A BCPL TRANSLATOR USING RECURSIVE DESCENT TECHNIQUE

by

VASİL KADİFELİ

B.A. in B.A. Boğaziçi University, 1982

Submitted to the Institute for Graduate Studies in Science and Engineering in partial fulfillment of the requirements for the degree of

Master of Science

in

Computer Engineering

Boğaziçi University

1984

A BCPL TRANSLATOR USING

RECURSIVE DESCENT TECHNIQUE

APPROVED BY

Doç.Dr. Tunç Balman .....

Dr. Selahattin Kuru .....

Dr. Akif Eyler .....

DATE OF APPROVAL: 3/7/84

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my thesis supervisor, Doç.Dr. Tunç Balman for his continuous interest and guidance throughout the course of this study. Especially, his careful evaluation of this manuscript was of invaluable assistance in the preparation of this thesis.

Vasil Kadifeli

ABSTRACT

This thesis describes the implementation of a BCPL translator on a CDC CYBER 170/815 computer system using Recursive Descent Technique. The translator has been written in Pascal. The output of the translator is CDC COMPASS assembler instructions. A minimal run-time library of COMPASS routines have been prepared to provide for the interface of the BCPL language to the operating system.

ÖZET

Bu tez bir BCPL çeviricisinin, bir CDC CYBER 170/815 bilgisayar sisteminde, Özyineli İnme Tekniği ile uygulanışını anlatmaktadır. Çevirici Pascal programlama dilinde yazılmıştır. Çeviricinin çıktısı CDC COMPASS çevirici dili komutlarıdır. BCPL dilinin işletim sistemi ile arabirimini sağlamak için COMPASS yordamlarından oluşan bir geçiş süresi kütüphanesi hazırlanmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

I. INTRODUCTION

Compilers are a necessary part of any computer system. Without them programming would have to be done in assembly or even machine language. This has made compiler construction an important, practical area of research in computer science [1].*

In the past compiler writers and designers seemed to form an elite group within computing science, set apart of their esoteric knowledge and ability to produce large and important system programs which really worked. The admiration of the computing public, whether deserved or not, is no longer merited now that the principles of programming-language implementation are so well understood. Compiler writting is no longer a mistery [2].

Today, computing community is well served with texts, good, bad and indifferent, on the subjects of compiling and compilers. One of the most popular methods of implementing a compiler is that of Recursive Descent and many compilers, including ones for the languages Algol 60, Pascal, Algol 68R and BCPL, have been written using this technique. Thus, for many years compiler writers have used recursive descent as an informal method, grafting on parts of other methods and using different techniques when expedient to do so. However, there is very little in the literature about them. Perhaps this is because the method was thought too obvious or too simple. This is not the same as saying it is trivial, because the simple is the better. However surveying the literature available, the following landmarks stand out [3]

i. In 1968 Foster published a notable paper called 'A Syntax Improving Device' (SID) in which he showed how to manipulate grammatical constructs for languages, if at all possible, into forms suitable for what is known as the recursive descent

---

* Numbers enclosed in brackets refer to the references at the end

method.

ii. Later in the same year, Lewis and Stearns published a paper 'Syntax-directed Transductions' which used the term LL(k) for the type of grammatical restrictions placed on languages to allow their syntax to be scanned from left to right without backup, using a top down or recursive descent method. This placed the whole theory on a sound basis.

iii. It is clear that the early Burroughs compilers were recursive descent compilers even though they were not specifically given that description. Hoare has pointed out that the early Elliott Algol was also a recursive descent compiler.

iv. In 1971 Knuth published a tutorial guide to the grammatical aspects of LL parsing called 'Top Down Syntax Analysis'. This lucid account deals, as does the Lewis and Stearns' paper, mainly the syntactic aspects.

v. In the same year a group of working at the Royal Radar Establishment at Malvern used Foster's SID to improve the description of the language Algol 68R andautomatically generate a recursive descent compiler for it.

vi. It was 1973 before any significant paper was written about the practical aspects of compiling as a whole, using recursive descent, when Ammann published 'The Method of Structured Programming Applied to the Development of a Compiler'. This explains how the method is applied to writting a Pascal compiler and uses the techniques of program refinement to good effect.

This thesis describes the implementation of a BCPL translator on a CDC CYBER 170/815 computer system. The translator has been programmed in a high level programming language namely Pascal. It accepts programs written in BCPL language and transforms them into CDC COMPASS assembler programs that can be compiled by the COMPASS assembler and then executed.

The method adopted to program the translator is the recursive descent technique. A library of routines written in COMPASS assembly language has been prepared to provide the interface of the BCPL language

to the operating system, which in this case is NOS 2.1.

The chapter following this introduction describes what is a compiler, phases of a compiler, explains the recursive descent compiling and finally describes the syntax of the BCPL language that is the subject of this thesis.

Chapter III describes the lexical analysis phase of the translator.

Chapter IV describes the tables used and maintained by the translator and explains routines that manipulate these tables.

Chapter V describes the syntactic analysis phase and how code is generated during that phase.

Chapter VI describes the error recovery techniques adopted to respond for erroneous constructs of the BCPL source program.

Chapter VII states the run-time organization of the code generated by the translator. It explains the run-time stacksand the way they are manipulated. It also describes how the operating registers of the machine are utilized at run-time.

Chapter VIII describes the run-time library routines in detail and how they are used to provide for the interface of the BCPL program with the operating system.

Chapter IX states some of the BCPL programs used to test and prove the correctness of the code generated by the translator.

Chapter X has been devoted for discussion and conclusion upon this study.

Finally nine appendices have been added to present details.

II. COMPILERS AND RECURSIVE DESCENT COMPILING

A compiler is a computer program that translates another program called the 'source program' into a third one, the 'object program'. The source program is written in the source language and solves a particular problem for a user. The object program produced solves the same problem but is expressed in the object language. In general the source language is one which users find easy and a natural tool to solve their problems. The object language on the other hand, is a natural one for some machine to execute. Thus a compiler can be viewed as a tool which transformsprograms from the user's domain of problem solving into the machine's domain of problem execution, without varying the meanings of the programs.

