)
      GO TO 30
C  HAVE REACHED HEADER OF MAIN LIST. CHECK IF END OF SUBLIST MESSAGES
C  BALANCE BEGINNING OF SUBLIST MESSAGES
  110 IF(LEVEL-LCNTR(LR)) 140,130,120
C  STILL HAVE END OF SUBLIST MESSAGES TO PRINT OUT
  120 PRINT 904
```

University of Cape Town

APPENDIX B

DERIV: FORTRAN LISTINGS

University of Cape Town

```
      FUNCTION ALGDIF(LST,DIFVAR,NAMEDF)
C   DIFFERENTIATION ROUTINE. SEE FLOWCHART IN APPENDIX C.
C   FLOWCHARTS OF SIMPLIFICATION PROCEDURES OF BASIC OPERATORS
C   GIVEN IN FIGURES 11 THROUGH 15 OF TEXT
C
C   LST IS LIST NAME OF EXPRESSION TO BE DIFFERENTIATED
C   DIFVAR IS HOLLERITH NAME OF VARIABLE OF DIFFERENTIATION
C   NAMEDF IS HOLLERITH TAG TO BE ATTACHED TO DERIVATIVE LIST
C
C   CP ARRAY HOLDS MASKS FOR SIXTH FIELDS OF A WORD
C   OPRS IS THE OPERATOR TABLE
C   IPTR & LNGTH ARRAYS HOLD POINTERS & LENGTHS OF OPERANDS
C   OF OPERATOR AT ANY STAGE
C   ICAT ARRAY HOLDS 'CATEGORY' OF OPERANDS: IF 1 A CONSTANT
C   IF 2 THE VARIABLE OF DIFFERENTIATION & IF 3 AN EXPRESSION
      COMMON /SLIP/ W(100)
      COMMON /IODIFN/CP(6),DUMMY(22),OPRS(40)
      INTEGER OPRS,IOPRS/32/
      DIMENSION IPTR(2),LNGTH(2),ICAT(2)
C   INODE IS POINTER IN INPUT EXPRESSION LST
      INODE=LNKR(LOCT(LST))
C   CREATE DERIVATIVE LIST, RETURNING ITS NAME AS FUNCTION VALUE
      ALGDIF=REALNO(LIST(LSTDIF))
C   INODED IS POINTER IN DERIVATIVE LIST
      INODED=LNKR(LSTDIF)
      I=LNKR(W(100))
C   SEARCH COLLATOR LIST FOR TAG OF DERIVATIVE LIST
5     I=MADRGT(I)
      IF(ID(I).EQ.2) GO TO 6
      IF(NAMEDF.EQ.NDAT(I)) GO TO 7
      GO TO 5
C   TAG NOT FOUND ON COLLATOR LIST SO ADD A NODE TO IT
6     I=NEWNOD(0,0,NAMEDF,I)
      GO TO 8
C   TAG ALREADY ON COLLATOR LIST SO OVERWRITE CONTENTS OF THIS NODE
7     DUMY=STRIND(NAMEDF,I+2)
C   INSERT POINTER TO DERIVATIVE IN RELEVANT CELL OF COLLATOR LIST
8     DUMY=PUTHAF(INODED,I)
C   INSERT VARIABLE OF DIFFERENTIATION & POINTER TO DERIVATIVE
C   IN THE DL OF LIST LST
      IDUM=NEWVAL(DIFVAR,INODED,LST)
C   START SCAN THROUGH LIST LST
      INODE=MADRGT(INODE)
C   ENTER ROUTINE RECURSIVELY
      DUMY=VISIT($10,-1)
C   EXIT FROM FUNCTION UPON RETURN TO THIS LEVEL OF RECURSION
      RETURN
C-----------------------------------------------------------------------
C   RECOGNITION MODULE
C   IS NODE POINTED TO BY INODE OF UNIT LENGTH?
10    IF(NCODE(INODE).NE.0) GO TO 19
C   SEARCH OPERATOR TABLE TO FIND POSITION OF OPERATOR FOUND ON LST IN IT
      DO 11 IDAT=1,IOPRS
      IF(NDAT(INODE).EQ.OPRS(IDAT)) GO TO 12
11    CONTINUE
C   OPERATOR ENCOUNTERED IN LIST THAT IS NOT ON OPRS TABLE: ERROR
      CALL WLKBCK(3)
```

University of Cape Town

```
C   DELINEATE OPERANDS OF OPERATOR ENCOUNTERED, ASSIGNING THEIR
C   POINTERS, LENGTHS AND 'CATEGORIES'
12      ITEMP=INODE
        DO 13 IK=1,2
        IPTR(IK)=MADRGT(ITEMP)
13      CALL HANDLE(IPTR(IK),ITEMP,LNGTH(IK))
        IDUM=NEWTOP(ITEMP,W(99))
        DO 16 IK=1,2
        IF(LNGTH(IK).GT.1) GO TO 15
        ICODE=NCODE(IPTR(IK))
        IF(ICODE.EQ.4.AND.NDAT(IPTR(IK)).EQ.INTGER(DIFVAR)) GO TO 14
        IF(INTGER(GETHAF(IPTR(IK))).NE.0) GO TO 15
        ICAT(IK)=1
        GO TO 16
14      ICAT(IK)=2
        GO TO 16
15      ICAT(IK)=3
16      CONTINUE
C   IS OPERATOR A 'UNARY' FUNCTION WITH CONSTANT OPERAND?
        IF(NPREC(IPTR(2)).EQ.8.AND.ICAT(2).EQ.1) GO TO 17
        GO TO (20,46,70,190,150,330,350,370,390,410,430,450,490,530,550,
     -570,590,610,630,650,670,690,710,730,750,780,810,830,850,870,890,
     - 900),IDAT
C   UNARY FUNCTION WITH CONSTANT OPERAND SO DERIVATIVE ZERO
17      INODED=NEWNOD(1,0,0,INODED)
        INODE=INTGER(POPTOP(W(99)))
C   ASCEND A LEVEL IN THE RECURSION
        CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C   UNIT LENGTH OPERAND MODULE
19      INODED=NDIF(INODE,INODED,DIFVAR)
        CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C   ADDITION MODULE
20      IF(ICAT(1).GT.1) GO TO 26
        IF(ICAT(2).GT.1) GO TO 22
        INODED=NEWNOD(1,0,0,INODED)
        GO TO 45
22      IF(ICAT(2).GT.2) GO TO 44
        INODED=NEWNOD(1,0,1,INODED)
        GO TO 45
26      IF(ICAT(1).GT.2) GO TO 30
        IF(ICAT(2).GT.1) GO TO 28
        INODED=NEWNOD(1,0,1,INODED)
        GO TO 45
28      IF(ICAT(2).GT.2) GO TO 34
        INODED=NEWNOD(1,0,2,INODED)
        GO TO 45
30      IF(ICAT(2).GT.1) GO TO 32
        IND=1
        GO TO 36
32      IF(IEQOPD(IPTR,LNGTH).NE.0) GO TO 34
        INODED=NEWNOD(0,5,OPRS(3),INODED)
        INODED=NEWNOD(1,0,2,INODED)
        IND=1
        GO TO 36
34      IND=0
```

University of Cape Town

```
      INODED=NEWNOD(0,3,OPRS(1),INODED)
36    IPM=1
38    INODE=IPTR(IPM)
      DO 40 IK=1,2
      IDUM=NEWTOP(IPTR(IK),W(99))
40    IDUM=NEWTOP(LNGTH(IK),W(99))
      IDUM=NEWTOP(IPM,W(99))
      IDUM=NEWTOP(IND,W(99))
      DUMY=VISIT($10,-1)
      IND=INTGER(POPTOP(W(99)))
      IPM=INTGER(POPTOP(W(99)))
      DO 42 IK=2,1,-1
      LNGTH(IK)=INTGER(POPTOP(W(99)))
42    IPTR(IK)=INTGER(POPTOP(W(99)))
      IF(IND.EQ.1) GO TO 45
      IF(IPM.EQ.2) GO TO 45
44    IPM=2
      GO TO 38
45    INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C  SUBTRACTION MODULE
46    IF(ICAT(1).GT.1) GO TO 50
      IF(ICAT(2).GT.1) GO TO 48
47    INODED=NEWNOD(1,0,0,INODED)
      GO TO 66
48    INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      GO TO 64
50    IF(ICAT(1).GT.2) GO TO 54
      IF(ICAT(2).GT.1) GO TO 52
      INODED=NEWNOD(1,0,1,INODED)
      GO TO 66
52    IF(ICAT(2).GT.2) GO TO 56
      GO TO 47
54    IF(ICAT(2).GT.1) GO TO 55
      IND=1
      GO TO 58
55    IF(IEQOPD(IPTR,LNGTH).EQ.0) GO TO 47
56    IND=0
      INODED=NEWNOD(0,4,OPRS(2),INODED)
58    IPM=1
60    INODE=IPTR(IPM)
      DO 61 IK=1,2
      IDUM=NEWTOP(IPTR(IK),W(99))
61    IDUM=NEWTOP(LNGTH(IK),W(99))
      IDUM=NEWTOP(IPM,W(99))
      IDUM=NEWTOP(IND,W(99))
      DUMY=VISIT($10,-1)
      IND=INTGER(POPTOP(W(99)))
      IPM=INTGER(POPTOP(W(99)))
      DO 62 IK=2,1,-1
      LNGTH(IK)=INTGER(POPTOP(W(99)))
62    IPTR(IK)=INTGER(POPTOP(W(99)))
      IF(IND.EQ.1) GO TO 66
      IF(IPM.EQ.2) GO TO 66
64    IPM=2
```

University of Cape Town

```
      GO TO 60
66    INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-------------------------------------------------------------------------------
C  MULTIPLICATION MODULE
70    ISW=0
      IF(ICAT(1).GT.1) GO TO 76
      IF(ICAT(2).GT.1) GO TO 72
      INODED=NEWNOD(1,0,0,INODED)
      GO TO 140
72    IF(ICAT(2).GT.2) GO TO 130
      INODED=IOPCPY(IPTR(1),LNGTH(1),INODED)
      GO TO 140
76    IF(ICAT(1).GT.2) GO TO 82
      IF(ICAT(2).GT.1) GO TO 78
      INODED=IOPCPY(IPTR(2),LNGTH(2),INODED)
      GO TO 140
78    IF(ICAT(2).GT.2) GO TO 80
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      INODED=IOPCPY(IPTR(1),LNGTH(1),INODED)
      GO TO 140
80    ISW=1
      GO TO 87
82    IF(ICAT(2).GT.1) GO TO 84
      IND=1
      GO TO 89
84    IF(ICAT(2).GT.2) GO TO 86
      ISW=2
      GO TO 87
86    IF(IEQOPD(IPTR,LNGTH).NE.0) GO TO 87
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      IND=1
      GO TO 89
87    IND=0
      INODED=NEWNOD(0,3,OPRS(1),INODED)
89    IPM=1
      IF(ISW.EQ.1) GO TO 125
90    INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODE=IPTR(IPM)
      DO 110 IK=1,2
      IDUM=NEWTOP(IPTR(IK),W(99))
110   IDUM=NEWTOP(LNGTH(IK),W(99))
      IDUM=NEWTOP(IPM   ,W(99))
      IDUM=NEWTOP(IND,W(99))
      IDUM=NEWTOP(ISW,W(99))
      DUMY=VISIT($10,-1)
      ISW=INTGER(POPTOP(W(99)))
      IND=INTGER(POPTOP(W(99)))
      IPM  =INTGER(POPTOP(W(99)))
      DO 120 IK=2,1,-1
      LNGTH(IK)=INTGER(POPTOP(W(99)))
120   IPTR(IK)=INTGER(POPTOP(W(99)))
125   JPM=MOD(IPM,2)+1
      INODED=IOPCPY(IPTR(JPM),LNGTH(JPM),INODED)
      IF(IND.EQ.1) GO TO 140
```

University of Cape Town

```
      IF(IPM.EQ.2) GO TO 140
130   IPM=2
      IF(ISW.EQ.2) GO TO 125
      GO TO 90
140   INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C--------------------------------------------------------------------
C  EXPONENTIATION MODULE
150   ISW=0
      IF(ICAT(1).GT.1) GO TO 160
      IDAT=NDAT(IPTR(1))
      IF(IDAT.NE.0.AND.IDAT.NE.1) GO TO 152
151   INODED=NEWNOD(1,0,0,INODED)
      GO TO 188
152   IF(ICAT(2).EQ.1) GO TO 151
      IF(ICAT(2).GT.2) GO TO 183
      IF(NDAT(IPTR(1)).NE.1HE) GO TO 154
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      DO 153 IK=1,2
153   INODED=IOPCPY(IPTR(IK),LNGTH(IK),INODED)
      GO TO 188
154   ISW=2
      GO TO 183
160   IF(ICAT(1).GT.2) GO TO 170
      IF(ICAT(2).GT.1) GO TO 166
      IDAT=NDAT(IPTR(2))
      IF(IDAT.EQ.0) GO TO 151
      IF(IDAT.NE.1) GO TO 162
      INODED=NEWNOD(1,0,1,INODED)
      GO TO 188
162   ISW=1
      IND=1
      GO TO 174
166   IF(ICAT(2).GT.2) GO TO 168
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      DO 167 IK=1,2
167   INODED=IOPCPY(IPTR(IK),LNGTH(IK),INODED)
      INODED=NEWNOD(0,3,OPRS(1),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,8,OPRS(6),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=IOPCPY(IPTR(1),LNGTH(1),INODED)
      GO TO 188
168   ISW=1
      GO TO 173
170   IF(ICAT(2).GT.1) GO TO 172
      IND=1
      GO TO 174
172   IF(ICAT(2).GT.2) GO TO 173
      ISW=2
173   IND=0
      INODED=NEWNOD(0,3,OPRS(1),INODED)
174   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=IOPCPY(IPTR(2),LNGTH(2),INODED)
      IF(ISW.EQ.1) GO TO 175
      INODED=NEWNOD(0,5,OPRS(3),INODED)
```

University of Cape Town

```
175    INODED=NEWNOD(0,7,OPRS(5),INODED)
       INODED=IOPCPY(IPTR(1),LNGTH(1),INODED)
       IF(NCODE(IPTR(2)).NE.1) GO TO 177
       INODED=NEWNOD(1,0,NDAT(IPTR(2))-1,INODED)
       GO TO 179
177    IF(NCODE(IPTR(2)).NE.2) GO TO 178
       INODED=NEWNOD(2,0,REALNO(NDAT(IPTR(2)))-1.0,INODED)
       GO TO 179
178    INODED=NEWNOD(0,4,OPRS(2),INODED)
       INODED=IOPCPY(IPTR(2),LNGTH(2),INODED)
       INODED=NEWNOD(1,0,1,INODED)
179    IF(ISW.EQ.1) GO TO 182
       INODE=IPTR(1)
       DO 180 IK=1,2
       IDUM=NEWTOP(IPTR(IK),W(99))
180    IDUM=NEWTOP(LNGTH(IK),W(99))
       IDUM=NEWTOP(ISW,W(99))
       IDUM=NEWTOP(IND,W(99))
       DUMY=VISIT($10,-1)
       IND=INTGER(POPTOP(W(99)))
       ISW=INTGER(POPTOP(W(99)))
       DO 181 IK=2,1,-1
       LNGTH(IK)=INTGER(POPTOP(W(99)))
181    IPTR(IK)=INTGER(POPTOP(W(99)))
182    IF(IND.EQ.1) GO TO 188
183    INODED=NEWNOD(0,5,OPRS(3),INODED)
       INODED=NEWNOD(0,7,OPRS(5),INODED)
       DO 184 IK=1,2
184    INODED=IOPCPY(IPTR(IK),LNGTH(IK),INODED)
       IF(NDAT(IPTR(1)).EQ.1HE) GO TO 186
       IF(ISW.EQ.2) GO TO 185
       INODED=NEWNOD(0,5,OPRS(3),INODED)
185    INODED=NEWNOD(0,8,OPRS(6),INODED)
       INODED=NEWNOD(3,0,1H#,INODED)
       INODED=IOPCPY(IPTR(1),LNGTH(1),INODED)
       IF(ISW.EQ.2) GO TO 188
186    INODE=IPTR(2)
       DUMY=VISIT($10,-1)
188    INODE=INTGER(POPTOP(W(99)))
       CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C  DIVISION MODULE
190    ISW=0
       INODED=NEWNOD(0,6,OPRS(4),INODED)
       IF(ICAT(1).GT.1) GO TO 196
       IDAT=NDAT(IPTR(1))
       IF(IDAT.NE.0) GO TO 192
191    INODED=NEWNOD(1,0,0,INODED)
       IDUM=INTGER(DELETE(MADLFT(INODED)))
       GO TO 250
192    IF(ICAT(2).EQ.1) GO TO 191
       IF(ICAT(2).GT.2) GO TO 194
       INODED=NEWNOD(0,4,OPRS(2),INODED)
       INODED=NEWNOD(3,0,1H#,INODED)
       INODED=IOPCPY(IPTR(1),LNGTH(1),INODED)
       INODED=NEWNOD(0,7,OPRS(5),INODED)
       INODED=IOPCPY(IPTR(2),LNGTH(2),INODED)
```

University of Cape Town

```
      INODED=NEWNOD(1,0,2,INODED)
      GO TO 250
194   IND=2
      INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      GO TO 252
196   IF(ICAT(1).GT.2) GO TO 200
      IF(ICAT(2).GT.1) GO TO 198
      INODED=NEWNOD(1,0,1,INODED)
      INODED=IOPCPY(IPTR(2),LNGTH(2),INODED)
      GO TO 250
198   IF(ICAT(2).EQ.2) GO TO 191
      ISW=1
      GO TO 206
200   IF(ICAT(2).GT.1) GO TO 202
      IND=1
      GO TO 252
202   IF(ICAT(2).GT.2) GO TO 204
      ISW=2
      GO TO 206
204   IF(IEQOPD(IPTR,LNGTH).EQ.0) GO TO 191
206   INODED=NEWNOD(0,4,OPRS(2),INODED)
      IPM=1
      IF(ISW.EQ.1) GO TO 232
210   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODE=IPTR(IPM)
      DO 220 IK=1,2
      IDUM=NEWTOP(IPTR(IK),W(99))
220   IDUM=NEWTOP(LNGTH(IK),W(99))
      IDUM=NEWTOP(IPM,W(99))
      IDUM=NEWTOP(ISW,W(99))
      DUMY=VISIT($10,-1)
      ISW=INTGER(POPTOP(W(99)))
      IPM=INTGER(POPTOP(W(99)))
      DO 230 IK=2,1,-1
      LNGTH(IK)=INTGER(POPTOP(W(99)))
230   IPTR(IK)=INTGER(POPTOP(W(99)))
232   JPM=MOD(IPM,2)+1
      INODED=IOPCPY(IPTR(JPM),LNGTH(JPM),INODED)
      IF(IPM.EQ.2) GO TO 234
      IPM=2
      IF(ISW.EQ.2) GO TO 232
      GO TO 210
234   INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=IOPCPY(IPTR(2),LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
250   INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
252   INODE=IPTR(IND)
      DO 254 IK=1,2
      IDUM=NEWTOP(IPTR(IK),W(99))
254   IDUM=NEWTOP(LNGTH(IK),W(99))
      IDUM=NEWTOP(IND,W(99))
      DUMY=VISIT($10,-1)
      IND=INTGER(POPTOP(W(99)))
      DO 256 IK=2,1,-1
```

University of Cape Town

```
      LNGTH(IK)=INTGER(POPTOP(W(99)))
256   IPTR(IK)=INTGER(POPTOP(W(99)))
      IF(IND.EQ.1) GO TO 258
      INODED=IOPCPY(IPTR(1),LNGTH(1),INODED)
      GO TO 234
258   INODED=IOPCPY(IPTR(2),LNGTH(2),INODED)
      GO TO 250
C------------------------------------------------------------------------
C  NATURAL LOGARITHM MODULE
330   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,6,OPRS(4),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C------------------------------------------------------------------------
C  COMMON LOGARITHM MODULE
350   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,8,OPRS(7),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=NEWNOD(3,0,1HE,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,6,OPRS(4),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C------------------------------------------------------------------------
C  SINE MODULE
370   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,8,OPRS(9),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C------------------------------------------------------------------------
C  COSINE MODULE
390   INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,8,OPRS(8),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C------------------------------------------------------------------------
C  TANGENT MODULE
410   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
```

University of Cape Town

```
      INODED=NEWNOD(0,8,OPRS(12),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C  COTANGENT MODULE
430   INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=NEWNOD(0,8,OPRS(13),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C  SECANT MODULE
450   INODE=IPTR(2)
      LSEC=12
460   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,8,OPRS(LSEC),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      IF(LSEC.EQ.10) GO TO 480
      LSEC=10
      GO TO 460
480   DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C  COSECANT MODULE
490   INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      LCSC=13
500   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,8,OPRS(LCSC),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      IF(LCSC.EQ.11) GO TO 520
      LCSC=11
      GO TO 500
520   DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C  ARCSINE MODULE
530   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,6,OPRS(4),INODED)
      INODED=NEWNOD(1,0,1,INODED)
```

University of Cape Town

```
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      INODED=NEWNOD(2,0,0.5,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0.0,-1)
C-----------------------------------------------------------------------
C ARCCOSINE MODULE
550   INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,6,OPRS(4),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      INODED=NEWNOD(2,0,0.5,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0.0,-1)
C-----------------------------------------------------------------------
C  ARCTANGENT MODULE
570   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,6,OPRS(4),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,3,OPRS(1),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0.0,-1)
C-----------------------------------------------------------------------
C  ARCCOTANGENT MODULE
590   INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=NEWNOD(0,3,OPRS(1),INODED)
      INODED=NEWNOD(0,6,OPRS(4),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,3,OPRS(1),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      DUMY=VISIT($10,-1)
```

University of Cape Town

```
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0.0,-1)
C-----------------------------------------------------------------------
C  ARCSECANT MODULE
610   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,6,OPRS(4),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,8,OPRS(20),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(2,0,0.5,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0.0,-1)
C-----------------------------------------------------------------------
C  ARCCOSECANT MODULE
630   INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,6,OPRS(4),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,8,OPRS(20),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(2,0,0.5,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0.0,-1)
C-----------------------------------------------------------------------
C  ABSOLUTE VALUE MODULE
650   PRINT 660,INODE
660   FORMAT(' ATTEMPT TO DIFFERENTIATE DISCONTINUOUS FUNCTION, MOD, OCC
     -URRING IN NODE ',06,' OF LIST')
      CALL WLKBCK(3)
C-----------------------------------------------------------------------
C  HYPERBOLIC SINE MODULE
670   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,8,OPRS(22),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
```

University of Cape Town

```
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C     HYPERBOLIC COSINE MODULE
690   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,8,OPRS(21),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C     HYPERBOLIC TANGENT MODULE
710   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=NEWNOD(0,8,OPRS(25),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C     HYPERBOLIC COTANGENT MODULE
730   INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=NEWNOD(0,8,OPRS(26),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C     HYPERBOLIC SECANT MODULE
750   INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      LSECH=25
760   INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,8,OPRS(LSECH),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      IF(LSECH.EQ.23) GO TO 770
      LSECH=23
      GO TO 760
770   DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C     HYPERBOLIC COSECANT MODULE
```

University of Cape Town

```
780    INODED=NEWNOD(0,4,OPRS(2),INODED)
       INODED=NEWNOD(3,0,1H#,INODED)
       INODE=IPTR(2)
       LCSCH=26
790    INODED=NEWNOD(0,5,OPRS(3),INODED)
       INODED=NEWNOD(0,8,OPRS(LCSCH),INODED)
       INODED=NEWNOD(3,0,1H#,INODED)
       INODED=IOPCPY(INODE,LNGTH(2),INODED)
       IF(LCSCH.EQ.24) GO TO 800
       LCSCH=24
       GO TO 790
800    DUMY=VISIT($10,-1)
       INODE=INTGER(POPTOP(W(99)))
       CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C   INVERSE HYPERBOLIC SINE MODULE
810    INODED=NEWNOD(0,5,OPRS(3),INODED)
       INODED=NEWNOD(0,6,OPRS(4),INODED)
       INODED=NEWNOD(1,0,1,INODED)
       INODED=NEWNOD(0,7,OPRS(5),INODED)
       INODED=NEWNOD(0,3,OPRS(1),INODED)
       INODED=NEWNOD(0,7,OPRS(5),INODED)
       INODE=IPTR(2)
       INODED=IOPCPY(INODE,LNGTH(2),INODED)
       INODED=NEWNOD(1,0,2,INODED)
       INODED=NEWNOD(1,0,1,INODED)
       INODED=NEWNOD(2,0,0.5,INODED)
       DUMY=VISIT($10,-1)
       INODE=INTGER(POPTOP(W(99)))
       CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C   INVERSE HYPERBOLIC COSINE MODULE
830    INODED=NEWNOD(0,3,OPRS(1),INODED)
       INODED=NEWNOD(3,0,1H#,INODED)
       INODED=NEWNOD(0,5,OPRS(3),INODED)
       INODED=NEWNOD(0,6,OPRS(4),INODED)
       INODED=NEWNOD(1,0,1,INODED)
       INODED=NEWNOD(0,7,OPRS(5),INODED)
       INODED=NEWNOD(0,4,OPRS(2),INODED)
       INODED=NEWNOD(0,7,OPRS(5),INODED)
       INODE=IPTR(2)
       INODED=IOPCPY(INODE,LNGTH(2),INODED)
       INODED=NEWNOD(1,0,2,INODED)
       INODED=NEWNOD(1,0,1,INODED)
       INODED=NEWNOD(2,0,0.5,INODED)
       DUMY=VISIT($10,-1)
       INODE=INTGER(POPTOP(W(99)))
       CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C   INVERSE HYPERBOLIC TANGENT MODULE
850    INODED=NEWNOD(0,5,OPRS(3),INODED)
       INODED=NEWNOD(0,6,OPRS(4),INODED)
       INODED=NEWNOD(1,0,1,INODED)
       INODED=NEWNOD(0,4,OPRS(2),INODED)
       INODED=NEWNOD(1,0,1,INODED)
       INODED=NEWNOD(0,7,OPRS(5),INODED)
       INODE=IPTR(2)
```

University of Cape Town

```
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C     INVERSE HYPERBOLIC COTANGENT MODULE
870   INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,6,OPRS(4),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      INODED=NEWNOD(1,0,1,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C     INVERSE HYPERBOLIC SECANT MODULE
890   INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,6,OPRS(4),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      INODED=NEWNOD(2,0,0.5,INODED)
      DUMY=VISIT($10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C-----------------------------------------------------------------------
C     INVERSE HYPERBOLIC COSECANT MODULE
900   INODED=NEWNOD(0,4,OPRS(2),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,6,OPRS(4),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,5,OPRS(3),INODED)
      INODED=NEWNOD(0,8,OPRS(20),INODED)
      INODED=NEWNOD(3,0,1H#,INODED)
      INODE=IPTR(2)
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
      INODED=NEWNOD(0,3,OPRS(1),INODED)
      INODED=NEWNOD(1,0,1,INODED)
      INODED=NEWNOD(0,7,OPRS(5),INODED)
```

University of Cape Town

```
      INODED=IOPCPY(INODE,LNGTH(2),INODED)
      INODED=NEWNOD(1,0,2,INODED)
      INODED=NEWNOD(2,0,0.5,INODED)
      DUMY=VISIT(%10,-1)
      INODE=INTGER(POPTOP(W(99)))
      CALL TERM(0,0,-1)
C----------------------------------------------------------------------
      END
```

University of Cape Town

```
      FUNCTION FNS(DUMMY)
      INTEGER FNS
      RETURN
C-----------------------------------------------------------------------
      ENTRY IEQOPD(IOPNDS,LNGTHS)
C  TEST IF THE TWO OPERANDS WITH STARTING ADDRESSES IN IOPNDS
C  ARRAY & OF LENGTHS STORED IN LNGTHS ARRAY ARE IDENTICAL
      DIMENSION IOPNDS(2),LNGTHS(2)
C  ARE THE OPERANDS OF EQUAL LENGTH?
      IF(LNGTHS(1).NE.LNGTHS(2)) GO TO 3
      I=IOPNDS(1)
      J=IOPNDS(2)
      K=0
C  SCAN OPERANDS RIGHTWARDS COMPARING THEIR NODES FOR EQUALITY
 1    IF(INHALT(I+1).NE.INHALT(J+1)) GO TO 3
      IF(INHALT(I+2).NE.INHALT(J+2)) GO TO 3
      K=K+1
      IF(K.GE.LNGTHS(1)) GO TO 2
      I=MADRGT(I)
      J=MADRGT(J)
      GO TO 1
C  OPERANDS ARE EQUAL
2     FNS=0
      RETURN
C  OPERANDS NOT EQUAL
3     FNS=-1
      RETURN
C-----------------------------------------------------------------------
      ENTRY NCODE(INODE)
C  ACCESS CONTENTS OF CODE FIELD OF NODE POINTED TO BY INODE
      FNS=INTGER(GETMRK(INODE))
      RETURN
C-----------------------------------------------------------------------
      ENTRY NDAT(INODE)
C  ACCESS DATUM FIELD OF NODE POINTED TO BY INODE
      FNS=INHALT(INODE+2)
      RETURN
C-----------------------------------------------------------------------
      ENTRY NEWNOD(ICODE,IPREC,IDAT,JNODE)
C  INSERT ARGUMENTS IN CODE, PREC & DATUM FIELDS RESPECTIVELY
C  OF A NEW NODE INSERTED NEXT TO THAT POINTED TO BY JNODE
C  RETURN ADDRESS OF NEW NODE AS FUNCTION VALUE
      FNS= NXTRGT(IDAT,JNODE)
      IF(ICODE.NE.-1) DUMY=PUTMRK(ICODE,FNS)
      IF(IPREC.NE.-1) DUMY=PUTFLG(IPREC,FNS)
      RETURN
C-----------------------------------------------------------------------
      ENTRY NPREC(INODE)
C  ACCESS CONTENTS OF PREC FIELD OF NODE POINTED TO BY INODE
      FNS=INTGER(GETFLG(INODE))
      RETURN
      END
```

University of Cape Town

```
      FUNCTION FNT(DUMMY)
      INTEGER FNT
      RETURN
C-----------------------------------------------------------------------
      ENTRY HANDLE(NBEG,NEND,LENGTH)
C   THIS SUBROUTINE FLOWCHARTED IN APPENDIX C
C   NBEG IS STARTING ADDRESS OF POLISH PREFIX ORDERED OPERAND TO BE
C   DELINEATED.
C   UPON COMPLETION OF PROCESS, NEND HOLDS ADDRESS OF LAST NODE OF
C   OPERAND & LENGTH THE NUMBER OF NODES IN IT.
C   KOPTR IS COUNT OF NUMBER OF OPERATORS IN OPERAND AS IT IS SCANNED
C   & KOPND IS COUNT OF NUMBER OF NODES IN OPERAND THAT ARE NOT
C   OPERATORS AS IT IS SCANNED RIGHTWARDS
C   NOPND IS TOTAL NUMBER OF UNIT LENGTH OPERANDS REQUIRED IN PREFIX
C   OPERAND AS IT IS SCANNED. THIS INCREASES WITH EVERY OPERATOR IN
C   OPERAND EXPRESSION
      FNT=NEND
      NTEMP=NBEG
      KOPTR=0
      KOPND=0
      NOPND=1
1     IF(NCODE(NTEMP).EQ.0) GO TO 2
       KOPND=KOPND+1
      GO TO 3
2     KOPTR=KOPTR+1
      NOPND=NOPND+1
3     IF(KOPND.LT.NOPND) GO TO 4
      LENGTH=2*KOPTR+1
      NEND=NTEMP
      RETURN
4     NTEMP=MADRGT(NTEMP)
      GO TO 1
C-----------------------------------------------------------------------
      ENTRY IOPCPY(IUNDIF,LENGTH,IDIF)
C   COPY OPERAND OF LENGTH 'LENGTH' STARTING AT NODE POINTED TO BY
C   'IUNDIF' INTO NEW NODES INSERTED TO RIGHT OF THAT POINTED TO BY IDIF
      I=IUNDIF
      DO 5 J=1,LENGTH
      IDIF=NEWNOD(NCODE(I),NPREC(I),NDAT(I),IDIF)
      DUMY=PUTHAF(GETHAF(I),IDIF)
5     I=MADRGT(I)
C   VALUE IS THE LAST CELL OF THE COPIED OPERAND
      FNT   =IDIF
      RETURN
C-----------------------------------------------------------------------
      ENTRY ISYNTX(LST)
      I=LNKR(LOCT(LST))
      KOPTR=0
      KOPND=0
    6 IF(KOPTR+1.LT.KOPND) GO TO 9
      I=MADRGT(I)
      IF(ID(I).EQ.2) GO TO 8
      IF(NCODE(I).EQ.0) GO TO 7
      KOPND=KOPND+1
      GO TO 6
    7 KOPTR=KOPTR+1
      GO TO 6
```

University of Cape Town

```
8 IF(KOPTR+1.NE.KOPND) GO TO 9
  FNT=0
  RETURN
9 FNT=-1
  RETURN
  END
```

University of Cape Town