Though the object code usually is directly executable, a translator usually transforms a symbolic program of the user into another symbolic program which can easily be transformed into an executable one. In this thesis the source program is written in BCPL and the object program is COMPASS assembler code. The translator on the other hand has been written in Pascal.

The compiler must analyse the source program and synthesize the object program. In fact nearly all compilers perform the analysis in two distinct phases called 'lexical' and 'syntactic' analysis. Any compiler has also a third important phase, that of 'code generation'.

2.1 Lexical Analysis Phase

This phase consists of an analysis of the microsyntax of the source program. This is the processing of a string of characters, transforming them into a string of 'basic symbols'. These include keywords (or reserve

words) such as 'if','let' and 'writeln', single symbol punctuation marks such as '(' and ';' , operator symbols such as '-' and '*', multiple symbols of the two above kinds such as '//' and ':=' , assembly of literals such as '125' and 'true', and finally the collecting together of the characters in identifiers such as 'x' and 'temperature'.

2.2 Syntax Analysis Phase

This is the second phase that has as its input the string of basic symbols produced by the lexical analysis. The output on the other hand is a 'parse tree' or 'syntax tree' which is an internal form of the program in a structure which allows subsequent phases to see the relationship of the parts of the program to each other and to the whole program.

2.3 Code Generation Phase

This phase is synthetic rather than analytic. It takes the parse tree that is the output of syntax analysis, traverses it and produces object code in at least a preliminary form.

2.4 Passes of a Compiler

One decission the compiler writer has to take is how to organize the phases into passes. A multipass compiler makes complete scans over the various forms the program goes through, both internal and external. Each pass reads the output from the previous one (or the source program if it is the first pass) and produces complete output for the next pass. No pass is invoked until the previous one is complete. For example if we were to organize the lexical analysis phase as a pass the compiler would first completely scan the source and produce a file of basic symbols. Space whether in main store or in backing store, must be found for this intermediate form of the program. The next pass which will include at least the syntax analyser will then read this file and produce its own output file.

However, in some compilers all the phases can be gathered together into one pass, and instead of storing complete files of intermediate data, the phases call each other as subroutines to ask for or provide information one piece at a time. Thus the syntax analyser may call the lexical analyser and ask it for the next basic symbol. It may also call the code generator to emit the next piece of code. Recursive descent is such a type of compiling method.

The organization of phases into passes may depend on the language being compiled. Some languages actually require several passes. For instance, if an object in a program can be used before it is declared then code cannot possibly be generated for the use of the object without having complete knowledge of its properties. In such cases a complete pass will have to be made to gather such knowledge and another to generate code based on that knowledge.

2.5 Recursive Descent Compiling

This method centers around the syntax analysis phase of the compiler which is divided up into a number of recognition routines, each of which has the task of checking whether a particular kind of phrase is present in the input. Each recognition procedure can call upon the services of other ones to recognize the appearance of subphrases and so on. Most of these routines will be mutually recursive reflecting the fact that within one sequence we can find others embedded at a lower level. In the same way, expressions can contain subexpressions, declarations include inner declarations and so on. Each of these has its own recognizer which is invoked from above when appropriate.

Some recognizers will have some choices to make. When such choices are to be made, decisions are always taken by looking at the input stream for the next basic symbol. However the task of a compiler is not merely to recognize correct programs; it must also produce object code. Therefore each recognizer has to be modified or refined in order to emit code.

Something important to notice here is that the syntax tree referred to as the output of the syntax analysis phase is never explicitely grown. This is because the syntax analysis phase and the code generation phase

are not separated into distinct passes, but rather integrated into one another in order to understand clearly what each recognizer-emitter does. The tree is implicit in the dynamic calling structure of the recognition routines and is traversed by the code generation phase as it is built, and branches no longer used are destroyed as the routines are exited.

Other refinements have to be introduced and these are based on error recovery and type checking. The type handling part of a recognizer must also be able to pass type information back to its parent recognizer.

2.6 LL(k) Grammars

To make the parsing phase ie. the tree building part more efficient, it may well be interleaved with semantic actions based on the structure of the tree itself. For instance, code may be generated and entries may be made in highly structured symbol table. However, these actions may have to be undone, and in some cases may even have to be done again if we consider a construct of the following form :

C ::= if B then C else C / if B do C

being parsed and if further the parser tries the 'if .. then .. else' production to start with, it will fail when do and then cause a mismatch. It will then have to backup. The boolean expression B may be a complicated one and the syntax of the language may allow to contain a whole block together with local declarations, possibly including those of procedures.

This trouble of back-tracking shows itself at places where a parser has to choose between several alternative productions with the same left hand side. The only information available to help it make the right choice is the input sequence encountered up to that time. Any method therefore that looks ahead in this way, will only work well if we provide some sort of buffer internal to the compiler where input can be examined. Conceptually this buffer represents the first few terminals of the input, the rest of which is still in the outside world. In practice, it is an advantage for efficiency and ease of access that this buffer be of fixed

length. A top-down parser, if given a lookahead buffer capable of holding k terminals, is called an LL(k) parser, where LL stands for 'Left to right scanning using a Leftmost derivation'. In practice k is usually one.

2.6.1 LL(1) Grammars and Recursive Descent Compiling

One of main features of the recursive descent method when used practically is that it must be able to do its recognition, type checking and code emission without backup; that is, if a recognition routine A decides to call another, B, it can be sure from the first that barring errors on the user's part, it has made the correct choice based on the input it has before it. This limits the kind of a language which can be compiled by that method. Such a language must be of LL(1) type.

Davie and Morrison [3] give a number of rules for transforming grammars which may help to make them LL(1) parsable. However, they still point out that these rules are not foolproof and that other methods, based on common sense may need to be applied.