```
      FUNCTION FNU(DUMMY)
      INTEGER FNU
      COMMON /SLIP/ W(100)
      RETURN
C-----------------------------------------------------------------------
      ENTRY INFORD(LSTIN,LSTOUT)
C  CONVERSION OF POLISH PREFIX LIST LSTIN TO INFIX LIST TO BE  CREATED
C  WITH NAME LSTOUT
C  THIS ROUTINE ADAPTED FROM FLOWCHART IN FIG. 9 OF TEXT EXCEPT
C  THAT LISTS RATHER THAN ARRAYS ARE USED.
C
C  IIN IS POINTER TRAVERSING INPUT LIST,IOUT IS POINTER IN OUTPUT
C  LIST. W(99) IS USED AS PUSH DOWN STACK. THE POINTERS TO THE
C  OPERANDS AT ANY STAGE ARE DETERMINED BY THE HANDLE ROUTINE & ARE
C  STORED IN THE IPTR ARRAY.
      DIMENSION IPTR(2)
      IF(ISYNTX(LOCT(LSTIN)).NE.0) GO TO 135
      IIN=MADRGT(LNKR(LSTIN))
C  CREATE LIST FOR OUTPUT INFIX ORDERED EXPRESSION
      IOUT=LNKR(LIST(LSTOUT))
      FNU=LSTOUT
      IF(NCODE(IIN).EQ.0) GO TO 10
      IOUT=IOPCPY(IIN,1,IOUT)
      RETURN
10    DUMY=VISIT($20,-1)
      RETURN
20    ITEMP=IIN
      DO 30 I=1,2
      IPTR(I)=MADRGT(ITEMP)
30    CALL HANDLE(IPTR(I),ITEMP,IDUM)
      IND=1
50    IPAR=0
      NODE=IPTR(IND)
      IF(NCODE(NODE)),70
      IF(NDAT(NODE).NE.1H#) IOUT=IOPCPY(NODE,1,IOUT)
60    IF(IND.EQ.2) CALL TERM(0,0,-1)
      IND=2
      IIN=MADLFT(IPTR(1))
      IOUT=IOPCPY(IIN,1,IOUT)
      GO TO 50
70    IF(NPREC(NODE).GT.NPREC(IIN)) GO TO 110
      IDUM=NDAT(IIN)
      IF(NPREC(NODE).LT.NPREC(IIN)) GO TO 80
      IF(IDUM.EQ.1H+.OR.IDUM.EQ.1H*) GO TO 110
      IF(IND.EQ.1) GO TO 110
80    IPAR=1
      IOUT=NEWNOD(0,2,1H(,IOUT)
110   DO 120 I=1,2
120   IDUM=NEWTOP(IPTR(I),W(99))
      IDUM=NEWTOP(IND,W(99))
      IDUM=NEWTOP(IPAR,W(99))
      IIN=NODE
      DUMY=VISIT($20,-1)
      IPAR=INTGER(POPTOP(W(99)))
      IF(IPAR.EQ.1) IOUT=NEWNOD(0,1,1H),IOUT)
      IND=INTGER(POPTOP(W(99)))
      DO 130 I=2,1,-1
```

University of Cape Town

```
130    IPTR(I)=INTGER(POPTOP(W(99)))
       GO TO 60
135    PRINT 901,LSTIN
901    FORMAT(//' POLISH PREFIX-ORDERED LIST ',O12,' SYNTACTICALLY INCORR
      -ECT')
       CALL WLKBCK(2)
C-----------------------------------------------------------------------
       ENTRY IRALEX(NAME)
C   DELETE EXPRESSION WITH LIST NAME 'NAME'
C   THIS IS AS IRALST OF SLIP BUT INFORMATION ON COLLATOR LIST
C   RELATING TO THE EXPRESSION IS ALSO REMOVED
       NHDR=LNKR(NAME)
       I=LNKR(W(100))
 140   I=MADRGT(I)
       IF(ID(I).EQ.2) GO TO 150
       IF(INTGER(GETHAF(I)).NE.NHDR) GO TO 140
       DUMY=DELETE(I)
 150   FNU   =IRALST(NAME)
       RETURN
C-----------------------------------------------------------------------
       ENTRY NDIF(INODE,INODED,DIFVAR)
C   THIS IS THE DERIVATIVE FUNCTION FOR AN OPERAND OF SINGLE LENGTH
C   SEE FLOWCHART IN APPENDIX C.
C   INODE IS POINTER TO OPERAND TO BE DIFFERENTIATED
C   INODED IS POINTER TO BOTTOM OF DERIVATIVE LIST
C   DIFVAR IS HOLLERITH NAME OF VARIABLE OF DIFFERENTIATION
C   IS DATUM A NULL OPERAND?
       IF(NDAT(INODE).EQ.'#') GO TO 170
C   IS OPERAND THE VARIABLE OF DIFFERENTIATION?
       NC=NCODE(INODE)
       IF(NC.EQ.4.AND.NDAT(INODE).EQ.INTGER(DIFVAR)) GO TO 160
C   IS THIS A SUBEXPRESSION NODE?
       IF(INTGER(GETHAF(INODE)).NE.0) GO TO 180
C   NODE IS NONE OF ABOVE SO A CONSTANT - DERIVATIVE IS ZERO
155    FNU =NEWNOD(1, 0,0,INODED)
       RETURN
C   NODE IS VARIABLE SO DERIVATIVE IS 1
160    FNU =NEWNOD(1, 0,1,INODED)
       RETURN
C   NODE IS NULL OPERAND SO DERIVATIVE IS AS WELL
170    FNU =NEWNOD(3, 0,NDAT(INODE),INODED)
       RETURN
C   NODE IS SUBEXPRESSION REFERENCE: OBTAIN ADDRESS OF SUBEXPR. HEADER
180    LSTDIF=INTGER(GETHAF(INODE))
C   CONVERT TO NAME-FORMAT
       CALL SETDIR(-1,LSTDIF,LSTDIF,LSTDIF)
C   SEARCH DL OF SUBLIST FOR ATTRIBUTE DIFVAR & RETURN CORRESPONDING
C   VALUE WHICH IS DERIVATIVE OF LSTDIF IF IT EXISTS, 0 OTHERWISE
       LSTDIF=ITSVAL(DIFVAR,LSTDIF)
C   HAS SUBEXPRESSION YET BEEN DIFFERENTIATED?
       IF(LSTDIF.EQ.0) GO TO 200
C   SEARCH COLLATOR LIST FOR TAG CORRESPONDING TO POINTER TO DERIVATIVE
C   OF SUBEXPRESSION
       I=LNKR(W(100))
190    I=MADRGT(I)
       IF(ID(I).EQ.2) GO TO 200
       IF(INTGER(GETHAF(I)).NE.LSTDIF) GO TO 190
```

University of Cape Town

```
C   NEW NODE ON DERIVATIVE LIST HAS TAG & POINTER OF DERIVATIVE OF
C   SUBEXPRESSION
        FNU =NEWNOD(4,0,NDAT(I),INODED)
        DUMY=PUTHAF(LSTDIF,FNU )
        RETURN
C   SUBEXPRESSION REFERENCED IN INODE NOT YET DIFFERENTIATED: ERROR
200     NNME=NDAT(INODE)
        PRINT 902,NNME,DIFVAR
902     FORMAT(' THE REFERENCED SUBLIST ',A6,' HAS NOT BEEN DIFFERENTIATED
      - WITH RESPECT TO VARIABLE ',A6)
        GO TO 155
C-----------------------------------------------------------------------
        ENTRY PREORD(LSTIN,LSTOUT)
C   THIS FUNCTION ADAPTED FROM FLOWCHART IN FIG 8 OF TEXT EXCEPT THAT
C   LISTS RATHER THAN ARRAYS ARE USED
C
C   IIN IS POINTER TRAVERSING INPUT LIST
C   IOUT IS POINTER IN OUTPUT LIST
C   W(99) IS USED AS THE PUSHDOWN STACK
C   LSTIN IS INPUT INFIX ORDERED LIST, LSTOUT IS NAME OF LIST TO
C   BE CREATED FOR OUTPUT POLISH PREFIX EXPRESSION
C   LSTIN IS SCANNED LEFTWARDS & LSTOUT BUILT UP IN SAME DIRECTION
        IIN=LNKR(LSTIN)
        IOUT=LNKR(LIST(LSTOUT))
        FNU=LSTOUT
C   INSERT PARENTHESES AROUND INPUT EXPRESSION
        IDUM=NEWNOD(0,2,1H(,IIN)
        IDUM=MADLFT(IIN)
        IDUM=NEWNOD(0,1,1H),IDUM)
220     IIN=MADLFT(IIN)
        IF(NCODE(IIN).EQ.0) GO TO 230
        IOUT=NXTLFT(NDAT(IIN),IOUT)
        DUMY=PUTFLG(NPREC(IIN),IOUT)
        DUMY=PUTHAF(GETHAF(IIN),IOUT)
        DUMY=PUTMRK(NCODE(IIN),IOUT)
        GO TO 220
230     IF(NDAT(IIN).NE.')') GO TO 250
240     IDUM=NEWTOP(IIN,W(99))
        GO TO 220
250     IF(NPREC(IIN).GE.NPREC(TOP(W(99)))) GO TO 260
        IDUM=INTGER(POPTOP(W(99)))
        IOUT=NXTLFT(NDAT(IDUM),IOUT)
        DUMY=PUTFLG(NPREC(IDUM),IOUT)
        DUMY=PUTHAF(GETHAF(IDUM),IOUT)
        DUMY=PUTMRK(NCODE(IDUM),IOUT)
        GO TO 250
260     IF(NDAT(IIN).NE.'(') GO TO 240
        IF(  NDAT(TOP(W(99))).NE.')') GO TO 240
        DUMY=POPTOP(W(99))
        IF(LISTMT(W(99)).NE.0) GO TO 220
        IDUM=LNKR(LSTIN)
C   DELETE PARENTHESES INSERTED AROUND INPUT EXPRESSION BEFORE ALGORITHM
C   WAS STARTED.
        DUMY=DELETE(LNKR(CONT(IDUM)))
        DUMY=DELETE(LNKL(CONT(IDUM)))
        RETURN
        END
```

University of Cape Town

```
      SUBROUTINE PRNTEX(LSTIN,IDEL)
C   A FLOWCHART OF THE PRINT ROUTINE APPEARS IN APPENDIX C.
      COMMON /SLIP/W(100)
      COMMON /IODIFN/CP,LP,RP,BLANK
      INTEGER LP(3),RP(3),BLANK,WORD,BUFFER(3),IBUF(3)
C   PRTBUF IS THE OUTPUT BUFFER OF 126 CHARACTERS
C   CP IS ARRAY OF MASKS TO ACCESS THE SIXTHS OF A WORD
C   BUFFER IS THE TEMPORARY VARIABLE BUFFER
C   IBUF IS USED IN CONVERSION OF NUMBERS TO HOLLERITH FORMAT
C   LP & RP ARE THE TABLES OF ALTERNATIVE TYPES OF PARENTHESES
C   ALLOWED
      DIMENSION PRTBUF(21),CP(6)
900   FORMAT(1H0,21A6)
901   FORMAT( )
902   FORMAT(////)
903   FORMAT(' INTERNAL NAME OF EXPRESSION UNKNOWN')
      NAME=LNKR(LSTIN)
      CALL TRCOFF
      CALL INFORD(LSTIN,LSTOUT)
      NEXT=LNKR(LSTOUT)
      I=LNKR(W(100))
15    I=MADRGT(I)
      IF(ID(I).EQ.2) GO TO 55
      IF(INTGER(GETHAF(I)).NE.NAME) GO TO 15
      NAME=NDAT(I)
      I=NEWNOD(4,0,NAME,NEXT)
      I=NEWNOD(4,0,1H=,I)
      IF(IDEL.EQ.9) GO TO 16
      IDUM=IRALEX(LSTIN)
C   INITIALISATION
16    IW=1
      IC=1
      LEFT=126
      DO 20 I=1,21
20    PRTBUF(I)=REALNO(BLANK)
      IDELIM=0
      ISUBL=0
      KPAR=0
      DO 30 I=1,3
      IBUF(I)=BLANK
30    BUFFER(I)=BLANK
C-----------------------------------------------------------------------
C   RIGHTWARD SCAN OF LIST
40    NEXT=MADRGT(NEXT)
      IF(ID(NEXT).NE.2) GO TO 60
      IRET=1
      GO TO 70
C   TERMINATION
50    PRINT 900,PRTBUF
      PRINT 902
      IDUM=IRALST(LSTOUT)
      RETURN
55    PRINT 903
      CALL WLKBCK(2)
C-----------------------------------------------------------------------
C   CHECK WHICH MODULE TO PROCEED TO
60    ICODE=NCODE(NEXT)
```

University of Cape Town

```
C   IS NODE A BASIC OPERATOR NODE?
      IF(ICODE.EQ.0.AND.(NPREC(NEXT).GE.3.AND.NPREC(NEXT).LE.7))GOTO 110
C   IS NODE A LEFT PARENTHESIS NODE?
      IF(ICODE.EQ.0.AND.NPREC(NEXT).EQ.2) GO TO 130
C   IS NODE A RIGHT PARENTHESIS NODE?
      IF(ICODE.EQ.0.AND.NPREC(NEXT).EQ.1) GO TO 150
C   IS NODE A SUBLIST REFERENCE?
      IF(INTGER(GETHAF(NEXT)).NE.0) ISUBL=1
C   NODE MUST HAVE ORDINARY ALPHABETIC OR NUMERIC ITEM
      GO TO 170
C------------------------------------------------------------------------
C   INSERTION INTO PRINT BUFFER
70    IF(NUMBER.GT.LEFT) GO TO 200
80    NUMBER=NUMBER-1
      DO 90 I=0,NUMBER
      IWS=I/6+1
      IS=MOD(I,6)+1
      ISYM=INTGER(SQOUT(CP(IS),BUFFER(IWS)))
      PRTBUF(IW)=SQIN(CP(IC),ISYM,PRTBUF(IW))
      IC=IC+1
      IF(IC.LE.6) GO TO 90
      IW=IW+1
      IC=1
90    CONTINUE
      LEFT=LEFT-NUMBER-1
      DO 100 I=1,3
100   BUFFER(I)=BLANK
      GO TO (50,120,140,160,180,181,190),IRET
C------------------------------------------------------------------------
C   NODE HAD ONE OF THE BINARY OPERATORS
110   IF(BUFFER(1).EQ.BLANK) GO TO 120
C   EMPTY VARIABLE BUFFER INTO PRINT BUFFER
      IRET=2
      GO TO 70
120   BUFFER(1)=NDAT(NEXT)
      NUMBER=1
      IDELIM=1
      GO TO 40
C------------------------------------------------------------------------
C   NODE HAD A LEFT PARENTHESIS
130   IF(BUFFER(1).EQ.BLANK) GO TO 140
C   EMPTY VARIABLE BUFFER INTO PRINT BUFFER
      IRET=3
      GO TO 70
140   KPAR=KPAR+1
      I=MOD(KPAR,2)+1
      BUFFER(1)=LP(I)
      NUMBER=1
      IDELIM=1
      GO TO 40
C------------------------------------------------------------------------
C   NODE HAD A  RIGHT PARENTHESIS
150   IF(BUFFER(1).EQ.BLANK) GO TO 160
C   EMPTY VARIABLE BUFFER INTO PRINT BUFFER
      IRET=4
      GO TO 70
160   I=MOD(KPAR,2)+1
```

University of Cape Town

```
      BUFFER(1)=RP(I)
      KPAR=KPAR-1
      NUMBER=1
      IDELIM=1
      GO TO 40
C-----------------------------------------------------------------------
C  NODE HAD AN ALPHANUMERIC DATUM
170   IF(BUFFER(1).EQ.BLANK) GO TO 180
C  EMPTY VARIABLE BUFFER INTO PRINT BUFFER
      IRET=5
      GO TO 70
180   IF(IDELIM.EQ.1) GO TO 181
C  INSERTION OF NECESSARY BLANK INTO PRINT BUFFER
      NUMBER=1
      IDELIM=1
      IRET=6
      GO TO 70
181   IF(ISUBL.EQ.0) GO TO 190
C  IF NODE A SUBLIST NODE INSERT $ INTO PRINT BUFFER
      NUMBER=1
      BUFFER(1)=1H$
      IRET=7
      GO TO 70
190   WORD=NDAT(NEXT)
      IF(NCODE(NEXT).EQ.1) GO TO 193
      IF(NCODE(NEXT).EQ.2) GO TO 194
      IST=1
      NUMBER=0
C  INSERTION OF CHARACTERS OF NODE DATUM INTO VARIABLE BUFFER
C  COUNT OF THEIR NUMBER MAINTAINED
191   ISYM=INTGER(SQOUT(CP(IST),WORD))
      IF(ISYM.EQ.5) GO TO 192
      NUMBER=NUMBER+1
      IST=IST+1
      IF(IST.LE.6) GO TO 191
192   BUFFER(1)=WORD
      GO TO 199
C  DATUM IS INTEGER SO ENCODE INTO HOLLERITH CHARACTERS
193   ENCODE(18,901,IBUF,NUMBER) WORD
      GO TO 195
194   RWORD=REALNO(WORD)
C  DATUM IS A REAL NUMBER SO ENCODE INTO HOLLERITH CHARACTERS
      ENCODE(18,901,IBUF,NUMBER) RWORD
C  RIGHT JUSTIFY CHARACTERS OF NUMBER & INSERT IN VARIABLE BUFFER
195   NW=1
      NC=1
      LESS=0
      NUMBER=NUMBER-1
      DO 197 I=0,NUMBER
      NWS=I/6+1
      NS=MOD(I,6)+1
      ISYM=INTGER(SQOUT(CP(NS),IBUF(NWS)))
      IF(ISYM.NE.5) GO TO 196
      LESS=LESS+1
      GO TO 197
196   BUFFER(NW)=INTGER(SQIN(CP(NC),ISYM,BUFFER(NW)))
      NC=NC+1
```

University of Cape Town