2.7 The BCPL Language

BCPL (Basic CPL) is a high level language, readable and easy to learn and provides facilities of the assembler language like address manipulation, indirection, left and right shift of bit patterns etc. It is actually a typless language in the sense that all data are viewed as bit patterns. The treatment of data is unusual and allows the power of a language with dynamically varying types. BCPL is related to CPL or Combined Programming Language. It was originally designed as a tool for compiler writting, therefore it is suitable for large nonnumerical problems and has been used for writting systems programs [4].

The implemented syntax closely resembles to that defined by M.Richards [4]. It has been modified however, to make it applicable for recursive descent. For example, parts of its syntax that are highly recursive like the following construct have been eliminated

E ::= ... / valof C / ...

where E stands for expression and C stands for any valid command like test E then C or C. So a statement of the following type is not allowed

```
<name>:=valof [ test <expr> then <command>
                           or   <command> ;
              resultis <expr>]
```

Also the three repeat commands

C repeat
C repeatwhile E
C repeatuntil E

have been unified into the form : repeat C until E. The declaration type global has been removed since the external feature of BCPL has not been implemented. The complete syntax of the language that is the object of this thesis can be found in Appendix A.

A new feature that can be considered quite powerful has been introduced. This is the keyword compass that provides for immediate insertion of assembler code into the code generated during the translation process. This feature enables the user to utilize facilities of the assembly language that BCPL does not support. Such a facility was possible to implement due to the fact that the translator is also producing a stream of assembler instructions.

2.8 Structure of the BCPL translator

The BCPL translator of this thesis has a structure that can be studied mainly in three parts: lexical analysis part, syntactic and semantic analysis part, and the code generation part. Figure 2.1 illustrates this structure together with the files the translator is manipulating.

The lexical analysis part is composed of a single routine, the scanner, that reads the BCPL source program, provides a listing of it, and splits that program into its basic symbols or tokens.

The syntactic and semantic part of the translator, which is actually a chain of recursive routines, provides for the recognition of BCPL type constructs. It makes syntactic and semantic checks upon them to prove the correctness of the source program. It also reports compile/translation time errors in case of constructs with incorrect syntax or semantics. The scanner is invoked to provide the next token from the source program, inorder to make correct choices in choosing a certain branch within the syntax of the BCPL language.

The code generation part is invoked by the syntactic and semantic routines to provide the appropriate assembler instructions corresponding to the semantics of the BCPL source program. This part is composed of two routines, 'gencode' and 'enterlib'. The first of them is the one invoked by routines of the previous part and generates specific assembler instructions. The second routine is called at the end of the translation process in order to read the BCPL library file (BCPLLIB) and append the required routines to the end of the file where the assembler code, routine gencode has produced, resides.

The translator and the BCPL library routines file have been placed to system's library, and the user can compile and execute a BCPL program by using the following operating system commands :

1. GET,BCPL,BCPLLIB/UN=LIBRARY
2. ATTACH,RELOPL/UN=INS,PN=PACK1
3. BCPL,sourcef,listf,codef
4. REWIND,codef
5. COMPASS,I=codef,B=binf
6. binf

The first two commands make the translator (BCPL) and the library file (BCPLLIB) and the system library file (RELOPL) local to the job so they can be used. The third command invokes the translator to read the BCPL

program that resides in file 'sourcef', place the listing of the program together with its translation time errors into file 'listf' (if not specified file output is assumed) and write the assembler code generated plus the required routines from file BCPLLIB into file 'codef' (if not specified file 'code' is assumed). The next two commands provide for the assembly of the translated program, and the resulting executable binary program is placed into file 'binf' or by default to file 'lgo'. The last command calls that file to be loaded and executed.

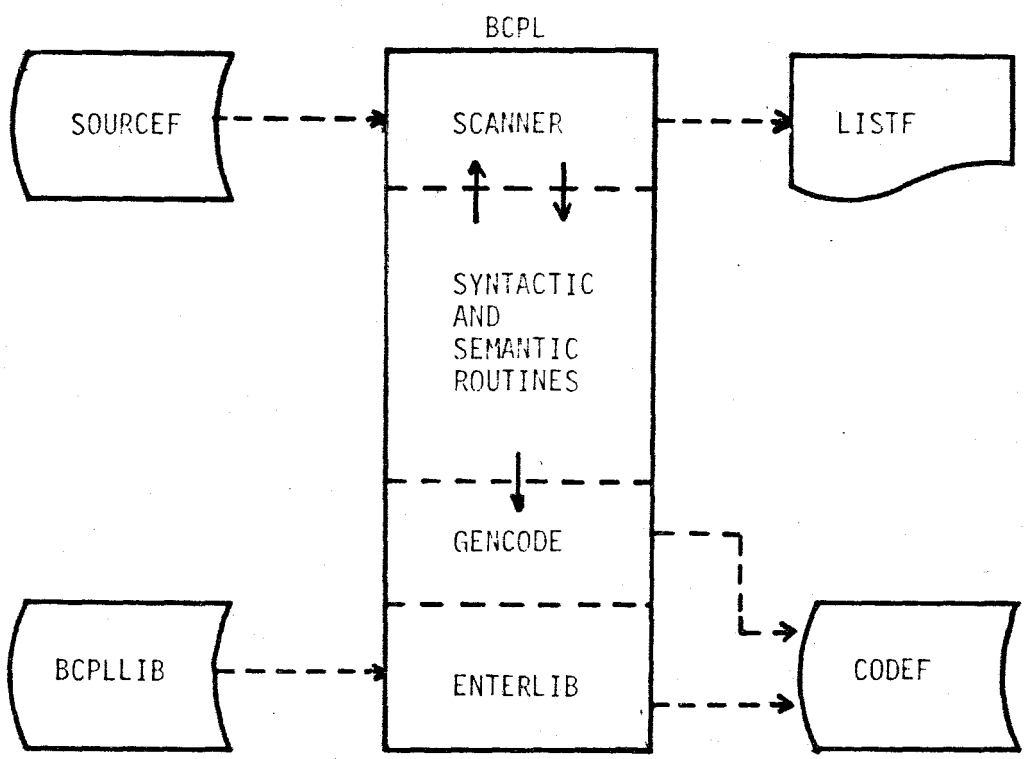


Figure 2.1 The BCPL translator and files it manipulates

III. LEXICAL ANALYSIS

The lexical analyser functions as an interface between the source program being compiled and the routines that check and interpret its syntax. This module of the compiler transforms each source statement into a form that is easier to manage and recognize. The lexical analyser takes its input in character code from the source program, detects its syntactic primitives of the language and converts them into concise fixed length items known as the tokens of the language [6]. The output produced by the lexical analyser is these tokens that are consumed by the syntactic and semantic routines or the parser.