```
      IF(NC.LE.6) GO TO 197
      NW=NW+1
      NC=1
197   CONTINUE
      NUMBER=NUMBER+1-LESS
      DO 198 I=1,3
198   IBUF(I)=BLANK
199   IDELIM=0
      ISUBL=0
      GO TO 40
C-----------------------------------------------------------------------
C  EMPTY THE FULL PRINT BUFFER & REINITIALISE
200   PRINT 900,PRTBUF
      IW=1
      IC=1
      LEFT=126
      DO 210 I=1,21
210   PRTBUF(I)=REALNO(BLANK)
      GO TO 80
      END
```

University of Cape Town

```
      COMPILER (DATA=SHORT)
      FUNCTION READEX(IPRT)
C   THE DERIV INPUT ROUTINE
C
C   CP IS THE ARRAY OF MASKS USED TO ACCESS ALL THE SIXTHS OF A WORD
C   LP & RP ARE THE LEFT & RIGHT PARENTHESIS TABLES, ALLOWING FOR VARIOUS
C   TYPES. DELIM IS THE TABLE OF ALLOWABLE DELIMITER SYMBOLS.
C   OPRS IS THE OPERATOR TABLE. RES IS THE TABLE OF RESERVED WORDS
C   (SUCH AS 'PI'). NALL IS THE TABLE OF SYMBOLS WHICH ARE NOT ALLOWED
C   ON A CARD, HERE REPRESENTED IN OCTAL CODE.
C   CRDBUF IS THE CARD BUFFER OF 78 'COLUMNS'.
C
      COMMON /SLIP/W(100)
      COMMON /IODIFN/CP,LP,RP,BLANK,DELIM,OPRS,RES
      DIMENSION CP(6)/077000000000,07700000000,077000000,0770000,07700,
     -077/,NALL(14)/003,004,044,046,052,053,054,056,057,072,073,076,077/
      DIMENSION CRDBUF(13)
      INTEGER OPRS(40)/1H+,1H-,1H*,1H/,1H! ,2HLN,3HLOG,3HSIN,3HCOS,3HTAN
     -,3HCOT,3HSEC,3HCSC,4HASIN,4HACOS,4HATAN,4HACOT,4HASEC,4HACSC,3HMOD
     -,4HSINH,4HCOSH,4HTANH,4HCOTH,4HSECH,4HCSCH,5HASINH,5HACOSH,5HATANH
     -,5HACOTH,5HASECH,5HACSCH/,RES(10)/1H#,1HE,2HPI,1HI/
      INTEGER LP(3)/1H(,1H<,1H[/,RP(3)/1H),1H>,1H]/,BLANK/1H /,IEND/1R@/
     -,SYMBOL,WORD,PLACE,DECPTQ,BUFFER(3)
      INTEGER DELIM(15)/1R ,1R+,1R-,1R*,1R/,1R!,1R(,1R<,1R[,1R),1R>,1R]/
      DATA IOPRS,IRES,INALL,IDELIM/32,4,13,14/
099   FORMAT( )
900   FORMAT(13A6,1X,A1)
901   FORMAT(1H ,13A6)
C   CREATE TEMPORARY LIST FOR INPUT INFIX EXPRESSION
      IHDR=LNKR(LIST(STACK))
      INODE=IHDR
C   INITIALISATION
      IS=1
      IWS=1
      IST=0
      PLACE=BLANK
      WORD=BLANK
      IGNORE=0
      ISUBL=0
      LNAME=1
      KLP=0
      KRP=0
      KOPND=-1
      KOPTR=0
      NUMBQ=0
      DECPTQ=0
      DO 10 I=1,3
10    BUFFER(I)=BLANK
C-----------------------------------------------------------------------
C   FILL CARD BUFFER FROM A CARD
20    READ 900,CRDBUF,ICONT
      IF(IPRT.NE.9) GO TO 30
      PRINT 901,CRDBUF
C   IW COUNTS WORDS OF CARD BUFFER THAT CURRENTLY CONSIDERING
C   IC COUNTS SYMBOLS OF EACH WORD OF THE CARD BUFFER
30    IW=1
40    IC=1
```

University of Cape Town

```
C-----------------------------------------------------------------------
C  SCANNING & RECOGNITION MODULE
50     ISYM=INTGER(SQOUT(CP(IC),CRDBUF(IW)))
       IF(ISYM.EQ.IEND) GO TO 380
       IF(LNAME.EQ.0) GO TO 55
       IF(ISYM.NE.INTGER(1R=)) GO TO 55
       ISYM=DELIM(1)
       GO TO 120
55     DO 60 I=1,INALL
       IF(ISYM.EQ.NALL(I)) GO TO 400
60     CONTINUE
       DO 65 I=1,IDELIM
       IF(ISYM.EQ.DELIM(I)) GO TO 120
65     CONTINUE
C  RECOGNITION STAGE FOR ALPHAMERIC VARIABLES, REALS OR INTEGERS.
C  SEE FIGURE 16 IN TEXT FOR FLOWCHART
       IF(IS.NE.1) GO TO 90
       IF(ISYM.EQ.INTGER(1R$)) GO TO 70
       IF(ISYM.GE.1R0.AND.ISYM.LE.1R9) NUMBQ=1
       IF(ISYM.EQ.INTGER(1R.)) DECPTQ=DECPTQ+1
       GO TO 120
70     ISUBL=1
       GO TO 150
90     IF(NUMBQ.NE.1) GO TO 120
       IF(ISYM.GE.1R0.AND.ISYM.LE.1R9) GO TO 120
       IF(ISYM.EQ.INTGER(1R.)) GO TO 110
       NUMBQ=0
       GO TO 120
110    DECPTQ=DECPTQ+1
C-----------------------------------------------------------------------
C  RECOGNITION OF DELIMITERS
120    SYMBOL=INTGER(SQIN(CP(6),ISYM,PLACE))
       ISYMBL=LANORM(SYMBOL)
       IF(ISYMBL.EQ.BLANK) GO TO 180
       DO 130 I=1,3
       IF(ISYMBL.EQ.LP(I)) GO TO 190
       IF(ISYMBL.EQ.RP(I)) GO TO 210
130    CONTINUE
       DO 140 IOPTR=1,5
       IF(ISYMBL.EQ.OPRS(IOPTR)) GO TO 230
140    CONTINUE
C  NON-DELIMITER ENCOUNTERED
       GO TO 260
C-----------------------------------------------------------------------
C  CHECK FOR END OF WORD & END OF BUFFER BEFORE CONTINUING SCAN
150    IF(IC.EQ.6) GO TO 160
       IC=IC+1
       GO TO 50
160    IF(IW.LT.13) GO TO 170
       IF(ICONT.EQ.BLANK) GO TO 380
       GO TO 20
170    IW=IW+1
       GO TO 40
C-----------------------------------------------------------------------
C  BLANK DELIMITER MODULE
180    IF(WORD.EQ.BLANK) GO TO 150
       IRET=1
```

University of Cape Town

```
      GO TO 300
C-----------------------------------------------------------------------
C  LEFT PARENTHESIS MODULE
190   IF(WORD.EQ.BLANK) GO TO 200
      IRET=2
      GO TO 300
200   INODE=NEWNOD(0,2,1H(,INODE)
      KLP=KLP+1
      GO TO 150
C-----------------------------------------------------------------------
C  RIGHT PARENTHESIS MODULE
210   IF(WORD.EQ.BLANK) GO TO 220
      IRET=3
      GO TO 300
220   INODE=NEWNOD(0,1,1H),INODE)
      KRP=KRP+1
      GO TO 150
C-----------------------------------------------------------------------
C  'BINARY' OPERATOR DELIMITER
230   IF(WORD.EQ.BLANK) GO TO 240
      IRET=4
      GO TO 300
240   IF(ISYMBL.NE.OPRS(1).AND.ISYMBL.NE.OPRS(2)) GO TO 250
      IF(LISTMT(STACK).EQ.0) GO TO 245
      IF(NPREC(MADLFT(IHDR)).LT.2) GO TO 250
C  + OR - WAS UNARY: INSERT NULL OPERAND
245   INODE=NEWNOD(3,0,1H#,INODE)
      KOPND=KOPND+1
C  CREATE OPERATOR NODE IN LIST
250   INODE=NEWNOD(0,IOPTR+2,OPRS(IOPTR),INODE)
      KOPTR=KOPTR+1
      GO TO 150
C-----------------------------------------------------------------------
C  SYMBOL ENCOUNTERED. ADD TO TEMPORARY BUFFER 'WORD'.
C  SYNTACTICAL CHECKS
260   IF(IS.GT.6) GO TO 280
270   WORD=INTGER(SHIN(6,SYMBOL,WORD))
271   IS=IS+1
      IF(NUMBQ.EQ.1) IST=IST+1
      IF(DECPTQ.GT.1) GO TO 410
      GO TO 150
280   IF(NUMBQ.EQ.1) GO TO 290
      IF(IGNORE.EQ.1) GO TO 150
      IGNORE=1
      GO TO 430
290   IF(IWS.EQ.3) GO TO 420
      BUFFER(IWS)=WORD
      WORD=BLANK
      IS=0
      IWS=IWS+1
      GO TO 270
C-----------------------------------------------------------------------
C  INSERTION OF TEMPORARY BUFFER INTO LIST
300   IF(LNAME.EQ.0) GO TO 304
C  SAVE EXPRESSION TAG
      INTNAM=LANORM(WORD)
      LNAME=0
```

University of Cape Town

```
      GO TO 370
304   IF(ISUBL.EQ.0) GO TO 309
C  SUBLIST REFERENCE INPUT: CHECK THAT SUBLIST EXISTS
      I=LNKR(W(100))
      WORD=LANORM(WORD)
305   I=MADRGT(I)
      IF(ID(I).EQ.2) GO TO 460
      IF(WORD.NE.NDAT(I)) GO TO 305
C  CREATE SUBLIST NODE
      INODE=NEWNOD(4,0,WORD,INODE)
      DUMY=PUTHAF(GETHAF(I),INODE)
      ISUBL=0
      GO TO 370
309   IF(NUMBQ.EQ.1) GO TO 350
      DO 310 IOPRR=6,IOPRS
      IF(LANORM(WORD).EQ.OPRS(IOPRR)) GO TO 330
310   CONTINUE
      DO 320 I=2,IRES
      IF(WORD.EQ.RES(I)) GO TO 340
320   CONTINUE
C  CREATE VARIABLE NODE
      INODE=NEWNOD(4,0,LANORM(WORD),INODE)
      GO TO 370
C  CREATE NULL OPERAND & OPERATOR NODES (ONE OF 'UNARY' FUNCTIONS
C  ENCOUNTERED)
330   INODE=NEWNOD(3,0,1H#,INODE)
      INODE=NEWNOD(0,8,OPRS(IOPRR),INODE)
      KOPTR=KOPTR+1
      GO TO 370
C  CREATE RESERVED WORD NODE
340   INODE=NEWNOD(3,0,LANORM(WORD),INODE)
      GO TO 370
C  NUMBER ENCOUNTERED
350   BUFFER(IWS)=LANORM(WORD)
      IF(DECPTQ.EQ.1) GO TO 360
C  DECODE INTEGER ENCOUNTERED & CREATE INTEGER NODE
      DECODE(IST,899,BUFFER)IDATUM
      INODE=NEWNOD(1,0,IDATUM,INODE)
      GO TO 370
C  DECODE REAL NUMBER ENCOUNTERED & CREATE REAL NODE
360   DECODE(IST,899,BUFFER) DATUM
      INODE=NEWNOD(2,0,DATUM,INODE)
C  REINITIALISE
370   IS=1
      IWS=1
      IST=0
      WORD=BLANK
      IGNORE=0
      NUMBQ=0
      DECPTQ=0
      DO 375 I=1,3
375   BUFFER(I)=BLANK
      KOPND=KOPND+1
C  RETURN STATEMENTS
      GO TO (150,200,220,240,390),IRET
C------------------------------------------------------------------------
C  SYNTACTICAL CHECKS & FINALISATION MODULE
```

University of Cape Town

```
C   IS TEMPORARY BUFFER EMPTY?
380    IF(WORD.EQ.BLANK) GO TO 390
C   TEMPORARY BUFFER NOT EMPTY. SPECIFY RETURN & GO TO INSERTION MODULE
       IRET=5
       GO TO 300
390    IF(KLP.NE.KRP) GO TO 440
       IF(KOPTR+1.NE.KOPND) GO TO 450
C   CONVERT INFIX LIST TO POLISH PREFIX
       CALL PREORD(STACK,DUMY)
C   RETURN NAME OF PREFIX LIST
       READEX=DUMY
C   UPDATE COLLATOR LIST
       IHDR=LNKR(DUMY)
       I=LNKR(W(100))
391    I=MADRGT(I)
       IF(ID(I).EQ.2) GO TO 392
       IF(INTNAM.EQ.NDAT(I)) GO TO 393
       GO TO 391
392    I=NEWNOD(0,0,INTNAM,I)
393    DUMY=STRIND(INTNAM,I+2)
394    DUMY=PUTHAF(IHDR,I)
C   DELETE TEMPORARY INFIX LIST
       IDUM=IRALST(STACK)
       RETURN
C-----------------------------------------------------------------------
C   ERROR MESSAGE MODULE
400    PRINT 800,ISYM
       CALL WLKBCK(1)
410    PRINT 801,WORD
       CALL WLKBCK(1)
420    PRINT 802,BUFFER
       CALL WLKBCK(1)
430    PRINT 803,WORD
       GO TO 271
440    PRINT 804
       CALL WLKBCK(1)
450    PRINT 805
       CALL WLKBCK(1)
460    PRINT 806,WORD
       CALL WLKBCK(1)
800    FORMAT(' ILLEGAL SYMBOL ',A1,' ENCOUNTERED IN EXPRESSION')
801    FORMAT(' MORE THAN 1 DECIMAL POINT IN NUMBER ',A6)
802    FORMAT(' NUMBER ',3A6,' HAS MORE THAN 18 CHARACTERS')
803    FORMAT(' VARIABLE ',A6,' HAS MORE THAN 6 CHARACTERS - TRUNCATED')
804    FORMAT(' EXPRESSION MISFORMED: PARENTHESES DO NOT BALANCE')
805    FORMAT(' EXPRESSION MISFORMED: MISSING OPERAND(S) OR OPERATOR(S)')
806    FORMAT('  SUBEXPRESSION ',A6,' REFERENCED BUT NOT YET DEFINED')
       END
```

University of Cape Town

APPENDIX C

FLOWCHARTS OF THE DERIV ROUTINES

University of Cape Town

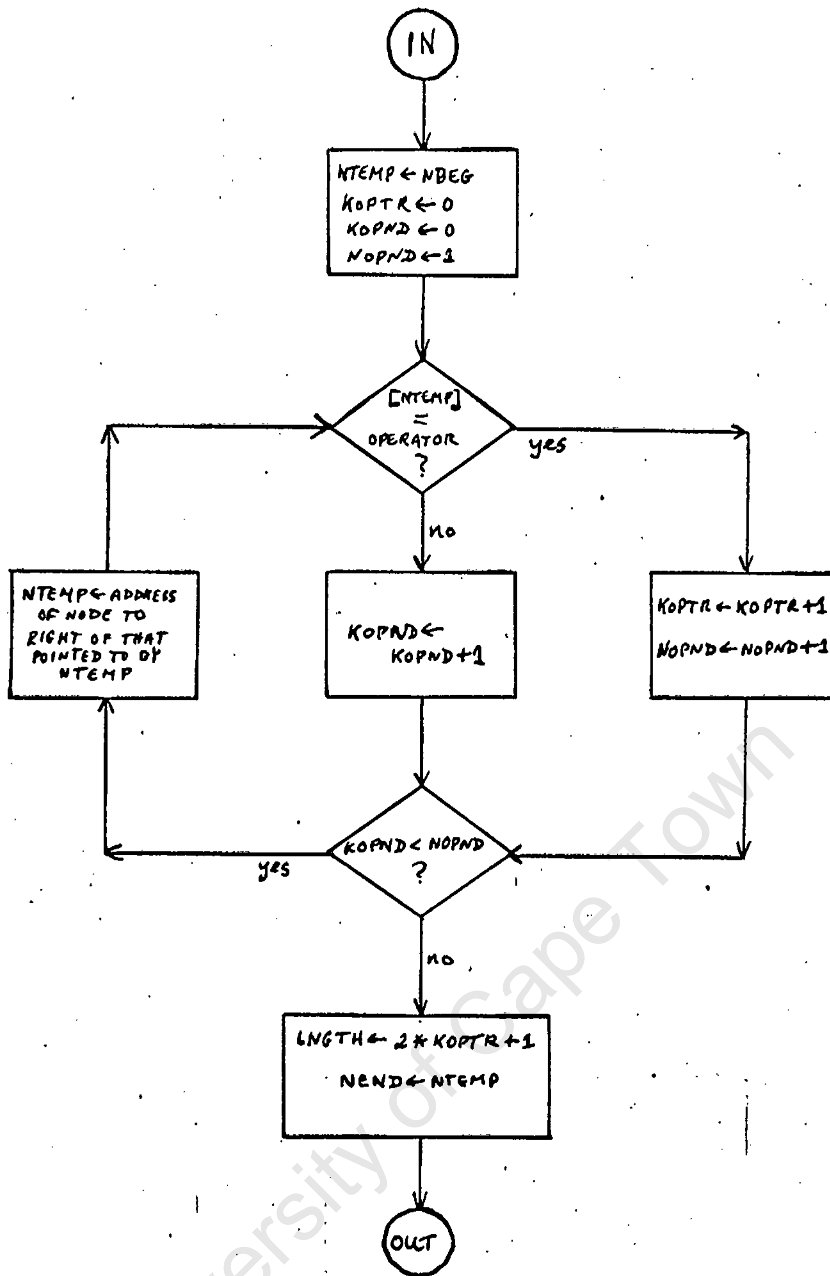


Fig C.1 . Flowchart of Subroutine HANDLE (NBEG, NEND, LNGTH). See listing in Appendix B. & description in Chapter 3. [ntemp] means the contents of the node of which ntemp holds the address. Nbeg holds the address of the start node of the operand to be delineated. Operand must be in Polish prefix ordering. Since the number of operators in the operand is one less than the number of operands of unit length, the number of nodes in the operand is KOPTR + KOPND, = 2* KOPTR + 1.

University of Cape Town

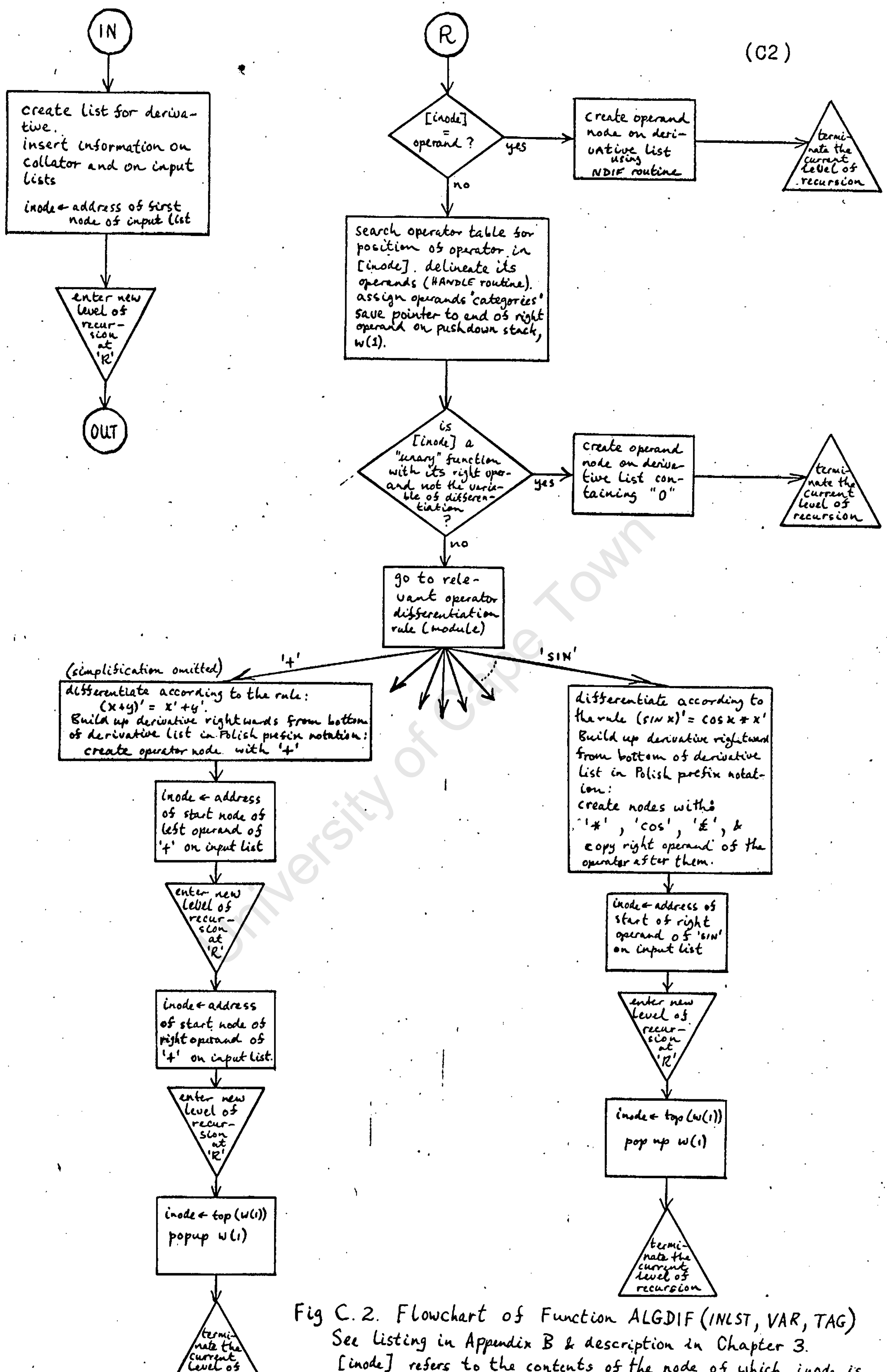


Fig C.2. Flowchart of Function ALGDIF (INLST, VAR, TAG)
See listing in Appendix B & description in Chapter 3.
[inode] refers to the contents of the node of which inode is

University of Cape Town

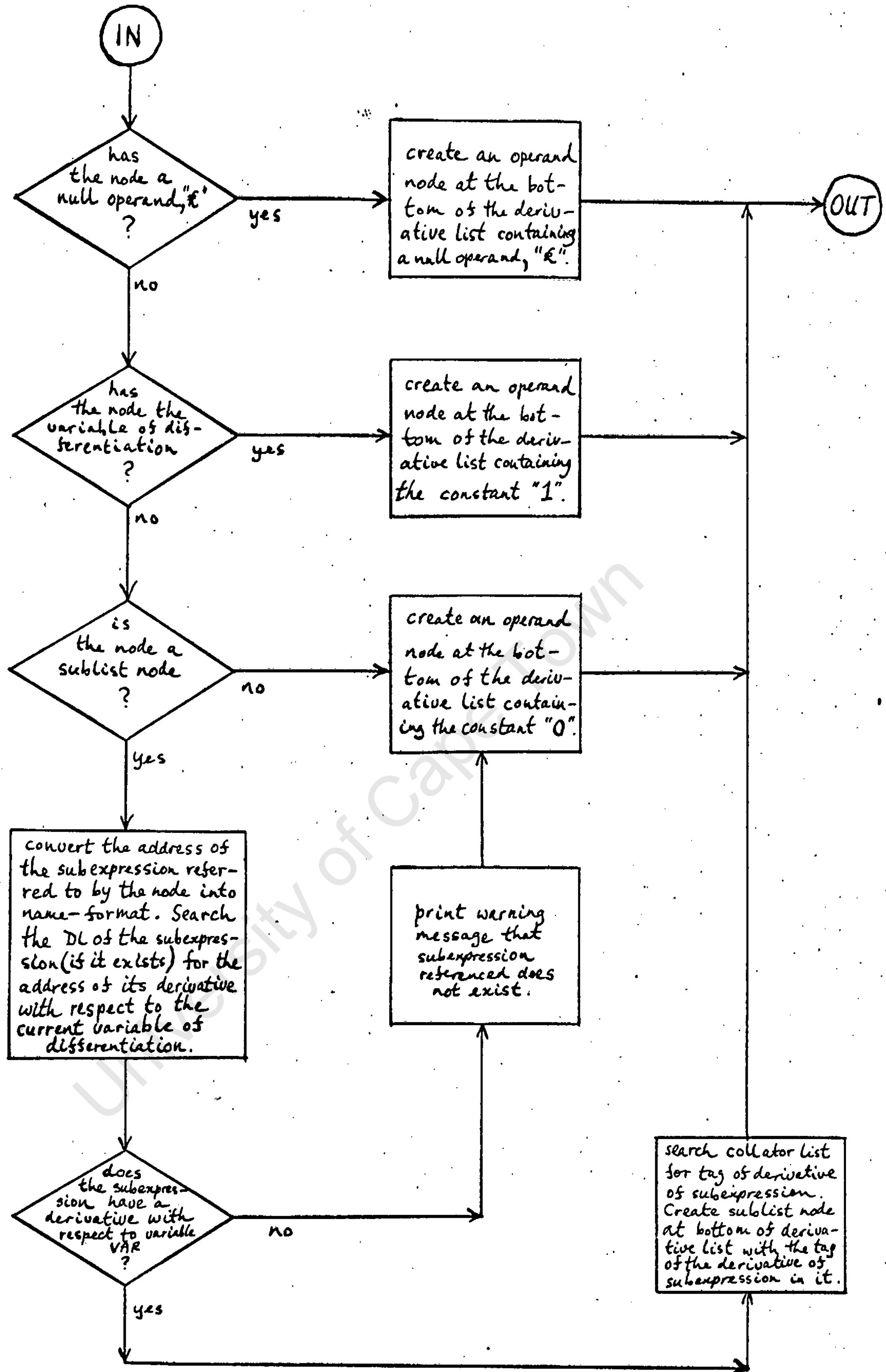
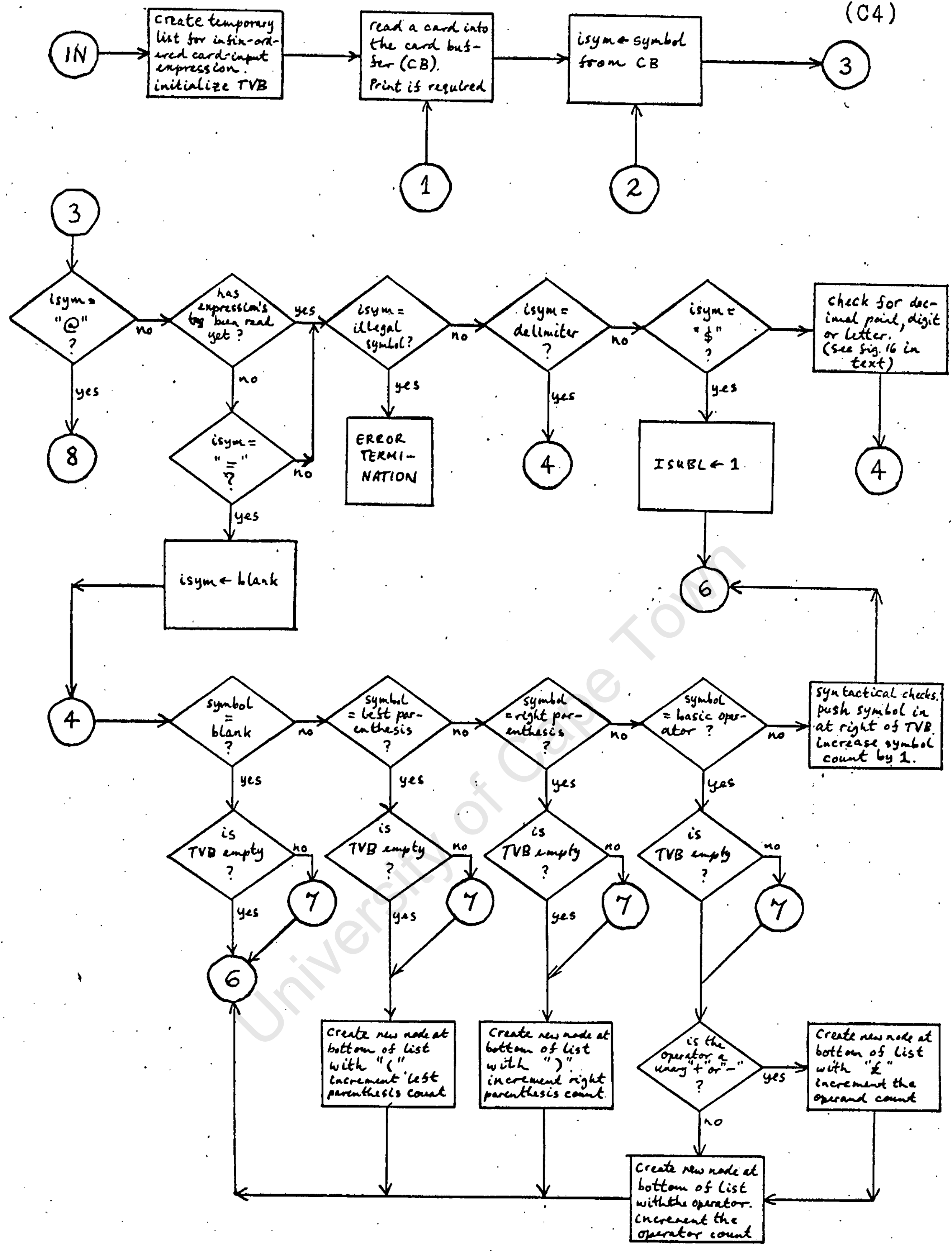


Fig. C.3. Flowchart of Function NDIF(INODE, INODED, VAR). See listing in Appendix B & description in Chapter 3. The node considered in the function is that on the input list pointed to by INODE. The derivative node is generated to the right of that pointed to by INODED on the derivative list. VAR is the variable of differentiation, input as Hollerith characters.

University of Cape Town



University of Cape Town

Fig. C.4. Flowchart of Function READEX(IPRT). (Continued Overleaf)
See listing in Appendix B & description in Chapter 3.
If IPRT is 9, each card image is printed as it is read.
CB refers to the card buffer into which the card image is read before interpretation. TVB is the mnemonic for the temporary variable buffer in which alphanumeric variables and numbers are built up character by character until a delimiter is encountered.

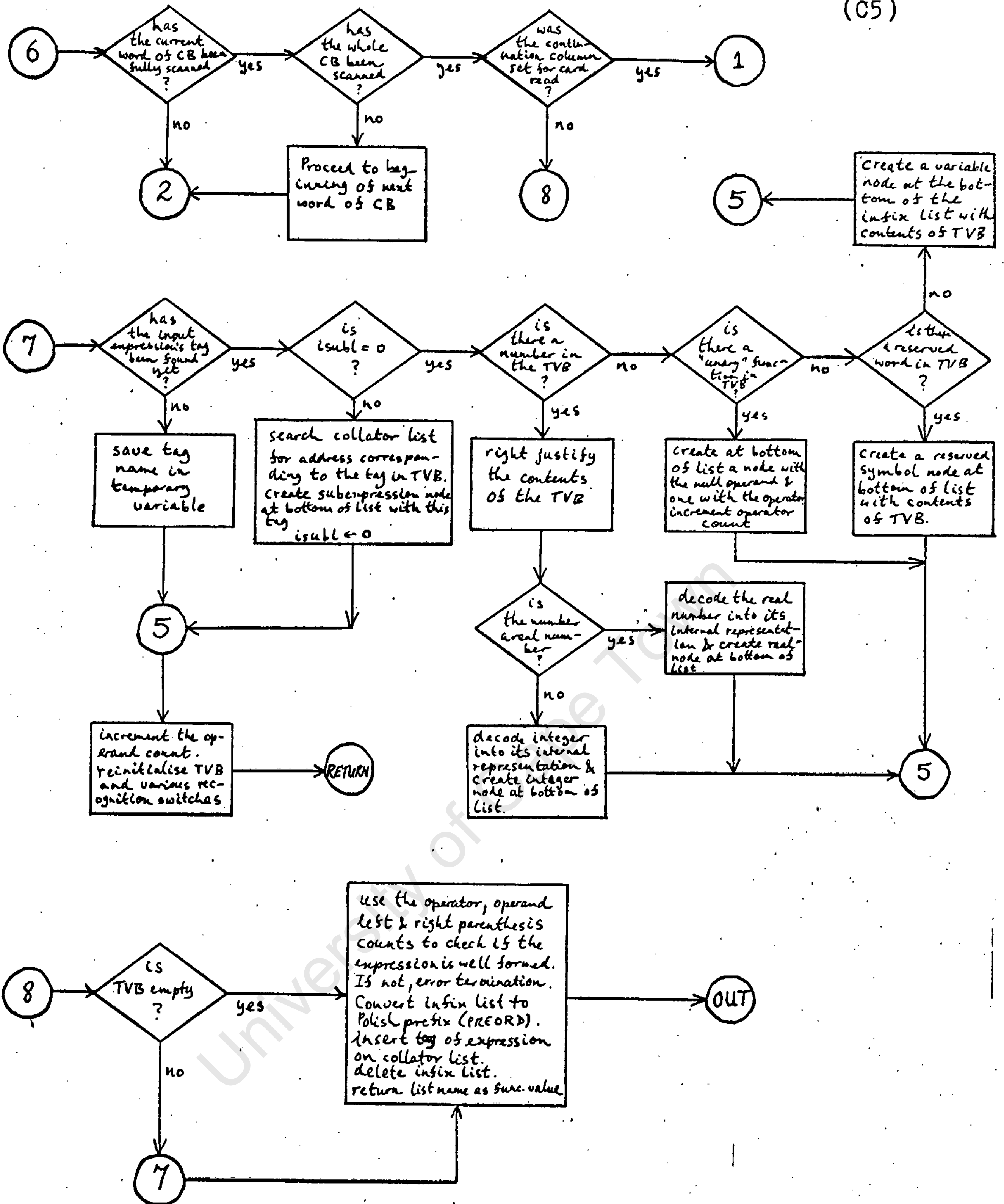


Fig. C.4 (continued). Flowchart of Function READEX (IPRT).

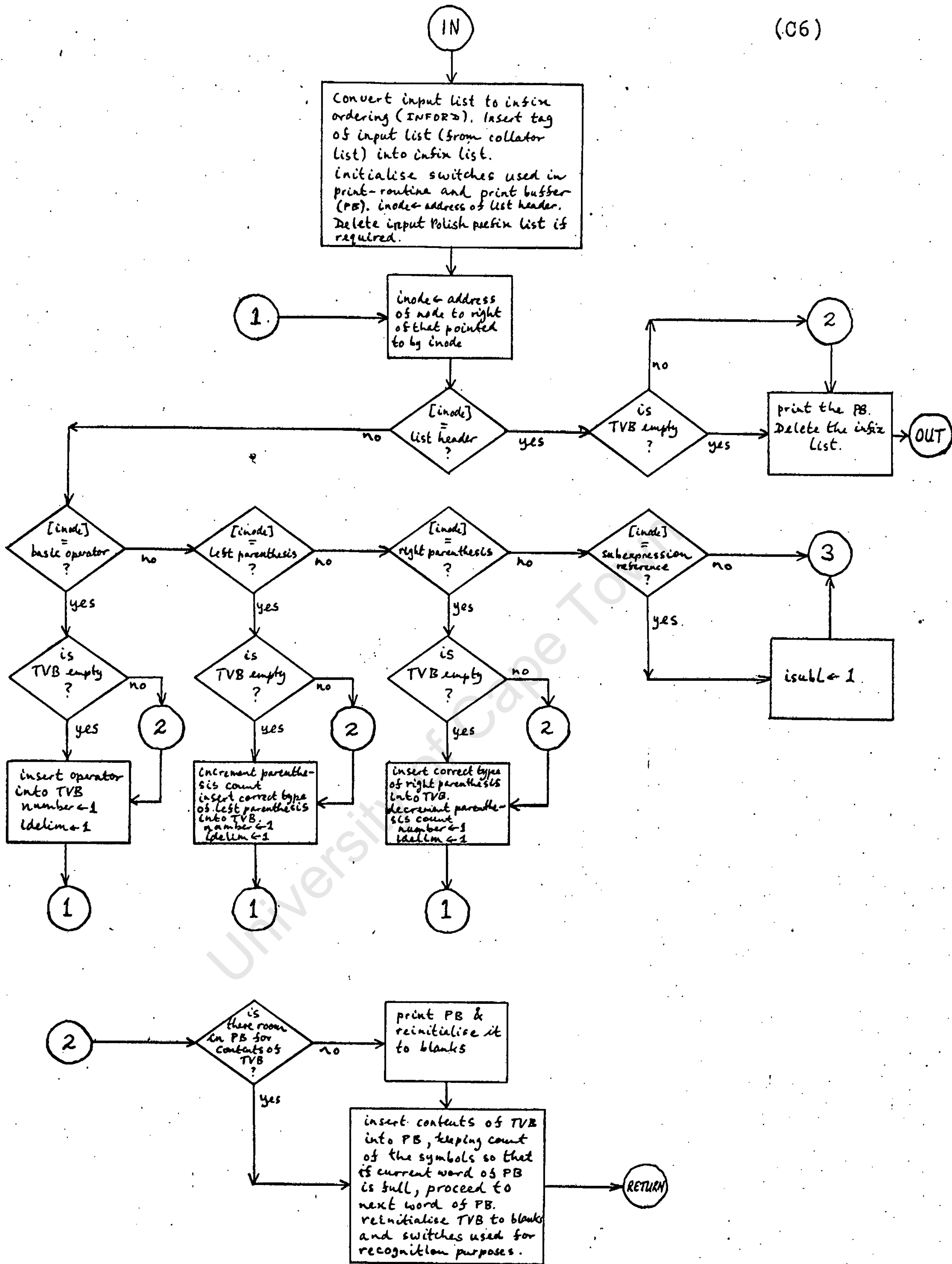
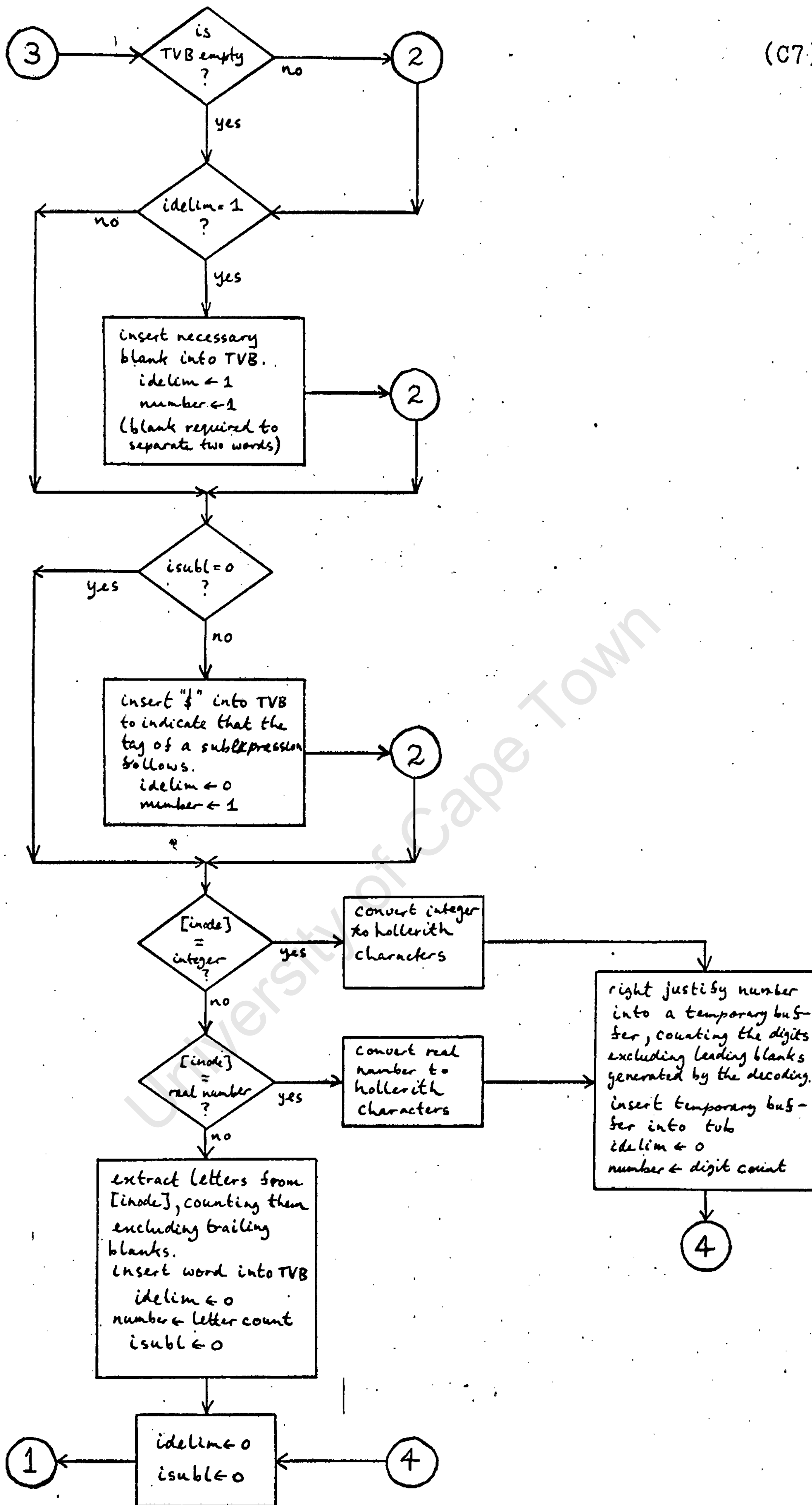


Fig. C.5. Flowchart of Subroutine PRNTEX(LST, IDEL). (continued overleaf)
See listing in Appendix B & description in Chapter 3.
If "idel" is 1, the input list LST is deleted by the subroutine.

University of Cape Town



University of Cape Town

Fig. C. 5 (continued). Flowchart of Subroutine PRNTEX.
[inode] refers to the contents of the node of which inode
contains the address. isubl = 1 if contents of TVB con-

APPENDIX D

EXAMPLES OF THE USE OF DERIV

University of Cape Town