There are several reasons for the conceptual separation of the lexical analyser from the other components of a compiler [6]. Its distinct and specialised algorithms clearly distinguish it as a module in its own right. The tokens passed onto the parser can be precisely defined to establish it as a subprogram that can be implemented and debugged on its own. This modular approach simplifies the design process and facilitates quicker debugging.

3.1 The Scanner

During lexical analysis the input stream that is the BCPL source program is scanned character by character to separate its tokens. This task is done by the 'scanner' which is a routine organized as a state diagram. Besides this task, the scanner has also other things to do.

First of all, a line of the input stream is read into a buffer and when the scanning of that line is finished, the line is printed to the output file inorder to provide a listing of the program that is being translated. Secondly, if any errors have been detected by the scanner or the syntactic and semantic routines, the scanner prints a

marker line below the translated line, in which positions of errors reported have been marked. As a third task, the scanner recognizes translator directives and takes the appropriate action. Translator directives control the listing of the program that is being traslated and may also affect the assembler code produced. Finally, if the token recognized is a constant, the scanner invokes a routine called 'linkconst' to enter that constant into the constants table. On the other hand if it is a name rather than a constant, a routine called 'reserved' is invoked to determine whether this name is a reserved keyword of the language or an identifier.

The input stream is viewed as 80 column card images and scanning continues until the end of file or the reserved keyword finish is encountered. A function called 'nextchar' is used to obtain the next character from the input buffer where the current line resides.

When a token is recognized, detailed information is placed to a global record called 'token', which has basically two fields :

class : this field is of user defined type 'symbol' which comprises user defined scalars to describe any token of the language such as 'id','colon', 'thensy' etc.

op/cstaddr : only one of these fields exists at a time and their existance depends on the class of the token. If the token is one of : relop, addop, mulop, addrop, shfop, readsy, writesy then field op is existing. If the token is a constant, that is one of : number, string then cstaddr is existing. In the first case op contains a more detailed description of the token while in the second cstaddr is a pointer to the relevant entry in the constants table. Table 3.1 illustrates the usage of this field.

CLASS	OP	CSTADDR
relop	eqop (=)	
	geop (>=)	
	gtop (>)	
	leop (<=)	
	ltop (<)	
	neop (<>)	
addop	plus (-)	
	minus (-)	
mulop	astr (*)	
	rdiv (/)	
	rem (rem)	
addrop	lvop (lv)	
	rvop (rv)	
shfop	lshf (lshift)	
	rshf (rshift)	
readsy/writesy	c (readch)	
	l (readln/writeln)	
	n (writen)	
	o (writeo)	
	s (writestr)	
number		pointer to constants table
string		pointer to constants table

Table 3.1 Utilization of op and cstaddr fields according to class field

As mentioned above the scanner operates according to a state table and the diagram in figure 3.1 represents this state table. Some explanations on the entries of that diagram follows :

ch : represents the starting character of the scanning process
ø : represents the cheracter blank
l : represents the set of characters 'A','B',...,'Z'
d : represents the set of characters '0','1',...,'9'
od : represents the set of characters '0','1',...,'7'

The resulting token types are within parentheses in the form (class) or (class,op).

A fact to be mentioned here is that the operation of the scanner depends upon the needs of the syntactic and semantic routines. That is, whenever these routines require the next token of the program being translated, they invoke the scanner to supply it.