```
C                       EXERCISE 1
C                       ----------
C     USED TO TEST THE ALGDIF DIFFERENTIATION RULES.
C
C     SET UP STORAGE FOR LIST OF AVAILABLE SPACE
      DIMENSION AVAIL(1000 )
C     LINKAGE TO CORE EFFICIENCY MEASURE. NODUTL KEEPS COUNT OF THE
C     NUMBER OF NODES CURRENTLY IN USE. MAXUTL MAINTAINS A COUNT
C     OF THE MAXIMUM NO. OF NODES USED IN A PROGRAM UP TO ANY POINT.
       COMMON /EFFIC/NODUTL,MAXUTL
      READ 10,M
C     SET UP AVAIL, SIZE M/3 NODES
      CALL INITAS(AVAIL,M)
C     N IS THE NUMBER OF INPUT EXPRESSIONS TO BE READ IN
      READ 10,N
10    FORMAT( )
C     PERFORM DIFFERENTIATION FOR EACH OF THE N INPUT EXPRESSIONS
      DO 20 I=1,N
C     READ EXPRESSION, ECHOING INPUT IMAGES (ARGUMENT IS 9)
      DUMY=READEX(0)
C     DIFFERENTIATE EXPRESSION WRT VARIABLE X
      DIFF=ALGDIF(DUMY,1HX,4HDIFF)
C     PRINT & DELETE INPUT EXPRESSION (ARGUMENT 2 IS ZERO)
      CALL PRNTEX(DUMY,0)
C     PRINT & DELETE DERIVATIVE
      CALL PRNTEX(DIFF,0)
      PRINT 15,MAXUTL,NODUTL
15     FORMAT(' MAX NO OF NODES USED =',I5/' NO OF NODES STILL TO BE',
     -' RETURNED TO AVAIL =',I5)
C     RESET MAXUTL TO ZERO SO AS TO TEST MAXIMUM CORE USAGE OF NEXT
C     EXPRESSION
       MAXUTL=0
20    CONTINUE
      PRINT 30
30    FORMAT(1H1)
      END
```

University of Cape Town

@MAP,SI   CALL UP THE COLLECTOR.
MAP 0023-09/16-11:41

1.   LIB SLIP*SLIP.

INSTRUCT COLLECTOR TO SEARCH SLIP "LIBRARY" FOR THE SUBROUTINES CALLED BY THE PROGRAM.
ADDRESSES ARE ALL OCTAL NUMBERS.

	INSTRUCTIONS:	DATA:
ADDRESS LIMITS	001000 044407	045000 057334
STARTING ADDRESS	044325	
WORDS DECIMAL	18184 IBANK	5341 DBANK

THEREFORE TOTAL CORE USED = <u>23525 WORDS</u>

SEGMENT MAIN			001000 044407		045000 057334
NBDCV$/FOR64	S	1	001000 001125	2	045000 045042
NFTV$/FOR	Y	1	001126 001150		
NCNVT$/FOR64	S	1	001151 001370	2	045043 045137
CARSAD$/62	T	1	001371 001573	2	045140 045567
CRELAD$/62	E	1	001574 002071	2	045570 046222
NOUT$/FOR67-2	M	1	002072 003076	2	046223 046251
NIOFR$/FOR67-3		1	003077 003253	2	046252 046407
NINPT$/FOR64	R	1	003254 004134	2	046410 046433
NFMT$/FOR67-1	O	1	004135 005010	2	046434 046510
NTAB$/FOR	U			2	046511 046547
FRU$/67	T				
SNOOPY/67	I	1	005011 012410	0	046550 046607
	N			2	046610 052142
NERR$/FOR67-1	E	1	012411 012777	2	052143 052322
NSTOP$/FOR62	S	1	013000 013023	2	052323 052332
NOSYM$/FOR67-1		1	013024 013237	2	052333 052336
NIFR$/FOR67-3		1	013240 013417	2	052337 052461
NISYM$/FOR67-1		1	013420 013611	2	052462 052465
NINTR$/FOR60		1	013612 013650	2	052466 052504
FNJ		1	013651 014072	0	052505 052525
	S			2	BLANK$COMMON
SRF	L	1	014073 014626	0	052526 052654
	I	3	SLIP	2	BLANK$COMMON
FND	P	1	014627 016434	0	052655 052665
	&			2	BLANK$COMMON
FNH	D	1	016435 017242	0	052666 053001
	E	3	SLIP	2	BLANK$COMMON
FNG	R	1	017243 020505	0	053002 053032
	I			2	BLANK$COMMON
NUCELL	V	1	020506 020636	0	053033 053057
		3	SLIP	2	BLANK$COMMON
				4	EFFIC
FNC		1	020637 022127	0	053060 053113
				2	BLANK$COMMON

University of Cape Town

(D3)

SLIP & DERIV ROUTINES

```
FNA                              1  022130 023214    0  053114 053125
                                                     2  BLANK$COMMON
FNT                              1  023215 023503    0  053126 053147
                                                     2  BLANK$COMMON
RECUR                            1  023504 023531    0  053150 053155
                                 3  SLIP             2  BLANK$COMMON
FNI                              1  023532 024206    0  053156 053204
                                 3  SLIP             2  BLANK$COMMON
FNK                              1  024207 024377    0  053205 053226
                                                     2  BLANK$COMMON
IRALST                           1  024400 024517    0  053227 053244
                                                     2  BLANK$COMMON
FNU                              1  024520 026007    0  053245 053355
                                 3  SLIP             2  BLANK$COMMON
FNB                              1  026010 030213    0  053356 053404
                                 3  SLIP             2  BLANK$COMMON
                                                     4  EFFIC
FNS                              1  030214 030505    0  053405 053423
                                                     2  BLANK$COMMON
FNE                              1  030506 031360    0  053424 053447
                                                     2  BLANK$COMMON
PRMTVS                           1  031361 031731    0  053450 053463
                                                     2  BLANK$COMMON
PRNTEX                           1  031732 033010    0  053464 053611
                                 3  SLIP             2  BLANK$COMMON
                                                     4  IODIFN
ALGDIF                           1  033011 042360    0  053612 054021
                                 3  SLIP             2  BLANK$COMMON
                                                     4  IODIFN
IODIFN (COMMON BLOCK)                                   054022 054137
READEX                           1  042361 043731    0  054140 054425
                                 3  SLIP             2  BLANK$COMMON
                                                     4  IODIFN
SLIP (COMMON BLOCK)                                     054426 054573
SRB                              1  043732 044324    0  054574 055325
                                 3  SLIP             2  BLANK$COMMON
                                                     4  EFFIC
EFFIC (COMMON BLOCK)                                    055326 055327
BLANK$COMMON (COMMON BLOCK)                             055330 055330
NAMES                            1  044325 044407    0  055331 057334
                                 3  EFFIC            2  BLANK$COMMON
```

MAIN PROGRAM

```
SYS$*RLIB$. LEVEL 67
END OF COLLECTION - TIME 12.903 SECONDS
```

University of Cape Town

```
@XQT    START EXECUTION. INPUT DATA: FIRST CARD 1000
                                     SECOND CARD 2
 A = X!X                             FOLLOWING CARDS TWO EXPRES-
                                     SIONS.


 DIFF = X!X*<1+LN X>


MAX NO OF NODES USED =    34
NO OF NODES STILL TO BE RETURNED TO AVAIL =    2

 B = X!LN X


 DIFF = LN X*X!<LN X-1>+X!LN X*LN X*1/X*1


MAX NO OF NODES USED =    78
NO OF NODES STILL TO BE RETURNED TO AVAIL =    4
```

University of Cape Town

```
C                          EXERCISE 2
C                          ----------
C   USED TO DEMONSTRATE DIFFERENTIATION WITH RESPECT TO A SUBEXPRESSION
C
C   DIMENSION STORAGE FOR LIST OF AVAILABLE SPACE.
        DIMENSION AVAIL(2000 )
C   LINKAGE TO CORE EFFICIENCY MEASURE.
        COMMON /EFFIC/NODUTL,MAXUTL
      READ 10,M
10    FORMAT( )
      PRINT 15
15    FORMAT(1H1)
C   SET UP AVAIL. SIZE M/3 NODES.
      CALL INITAS(AVAIL,M)
C   READ IN A=A(X) WITH TAG A. LIST-NAME IS ALSO A.
      A=READEX(9)
C   READ IN B=B(A,X).
      B=READEX(9)
C   DIFFERENTIATE B WRT A, GIVING EXPRESSION C WITH TAG ALSO 'C'.
      C=ALGDIF(B,1HA,1HC)
C   PRINT EXPRESSION C. ARGUMENT 2 BEING NON-ZERO PREVENTS THE LIST
C   FROM BEING DELETED AFTER PRINTING.
      CALL PRNTEX(C,9)
       PRINT 20,MAXUTL
20     FORMAT(' MAX NO OF NODES USED=',I5)
      PRINT 15
      END
```

University of Cape Town

```
A=SINH(ACOSH X)@
B=$A!(TANH $A)@

 C = TANH $A*$A!<TANH $A-1>+$A!TANH $A*LN $A*SECH $A!2*1


MAX NO OF NODES USED=   96
```

University of Cape Town

```
C                       EXERCISE 3(A)
C                       -------------
C   USED TO TEST TIME & CORE EFFICIENCY OF A LARGE INPUT EXPRESSION
C   WHERE IT IS NOT BROKEN UP INTO SUBEXPRESSIONS.
C
C   RESERVE STORAGE FOR LIST OF AVAILABLE SPACE
      DIMENSION AVAIL(7000)
C   LINKAGE TO CORE EFFICIENCY MEASURE
       COMMON /EFFIC/NODUTL,MAXUTL
      PRINT 5
5     FORMAT(1H1)
C   SET UP AVAIL
      CALL INITAS (AVAIL,7000)
C   READ EXPRESSION INTO LIST A
      A=READEX(0)
C   DIFFERENTIATE A WRT VARIABLE I
      DA=ALGDIF(A,1HI,2HDA)
C   PRINT & DELETE EXPRESSION READ IN
      CALL PRNTEX(A,0)
C   PRINT & DELETE DERIVATIVE EXPRESSION
      CALL PRNTEX(DA,0)
       PRINT 10,MAXUTL
10     FORMAT(' MAX NO OF NODES USED',I5)
      PRINT 5
      END
```

University of Cape Town

```
A =<G•SIN I!2+H•COS I!2+F+(<F!2+2•F•(G•SIN I!2+H•COS I!2)+G!2•SIN I!2+H!2•COS I!2>•<1+2•T•(G•SIN I!2+H•COS I!2)+T!2•(G!2•SIN
I!2+H!2•COS I!2)>)!.50000+T•(F•<G•SIN I!2+H•COS I!2>+G!2•SIN I!2+H!2•SIN I!2)>/<(1-F•T)•(G•SIN I!2+H•COS I!2+F+<F!2+2•F•(G•SIN
I!2+H•COS I!2)+G!2•SIN I!2+H!2•COS I!2>!.50000)>


DA =<(2•SIN I!1•COS I•1•G+2•COS I!1•<-SIN I•1>•H+.50000•<(F!2+2•F•<G•SIN I!2+H•COS I!2>+G!2•SIN I!2+H!2•COS I!2)•(1+2•T•<G•
SIN I!2+H•COS I!2>+T!2•<G!2•SIN I!2+H!2•COS I!2>)>!-.50000•<(0+0•<G•SIN I!2+H•COS I!2>•<2•SIN I!1•COS I•1•G+2•COS I!1•(-SIN I•
1)•H>•2•F+0•SIN I!2+2•SIN I!1•COS I•1•G!2+0•COS I!2+2•COS I!1•<-SIN I•1>•H!2)•(1+2•T•<G•SIN I!2+H•COS I!2>+T!2•<G!2•SIN I!2+H!
2•COS I!2>)+(0•<G•SIN I!2+H•COS I!2>•<2•SIN I!1•COS I•1•G+2•COS I!1•(-SIN I•1)•H>•2•T+0•<G!2•SIN I!2+H!2•COS I!2>+<0•SIN I!2+2
•SIN I!1•COS I•1•G!2+0•COS I!2+2•COS I!1•(-SIN I•1)•H!2>•T!2)•(F!2+2•F•<G•SIN I!2+H•COS I!2>+G!2•SIN I!2+H!2•COS I!2)>•<(2•SIN
I!1•COS I•1•G+2•COS I!1•<-SIN I•1>•H)•F+0•SIN I!2+2•SIN I!1•COS I•1•G!2+0•SIN I!2+2•SIN I!1•COS I•1•H!2>•T)•(1-F•T)•(G•SIN I!
2+H•COS I!2+F+<F!2+2•F•(G•SIN I!2+H•COS I!2)+G!2•SIN I!2+H!2•COS I!2>!.50000)-(<-0>•<G•SIN I!2+H•COS I!2+F+(F!2+2•F•<G•SIN I!2
+H•COS I!2>+G!2•SIN I!2+H!2•COS I!2)!.50000>+<2•SIN I!1•COS I•1•G+2•COS I!1•(-SIN I•1)•H+.50000•(F!2+2•F•<G•SIN I!2+H•COS I!2>
+G!2•SIN I!2+H!2•COS I!2)!-.50000•(0+0•<G•SIN I!2+H•COS I!2>•<2•SIN I!1•COS I•1•G+2•COS I!1•(-SIN I•1)•H>•2•F+0•SIN I!2+2•SIN
I•1•COS I•1•G!2+0•COS I!2+2•COS I!1•<-SIN I•1>•H!2)>•<1-F•T>)•(G•SIN I!2+H•COS I!2+F+<(F!2+2•F•<G•SIN I!2+H•COS I!2>+G!2•SIN I
!2+H!2•COS I!2)•(1+2•T•<G•SIN I!2+H•COS I!2>+T!2•<G!2•SIN I!2+H!2•COS I!2>)>!.50000+T•<F•(G•SIN I!2+H•COS I!2)+G!2•SIN I!2+H!2
•SIN I!2>)>/<(1-F•T)•(G•SIN I!2+H•COS I!2+F+<F!2+2•F•(G•SIN I!2+H•COS I!2)+G!2•SIN I!2+H!2•COS I!2>!.50000)>!2


MAX NO OF NODES USED 2274
```

University of Cape Town

```
C                           EXERCISE 3(B)
C                           -------------
C     AS EX. 3(A), BUT LARGE EXPRESSION INPUT IS BROKEN UP INTO FOUR
C     SUBEXPRESSIONS. ALL INPUT IS PERFORMED FOLLOWED IN SEQUENCE BY ALL
C     DIFFERENTIATION AND PRINTING.
C
C     SET UP STORAGE FOR AVAIL.
      DIMENSION AVAIL(2000 )
C     LINKAGE TO CORE EFFICIENCY MEASURE
      COMMON /EFFIC/NODUTL,MAXUTL
C     A ARRAY HAS THE NAMES OF THE INPUT EXPRESSIONS, DA THOSE OF THEIR
C     DERIVATIVES. THE NAMES-ARRAY CONTAINS THE TAGS OF THE DERIVATIVES.
      DIMENSION A(4),DA(4),NAMES(4)/2HDA,2HDB,2HDC,2HDD/
C     SET UP AVAIL.
      CALL INITAS(AVAIL,2000 )
      PRINT 20
C     READ THE INPUT EXPRESSIONS IN THE CORRECT ORDER.
      DO 1 I=1,4
1     A(I)=READEX(0)
C     DIFFERENTIATE THE INPUT EXPRESSIONS WRT I IN CORRECT ORDER.
      DO 2 I=1,4
2     DA(I)=ALGDIF(A(I),1HI,NAMES(I))
C     PRINT & DELETE THE INPUT EXPRESSIONS IN SEQUENCE
      DO 3 I=1,4
3     CALL PRNTEX(A(I),0)
C     PRINT & DELETE THE DERIVATIVES IN SEQUENCE
      DO 4 I=1,4
4     CALL PRNTEX(DA(I),0)
      PRINT 10,MAXUTL
10    FORMAT(' MAX NO OF NODES USED=  ',I5)
      PRINT 20
20    FORMAT(1H1)
      END
```

University of Cape Town

(D10)

```
A = G*SIN I!2+H*COS I!2

B =<G*SIN I>!2+<H*COS I>!2

C = F!2+2*F*$A+$B

D =<$A+F+($C*<1+2*T*$A+T!2*$B>)!.50000+T*(F*$A+$B)>/<(1-F*T)*($A+F+$C!.50000)>

DA = 2*SIN I!1*COS I*1*G+2*COS I!1*<-SIN I*1>*H

DB = 2*<G*SIN I>!1*COS I*1*G+2*<H*COS I>!1*<-SIN I*1>*H

DC = D+D*$A+$DA*2*F+$DB

DD =<($DA+.50000*<$C*(1+2*T*$A+T!2*$B)>!-.50000*<$DC*(1+2*T*$A+T!2*$B)+(D*$A+$DA*2*T+D*$B+$DB*T!2)*$C>+<$DA*F+$DB>*T)*(1-F*T)
*($A+F+$C!.50000)-(<-D>*<$A+F+$C!.50000>+<$DA+.50000*$C!-.50000*$DC>*<1-F*T>)*($A+F+<$C*(1+2*T*$A+T!2*$B)>!.50000+T*<F*$A+$B>)
>/<(1-F*T)*($A+F+$C!.50000)>!2

MAX NO OF NODES USED=    379
```

University of Cape Town

```
C                      EXERCISE 3(C)
C                      -------------
C   AS EX. 3(B), BUT INPUT/OUTPUT AND DIFFERENTIATION ARE PERFORMED
C   IN SEQUENCE FOR EACH OF THE INPUT EXPRESSIONS.
C
      DIMENSION AVAIL(2000)
      DIMENSION A(4),DA(4),NAMES(4)/2HDA,2HDB,2HDC,2HDD/
      COMMON /EFFIC/NODUTL,MAXUTL
      PRINT 20
      CALL INITAS(AVAIL,2000)
      DO 1 I=1,4
      A(I)=READEX(0)
      DA(I)=ALGDIF(A(I),1HI,NAMES(I))
      CALL PRNTEX(A(I),9)
1     CALL PRNTEX(DA(I),9)
      PRINT 10,MAXUTL
10    FORMAT(' MAX NO OF NODES USED=',I5)
      PRINT 20
20    FORMAT(1H1)
      END
```

University of Cape Town

```
A = G*SIN I!2+H*COS I!2

DA = 2*SIN I!1*COS I*1*G+2*COS I!1*<-SIN I*1>*H

B =<G*SIN I>!2+<H*COS I>!2

DB = 2*<G*SIN I>!1*COS I*1*G+2*<H*COS I>!1*<-SIN I*1>*H

C = F!2+2*F*SA+SB

DC = 0+0*SA+SDA*2*F+SDB

D =<SA+F+(SC*<1+2*T*SA+T!2*SB>)!.50000+T*(F*SA+SB)>/<(1-F*T)*(SA+F+SC!.50000)>

DD =<(SDA+.50000*<SC*(1+2*T*SA+T!2*SB)>!-.50000*<SDC*(1+2*T*SA+T!2*SB)+(0*SA+SDA*2*T+0*SB+SDB*T!2)*SC>+<SDA*F+SDB>*T)*(1-F*T)
*(SA+F+SC!.50000)-(<-0>*<SA+F+SC!.50000>+<SDA+.50000*SC!-.50000*SDC>*<1-F*T>)*(SA+F+<SC*(1+2*T*SA+T!2*SB)>!.50000+T*<F*SA+SB>)
>/<(1-F*T)*(SA+F+SC!.50000)>!2

MAX NO OF NODES USED=  562
```

```
C                    EXERCISE 4(A)
C                    -------------
C  PARTIAL DIFFERENTIATION EXAMPLE. PRINTING OF ALL EXPRESSIONS
C  IS PERFORMED AT THE END OF DIFFERENTIATION.
C
      DIMENSION AVAIL(3000)
      COMMON /EFFIC/NODUTL,MAXUTL
C  SET UP AVAIL.
      CALL INITAS(AVAIL,3000)
      PRINT 5
5     FORMAT(1H1)
C  READ IN A=A(X).
      A=READEX(0)
C  DIFFERENTIATE A WRT X YIELDING AX= A/ X.
      AX=ALGDIF(A,1HX,2HAX)
C  DIFFERENTIATE AX WRT X YIELDING AXX=  A/ X .
      AXX=ALGDIF(AX,1HX,3HAXX)
C  DIFFERENTIATE AX WRT Y YIELDING AXY=  A/ X Y.
      AXY=ALGDIF(AX,1HY,3HAXY)
C  DIFFERENTIATE A TWICE WRT Y YIELDING AYY=  A/ Y .
      AYY=ALGDIF(ALGDIF(A,1HY,2HAY),1HY,3HAYY)
C  PRINT & DELETE ALL THE EXPRESSIONS IN SEQUENCE.
      CALL PRNTEX(A,0)
      CALL PRNTEX(AXX,0)
      CALL PRNTEX(AXY,0)
      CALL PRNTEX(AYY,0)
      PRINT 10,MAXUTL
10    FORMAT(' MAX NO OF NODES USED =',I5)
      PRINT 5
      END
```

University of Cape Town

(D14)

```
A = X!LOG<X+Y>

AXY = LOG E*1/<X+Y>*1*X!<LOG(X+Y)-1>+<(LOG<X+Y>-1)*X!(LOG<X+Y>-1-1)+X!(LOG<X+Y>-1)*LN X*LOG E*1/(X+Y)*1>*LOG<X+Y>+<LOG(X+Y)*X
!(LOG<X+Y>-1)+X!LOG(X+Y)*LN X*LOG E*1/(X+Y)*1>*LN X*LOG E*1/<X+Y>*1+<1/X*1*LOG E*1/(X+Y)*1+(LOG E*1/E*0*1/<X+Y>*1+<-1*1>/<X+Y>
!2*1*LOG E)*LN X>*X!LOG<X+Y>

AXY = LOG E*1/<X+Y>*1*X!<LOG(X+Y)-1>+X!<LOG(X+Y)-1>*LN X*LOG E*1/<X+Y>*1*LOG<X+Y>+X!LOG<X+Y>*LN X*LOG E*1/<X+Y>*1*LN X*LOG E*
1/<X+Y>*1+<1/X*0*LOG E*1/(X+Y)*1+(LOG E*1/E*0*1/<X+Y>*1+<-1*1>/<X+Y>!2*1*LOG E)*LN X>+X!LOG<X+Y>

AYY = X!LOG<X+Y>*LN X*LOG E*1/<X+Y>*1*LN X*LOG E*1/<X+Y>*1+<1/X*0*LOG E*1/(X+Y)*1+(LOG E*1/E*0*1/<X+Y>*1+<-1*1>/<X+Y>!2*1*LOG
E)*LN X>*X!LOG<X+Y>