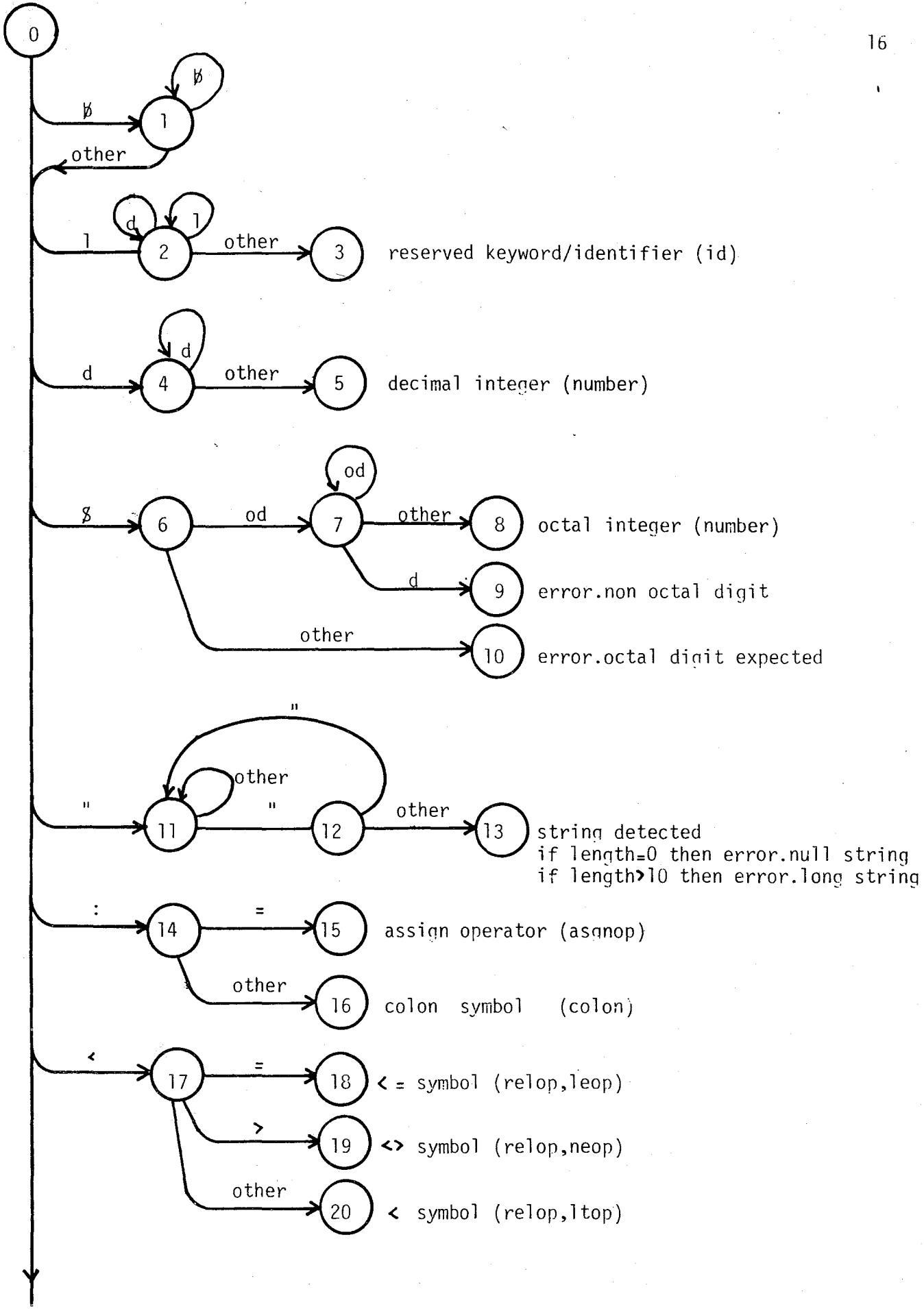


Figure 3.1 State diagram of the scanning process (continued on next page)

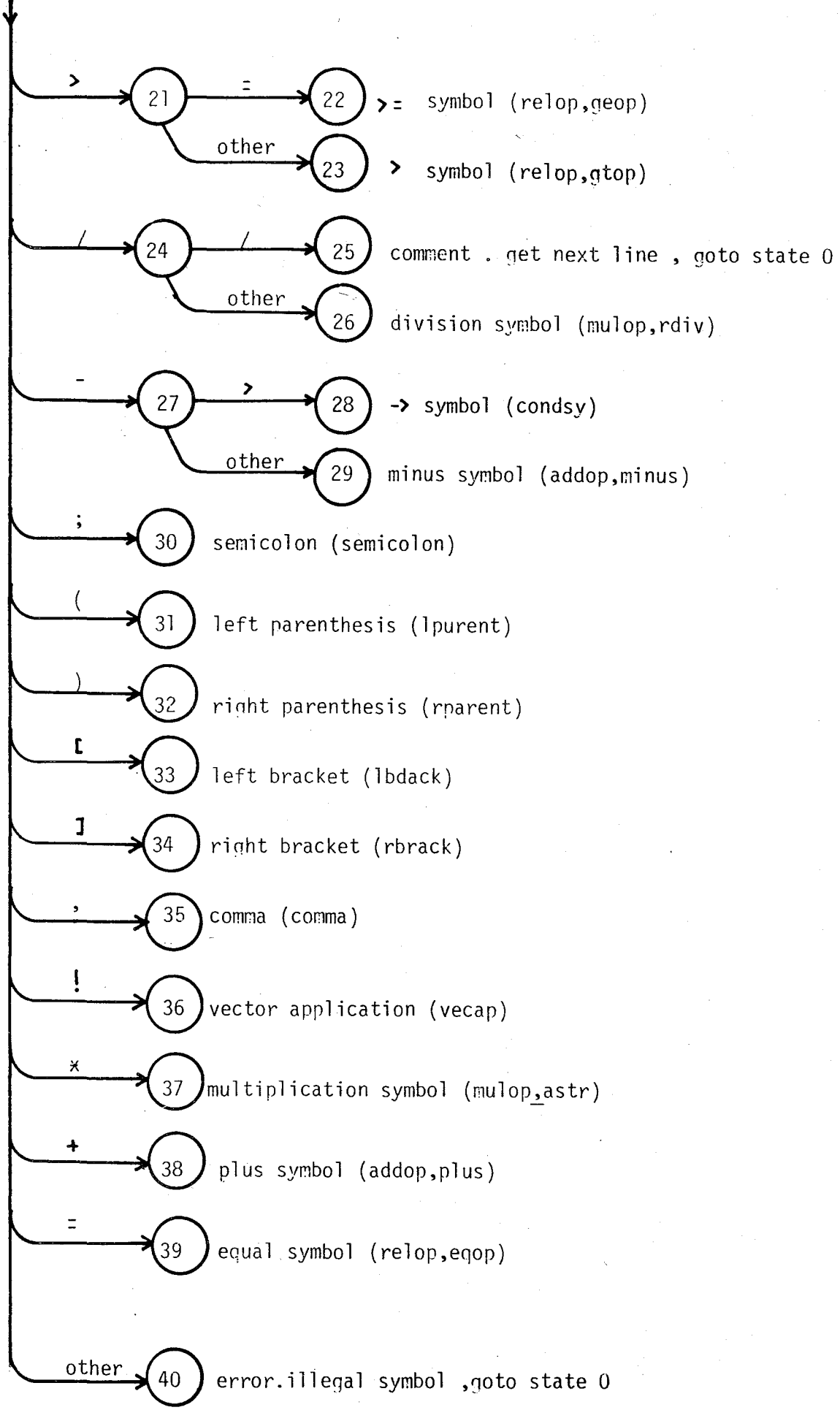


Figure 3.1 (continued) State diagram of the scanning process

IV. TABLES USED AND MAINTAINED BY THE TRANSLATOR

During the translation phase the translator uses and maintains mainly three tables. One of them,the keywords table is a static one and is initialized before the translation process starts. The other two, namely the constants table and the symbol table, are built up as the translation process proceeds. However, while the size of the constants table is constantly increasing, the size of the symbol table is both going up and down. Detailed description of these three tables follows.