MAX NO OF NODES USED =  792
```

```
C                      EXERCISE 4(B)
C                      -------------
C   AS EX. 4(A), BUT PRINTING OF EXPRESSIONS IS PERFORMED
C   IMMEDIATELY AFTER THEIR DIFFERENTIATION, AND IF THEY ARE NOT
C   NEEDED FURTHER IN THE PROGRAM, THEY ARE THEN DELETED.
C
      DIMENSION AVAIL(2000)
      COMMON /EFFIC/NODUTL,MAXUTL
      PRINT 20
      CALL INITAS(AVAIL,2000)
      A=READEX(0)
      CALL PRNTEX(A,9)
      AX=ALGDIF(A,1HX,2HAX)
      AXX=ALGDIF(AX,1HX,3HAXX)
      CALL PRNTEX(AXX,0)
      AXY=ALGDIF(AX,1HY,3HAXY)
      CALL PRNTEX(AXY,0)
      AYY=ALGDIF(ALGDIF(A,1HY,2HAY),1HY,3HAYY)
      CALL PRNTEX(AYY,0)
      PRINT 10,MAXUTL
   10 FORMAT(' MAX NO OF NODES USED =',I5)
      PRINT 20
   20 FORMAT(1H1)
      END
```

University of Cape Town

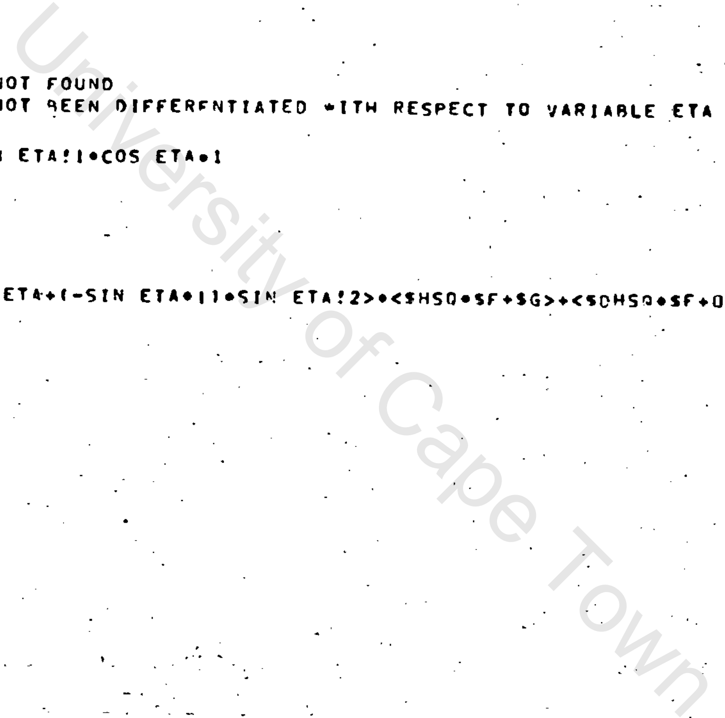
(D16)

```
A = X!LOG<X+Y>

AXX = LOG E•1/<X+Y>•1•X!<LOG(X+Y)-1>+<(LOG<X+Y>-1)•X!(LOG<X+Y>-1-1)+X!(LOG<X+Y>-1)•LN X•LOG E•1/(X+Y)•1>•LOG<X+Y>+<LOG(X+Y)•X
!(LOG<X+Y>-1)+X!LOG(X+Y)•LN X•LOG E•1/(X+Y)•1>•LN X•LOG E•1/<X+Y>•1+<1/X•1•LOG E•1/(X+Y)•1+(LOG E•1/E•0•1/<X+Y>•1+<-1•1>/<X+Y>
!2•1•LOG E)•LN X>•X!LOG<X+Y>

AXY = LOG E•1/<X+Y>•1•X!<LOG(X+Y)-1>+X!<LOG(X+Y)-1>•LN X•LOG E•1/<X+Y>•1•LOG<X+Y>+X!LOG<X+Y>•LN X•LOG E•1/<X+Y>•1•LN X•LOG E•
1/<X+Y>•1+<1/X•0•LOG E•1/(X+Y)•1+(LOG E•1/E•0•1/<X+Y>•1+<-1•1>/<X+Y>!2•1•LOG E)•LN X>•X!LOG<X+Y>

AYY = X!LOG<X+Y>•LN X•LOG E•1/<X+Y>•1•LN X•LOG E•1/<X+Y>•1+<1/Y•0•LOG E•1/(X+Y)•1+(LOG E•1/E•0•1/<X+Y>•1+<-1•1>/<X+Y>!2•1•LOG
E)•LN X>•X!LOG<X+Y>

MAX NO OF NODES USED = 507
```

University of Cape Town

```
C                     EXERCISE 5
C                     ----------
C   LARGE DIFFERENTIATION EXAMPLE WHERE THE EXPRESSION IS BROKEN UP
C   INTO 13 SUBEXPRESSIONS. THE MAIN EXPRESSION HAS THE TAG CHI1.
C   D(CHI1)/D(XI) AND D(CHI1)/D(ETA) ARE REQUIRED.
C
      DIMENSION AVAIL(5000)
C   LINKAGE TO CORE EFFICIENCY MEASURE.
      COMMON /EFFIC/NODUTL,MAXUTL
C   A ARRAY HAS LIST-NAMES OF THE INPUT EXPRESSIONS. DA THOSE OF
C   THEIR DERIVATIVES. NAMES CONTAINS THE TAGS OF THE DERIVATIVES.
      DIMENSION A(13),DA(13),NAMES(13)/3HDMU,4HDHSQ,3HDF1,3HDF2,3HDF3,
     -3HDH1,3HDH2,3HDH3,3HDH4,5HDBIGH,2HDG,2HDF,5HDCHI1/
C   SET UP THE LIST OF AVAILABLE SPACE.
      CALL INITAS(AVAIL,5000)
      PRINT 20
C   READ INPUT EXPRESSIONS IN CORRECT SEQUENCE.
      DO 1 I=1,13
C   ARGUMENT OF READEX CALL IS 9 SO INPUT IMAGES ARE ECHOED .
C   IN PRINTOUT THUS ELIMINATING NEED TO PRINT INPUT EXPRESSIONS
C   USING PRNTEX.
1     A(I)=READEX(9)
      PRINT 2
2     FORMAT('1DIFFERENTIATE CHI1 WRT ETA')
C   DIFFERENTIATE IN CORRECT SEQUENCE WRT ETA (ONLY HSQ & CHI1
C   ARE FUNCTIONS OF ETA, SO THE OTHER SUBEXPRESSIONS DO NOT
C   HAVE TO BE DIFFERENTIATED WRT ETA).
      DA(2)=ALGDIF(A(2),3HETA,NAMES(2))
      DA(13)=ALGDIF(A(13),3HETA,NAMES(13))
C   PRINT & DELETE DERIVATIVES IN SEQUENCE.
      CALL PRNTEX(DA(2),0)
      CALL PRNTEX(DA(13),0)
      PRINT 3
3     FORMAT('1DIFFERENTIATE CHI1 WRT XI')
C   DIFFERENTIATE WRT XI IN CORRECT SEQUENCE.
      DO 4 I=1,13
4     DA(I)=ALGDIF(A(I),2HXI,NAMES(I))
C   PRINT & DELETE DERIVATIVES IN SEQUENCE.
      DO 5 I=1,13
5     CALL PRNTEX(DA(I),0)
      PRINT 10,MAXUTL
10    FORMAT(' MAX NO OF NODES USED =',I5)
      PRINT 20
20    FORMAT(1H1)
      END
```

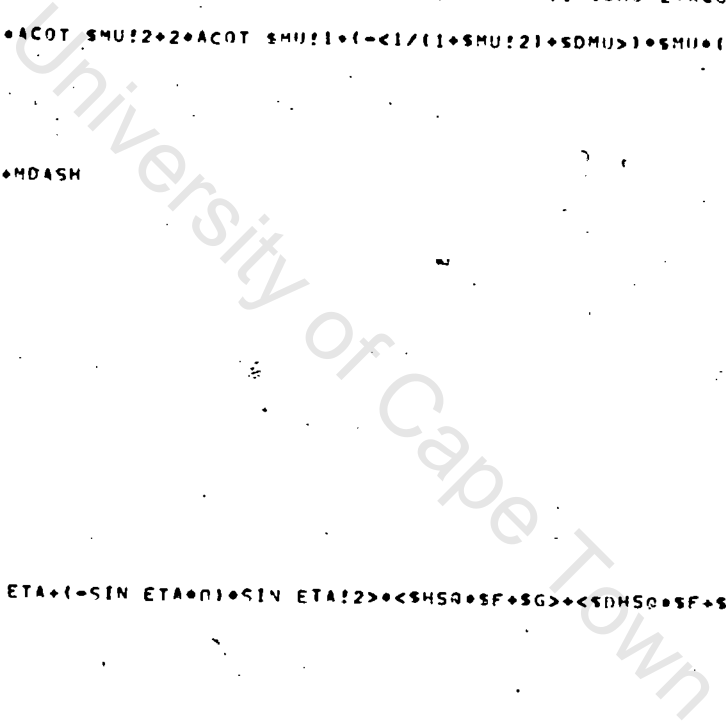
University of Cape Town




```
DIFFERENTIATE CHI1 WRT ETA


LIST-NAME = 062976062976

ATTRIBUTE-VALUE LIST REQUIRED BUT NOT FOUND
THE REFERENCED SUBLIST F       HAS NOT BEEN DIFFERENTIATED WITH RESPECT TO VARIABLE ETA


LIST-NAME = 062363062363

ATTRIBUTE-VALUE LIST REQUIRED BUT NOT FOUND
THE REFERENCED SUPLIST G       HAS NOT BEEN DIFFERENTIATED WITH RESPECT TO VARIABLE ETA

 DHSQ = 2•COSH XI!1•SINH XI•0-2•SIN ETA!1•COS ETA•1



 DCHI1 =<2•SIN ETA!1•COS ETA•1•COS ETA+(-SIN ETA•1)•SIN ETA!2>•<$HSQ•$F+$G>+<$DHSQ•$F+D•$HSQ+D>•SIN ETA!2•COS ETA
```

```
DIFFERENTIATE CHII WRT XI
 DMU = COSH XI*1


 DMSQ = 2*COSH XI!1*SINH XI*1-2*SIN ETA!1*COS ETA*0


 DF1 = 4*SMU!3*SDMU*7+2*SMU!1*SDMU*7.6667+-<(SDMU*<7*SMU!4+10*SMU!2+3>+<4*SMU!3*SDMU*7+2*SMU!1*SDMU*10>*SMU)*ACOT SMU+(-<1/(1+
SMU!2)+SDMU>)*SMU*(7*SMU!4+10*SMU!2+3)>


 DF2 = 2*SMU!1*SDMU+-<(SDMU*<1+SMU!2>+2*SMU!1*SDMU*SMU)*ACOT SMU+(-<1/(1+SMU!2)+SDMU>)*SMU*(1+SMU!2)>


 DF3 =<3*SMU!2*SDMU*1.1250+SDMU-(<4*SMU!3*SDMU*9+2*SMU!1*SDMU*11>*.25000*ACOT SMU+<-(1/<1+SMU!2>+SDMU)>*.25000*<9*SMU!4+11*SMU
!2+2>)+(SDMU*.12500*<9*SMU!4+14*SMU!2+5>+<4*SMU!3*SDMU*9+2*SMU!1*SDMU*14>*.12500*SMU)*ACOT SMU!2+2*ACOT SMU!1*(-<1/(1+SMU!2)+S
DMU>)*.12500*SMU*(9*SMU!4+14*SMU!2+5)>*FX102

 DH1 =-6*SMU!5*SDMU*7-4*SMU!3*SDMU*10.667-2*SMU!1*SDMU*3.0667+<3*SMU!2*SDMU*(7*SMU!4+13*SMU!2+6)+(4*SMU!3*SDMU*7+2*SMU!1*SDMU*
13)*SMU!3>*ACOT SMU+<-(1/<1+SMU!2>+SDMU)>*SMU!3*<7*SMU!4+13*SMU!2+6>


 DH2 =-4*SMU!3*SDMU*2-2*SMU!1*SDMU*1.3333+<3*SMU!2*SDMU*2*(1+SMU!2)+2*SMU!1*SDMU*2*SMU!3>*ACOT SMU+<-(1/<1+SMU!2>+SDMU)>*2*SMU
!3*<1+SMU!2>


 DH3 = SDMU*FX102*<-1.1250*SMU!4-1.6250*SMU!2-.50000+.25000*SMU*(9*SMU!4+16*SMU!2+7)*ACOT SMU-.12500*(9*SMU!6+19*SMU!4+11*SMU!
2+1)*ACOT SMU!2>+<-4*SMU!3*SDMU*1.1250-2*SMU!1*SDMU*1.6250+(<SDMU*.25000*(9*SMU!4+16*SMU!2+7)+(4*SMU!3*SDMU*9+2*SMU!1*SDMU*16)
```

```
•.25000•SMU>•ACOT SMU+<-(1/<1+SMU!2>+SDMU)>•.25000+SMU•<9•SMU!4+16•SMU!2+7>)-(<6•SMU!5•SDMU•9+4•SMU!3•SDMU•19+2•SMU!1•SDMU•11>
•.12500•ACOT SMU!2+2•ACOT SMU!1•<-(1/<1+SMU!2>+SDMU)>•.12500•<9•SMU!6+19•SMU!4+11•SMU!2+1>)>•SMU•FX102


DH4 = D•<SMU-(1+2•SMU!2)•ACOT SMU+SMU•(1+SMU!2)•ACOT SMU!2>+<SDMU-(2•SMU!1•SDMU•2•ACOT SMU+<-(1/<1+SMU!2>+SDMU)>•<1+2•SMU!2>)
+(SDMU•<1+SMU!2>+2•SMU!1•SDMU•SMU)•ACOT SMU!2+2•ACOT SMU!1•(-<1/(1+SMU!2)+SDMU>)•SMU•(1+SMU!2)>•2•FX102


DSIGH = SDH1•A1+SDH2•A2+SDH3+SDH4+HDASH


DG = SDF2•B2+SDRIGH


DF = SDF1•A1+SDF2•A2+SDF3


DCHI1 =<2•SIN ETA!1•COS ETA•D•COS ETA+(-SIN ETA•D)•SIN ETA!2>•<SHSQ•SF+SG>+<SDHSQ•SF+SDF•SHSQ+SDG>•SIN ETA!2•COS ETA


MAX NO OF NODES USED = 1494
```



INDEX OF FUNCTION LISTINGS

routine	page
ADVLEL	A10
ADVLER	A10
ADVLL	A7
ADVLNL	A10
ADVLNR	A10
ADVLR	A7
ADVLWL	A10
ADVLWR	A10
ADVSEL	A10
ADVSER	A11
ADVSL	A7
ADVSNL	A10
ADVSNR	A11
ADVSR	A8
ADVSWL	A11
ADVSWR	A11
ALGDIF	B1
BOT	A1
CONT	A26
DELETE	A9
* DERROR	A33
EQUAL	A9
GETFLG	A26
GETHAF	A26
GETMRK	A26
* HANDLE	B17
ID	A27
IEQOPD	B16
* INFORD	B19
INHALT	A26
* INITAS	A33
INITRD	A3
INLSTL	A3
INLSTR	A3
INTGER	A27
IOPCPY	B17
IRALEX	B20
IRALST	A24
IRARDR	A12
ISYNTX	B17
ITSVAL	A12
LANORM	A27
LCNTR	A3
LDATVL	A22
LIST	A12
LISTAV	A15
LISTMT	A3
LNKL	A27
LNKLP	A27
LNKR	A27
LOCT	A23
LOFRDR	A4
LPNTR	A4
LPURGE	A22
LRDRCP	A12
LRDROV	A13
* LSPACE	A34
LSSCPY	A20
LSTEQL	A20
LSTMRK	A4
LSTPRO	A4
* LSTRCE	A34
LVLRVT	A4
LVLRV1	A5
MADATR	A5
MADLFT	A5
MADNBT	A5
MADNTP	A6
MADOV	A27
MADRGT	A6
MAKEDL	A15
MRKLSS	A15
MRKLST	A6
MTDLST	A13
MTLIST	A6
NAMEDL	A6
NAMTST	A23
NCODE	B16
NDAT	B16
NDIF	B20
NEWBOT	A13
NEWNOD	B16
NEWTOP	A13
NEWVAL	A21
NOATVL	A13
NPREC	B16
NUCELL	A25
NULSTL	A15
NULSTR	A16
NXTLFT	A16
NXTRGT	A16
PARMTN	A17
PARMT2	A17
POPBOT	A1
POPTOP	A1
* PREORD	B21
* PRESRV	A34
* PRLSTS	A34
* PRNTEX	B22
PUTFLG	A27
PUTHAF	A27
PUTMRK	A28

University of Cape Town

	routine	page		routine	page		routine	page
*	RCELL	A33		SEQSR	A1		SUBSBT	A19
	RDLSTA	A17	*	SETDIR	A28		SUBST	A2
	READEX	B26	*	SETD3	A28		SUBSTP	A19
	REALNO	A27	*	SETIND	A28	*	TERM	A32
	REED	A1	*	SETI3	A29		TOP	A2
*	RESTOR	A36		SHIN	A29	*	TRCOFF	A31
	SEQLL	A1		SQIN	A29	*	TRCON	A30
	SEQLR	A1		SQOUT	A30		VISIT	A32
	SEQRDR	A1		STRDIR	A30	*	WLKBCK	A31
	SEQSL	A1		STRIND	A30			

* These routines are subroutines rather than functions.

University of Cape Town