4.1 The Keywords Table

The keywords table is straightforward one that contains all the reserved keywords of the BCPL language plus the names of the non-standard input/output functions of this implementation. Each entry includes a ten character string that contains the name of the entry, like 'let', 'writeln' etc., and two other fields one for the class of the keyword and one for the operation. For example the entry for the non-standard output function 'writeln' contains the string 'WRITELN ' in its name field, the scalar constant 'writesy' that means write symbol in its class field that is a member of the global set 'symbol', and another scalar constant 'l' that stands for line in its op field which is also a member of another global set, namely 'operator'.

As mentioned above the table is initialized in alphabetical order of keywords before the translation process starts. This table is used by a routine called 'reserved' that contributes to the lexical analysis phase and is invoked by the scanner after a name has been recognized. The name is passed to that routine as a string where binary search

is used to find a matching entry for that string. Since the table contains less than 64 entries the number of accesses to the table in the worst case is not more than six, that is $\log_2 64$.

If a successful result appears after the search, then this is a keyword and the fields of the global record 'token' are initialized by those of the entry. Otherwise this is absolutely an identifier and 'token' is initialized accordingly.

4.2 The Constants Table

The constants table is actually built during the lexical analysis phase and utilized during code generation. However a single entry, that is the value 'true' of the BCPL language which is a bit string of all ones, is made to that table as the first one during initialization stage. This is done to simplify things at run-time because such a value is recognized by the system as a minus zero, and refering to that table is a simple way of regenerating the value 'true' at run-time whenever required.

The table has been organized as a linked list, where entries are made dynamically at its end. A global variable CTSTART points to the first entry and another one CTLAST points to the last entry of the table at translation time.

Any costant recognized by the scanner is passed to a routine called 'linkconst' where a sequential search is made through the list to find a previous entry of the same constant. If found a pointer to that entry is returned, otherwise a new entry is made to the endof the table and a pointer to it is returned. This provides a more optimized size for that table at translation time. The same is also true for run-time because this table is dumped to the end of the code file in order to be used at run-time.

4.3 The Symbol Table

The symbol table has been organized as a stack-type linked-list of unbalanced binary trees where each tree is representing the data segment

of a certain BCPL block (either main program or procedure or function) and the last one belongs to the block that is being translated or executed at run-time. This type of organization represents the nesting level of procedures. The tree of a block that has already been translated is not present at the symbol table and this contributes to a more optimized size for that table.

The root of each tree is a header node containing information about the block it belongs to. The most important of them are : the name of the block, the number of its parameters plus the number of its local variables, a link to its first parameter, the type of the block that is procedure or function, and a pointer to the next tree if exists.

The header node for the main program's tree is created at initialization time and two pointers are used in the manipulation of the symbol table at translation time. STSTART points to the header node of the tree that belongs to the main program and CLEVPTR that stands for 'current level pointer' points to the header node of the last tree which belongs to the block being translated.

Whenever the translator finishes with the translation of a BCPL block (that always corresponds to the last tree) the whole tree except the root node is disposed and the route or header node is inserted at the appropriate place in the previous tree for future references that are calls made to that block. Figure 4.1 illustrates this process.

4.4 Symbol Table Manipulating Routines

The symbol table is manipulated mainly by six routines which are described in following sections.

4.4.1 Routine ENTERID

This is a routine to enter a new identifier into the symbol table. An identifier is either a variable, a label or a routine name either procedure or function. Two different types of entries are made and this is indicated to routine enterid by an argument of the following type :

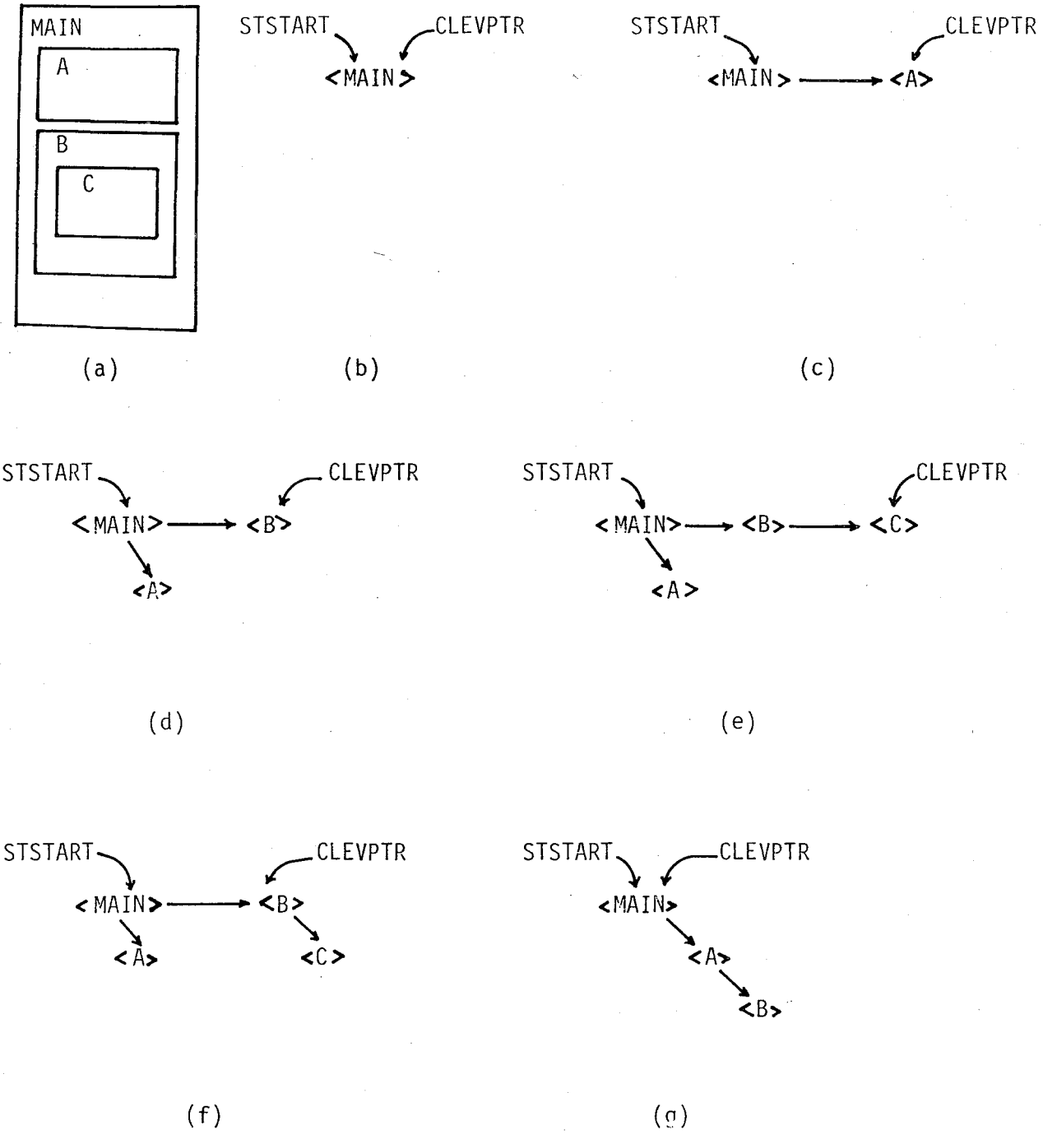


Figure 4.1 Manipulation of Symbol Table

(a) Structure of program to be translated

(b) Start of translation

(c) Block A is encountered

(d) Block A finishes, block B is encountered

(e) Block C is encountered

(f) Block C finishes

(g) Block B finishes

enterlevel = (eclev , enlev)

If the name to be entered is a routine name, then the scalar constant 'enlev' is passed as an argument which actually means that the name has to be entered as the header node of a new tree level. Thus routine enterid creates a new header node where the name of the routine is placed, and this node is linked to the header node of the previous tree that was pointed to by CLEVPTR. Contents of CLEVPTR are also changed to point to this new tree root.

On the other hand if the name to be introduced is a variable or a label name the scalar constant 'eclev' is passed to indicate that the entry is going to be made at the appropriate place into the last tree ie. the current tree pointed to by CLEVPTR. This is the case where the definition of a new variable, or vector, or table, or label, or the reference of a new label in a goto command has been encountered within a block that is being translated.

In either case a pointer to the new entry is returned into the global identifier 'idptr'.

4.4.2 Routine SEARCHID

This is a routine to search the symbol table for a specific name. Since the symbol table organization is in the form of binary trees the number of seeks is minimized.

Again two types of searche can be made depending on an argument of type :

searchlevel = (salev , sclev)

If the name to be searched is a label name than only the current level or tree is searched, and this is notified by the scalar constant 'sclev' passed as an argument. This is the case in this implementation because goto commands have been limited to reference only local labels. The returned value is a pointer to the entry found if search was successful otherwise nil is returned.

On the other hand if the name to be searched is not a label, then search continues at all trees going backwards from the last one to the first one until a matching entry is found or the whole symbol table is exhausted. That type of search is specified by the scalar 'salev' which means search all levels. In this case three different things are returned. First the pointer to the entry found, second the number of levels searched, and third an indication of whether that node is a top node ie. a header node or not. The latter indicates whether the name was defined as a routine name or not.

Switching from a tree to a previous tree while searching is accomplished by calling routine 'prevlevptr' which is explained later.

4.4.3 Routine FREENODE

This routine releases a certain node from the symbol tabel. This node is always a leaf node of a certain tree. More details are given in the next section where routine 'freelevel' is explained.

4.4.4 Routine FREELEVEL

This routine is used to release a complete tree (always the last one) after the translation of a BCPL block has been completed. The tree is traversed in postorder, that is :

```
traverse left subtree
traverse right subtree
process node
```

and whenever a leaf node is reached, routine freenode is called to release that node. However before releasing a certain node a check is made to see whether it contains a label marked as undefined. If that is the case an informative error is issued and the corresponding label is inserted into the code file at the end of the code generated for that block. This is done to achieve some recovery and maintain the correctness of the code generated.

At the end of the traversal process when time comes to process the header or root node, it is not released but entered at the appropriate place into the previous tree. Contents of CLEVPTR are also modified to point to the previous tree header.

4.4.5 Routine PREVLEVPTR

This routine accepts an argument that is a pointer to the header node of a tree within the symbol table and returns the pointer to the header node of the preceeding tree if any exists, otherwise nil is returned.

4.4.6 Routine BUILDPARAM

This routine links together the formal parameters of a certain BCPL block in the order they have been encountered. Figure 4.2 illustrates a resulting tree of a BCPL procedure after its definition constracts have been processed.

```
let  PROC(c,a,g,b) be
  [ static  [i:0 ; x:0 ; y:0] ;
        .
        .
        .
        .
        .
        .  ] ;  // end procedure PROC
```

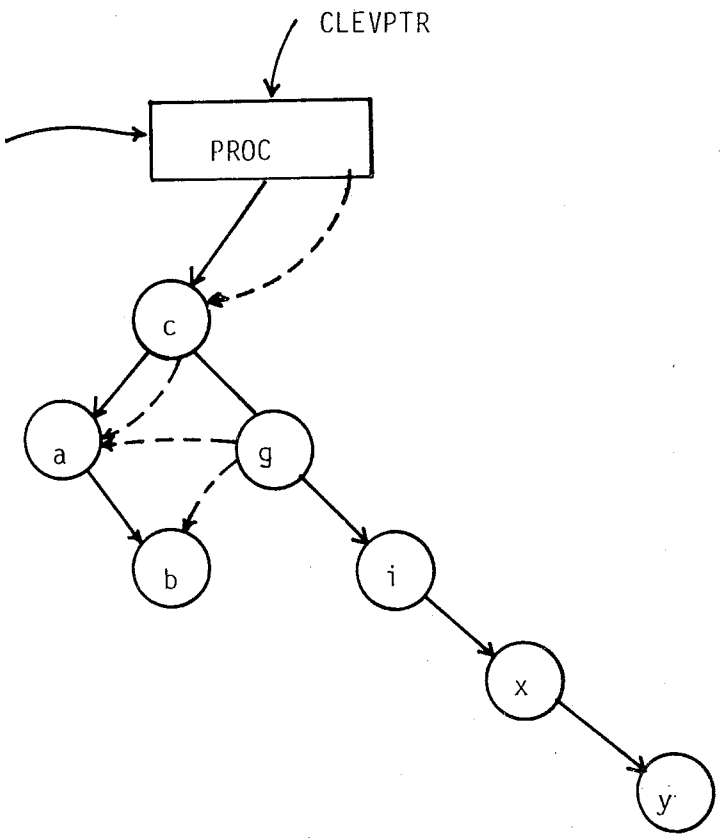


Figure 4.2 A BCPL procedure and its corresponding tree into the symbol table (threaded lines are links among formal parameters).

V. SYNTACTIC ANALYSIS AND CODE GENERATION

The syntax analysis forms the heart of recursive descent technique. Having the grammar of the language that fits for one-pass compilation, the syntax analysis layer can be written down from a formal specification of the language in BNF. Appendix A gives the syntax of the BCPL language implemented,in BNF notation.

As mentioned in Chapter II the syntax analyser is made up of a set of mutually recursive procedures, one for each non-terminal symbol in the grammar. The syntax analyser deals only with the basic symbols of the language and not the individual characters that make up the symbol. Since the syntax anlyser always asks for the next basic symbol in the input stream a procedure called 'scanner' and described in Chapter III has to be provided. Thus terminal symbols are compiled by the scanner.

Two other procedures, described in Chapter VI are used to make the job simpler. These are procedure 'mustbe' and function 'have'. Procedure 'mustbe' is useful in cases where there is no option in the choice of symbols. It checks the current basic symbol and advences to the next one if check is correct. Function 'have' is used when we have more than one choice to proceed. It also advances to the next symbol if the current symbol is the one expected.

A global record called token keeps the type of the basic symbol, that the scanner has recognized, and its value is used by the syntax analyser.

5.1 BNF and Coding

The syntax of a language can be defined by two separate rules and the analyser has to be written according to them [3]. The two sets of rules are :

i. grammar written in BNF
ii. type matching rules

The type matching rules are utilized while parsing an expression. However since BCPL is a typless language only the first set of rules is required to write the BCPL syntax analyser.

Since the anlyser is going to be written from the BNF of the language, productions of BNF have to be translated into the language that the syntax analyser is going to be written. Productions of the form :

```
<A> ::= b <B>
```

translate directly into the code :

```
mustbe('b') ; B
```

A production of the form :

```
< A> ::= b < B> /  c < C >
```

therefore translates into :

```
if have('b')
   then B
   else begin
          mustbe('c') ;
          C
        end
```

And in more general a production of the form :

```
< A> ::= b < B> /  c < C> /  d < D> /  e < E>
```

translates into :

```
case token of
  'b' : begin
          scanner ; B
        end ;
  'c' : begin
          scanner ; C
        end ;
  'd' : begin
          scanner ; D
        end ;
  otherwise begin
              mustbe('e') ; E
            end ;
end (* case *)
```

The repetition metasymbol '*' can also be coded. For example, the production :

$$\langle A \rangle ::= \langle B \rangle \{ b \langle B \rangle \}^{*}$$

translates into :

```
B ;
while have('b') do B
```

A more difficult situation to code is where each option in production starts with a non-terminal symbol. In this case all but one of the non-terminals are followed until all the legal starting symbols are found. For example

$$\langle A \rangle ::= \langle B \rangle \ / \ \langle C \rangle$$
$$\langle B \rangle ::= \ d \ \langle D \rangle \ / \ \ e \ \langle E \rangle$$

translates into :

```
case token of
  'd' : begin
          scanner ; D
        end ;
  'e' : begin
          scanner ; E
        end ;
  otherwise C ;
end (* case *)
```

This eliminates some of the procedures, here procedure B , from the the syntax analyser. However care should be taken especially when the productions removed are used elsewhere in the grammar. Therefore another solution can be found :

```
case token of
  'd','e' : B ;
  otherwise C ;
end (* case *)
```

And procedure B has the following form :

```
if have('d')
   then D
   else begin
          scanner ; E
        end
```

5.2 The Syntax Analyser

In the translator the procedure that parses a non-terminal has a name corresponding to the name of the production in the BNF form of the language. The main program of the translator has the following form :

```
initialize ;
scanner ;
block ;
```

where 'initialize' is a procedure to provide for initialization tasks of the translation process, 'scanner' is called to obtain the first basic symbol of the BCPL source program and 'block' is the procedure that will start the translation of the BCPL source program. This production has the following form :

< program > ::= < block >

procedure block in its turn calls repeatedly procedure 'declaration' to parse declarations and then procedure 'command' to parse commands until the symbol finish is encountered. The BNF form of block is the following :

< block > ::= $\{$<declaration>;$\}^*$ $\{$<command>;$\}^*$ finish

Procedure command requires careful study since it is the starting point for the translation of any BCPL command. The production command is a lengthy one and can be found in Appendix A. The structure of procedure command is a Pascal case statement that recognizes the first terminal symbol of a command type and then calls the appropriate routine to parse that command. The following code is a simplified version of procedure command :

```
case token of
  testsy : begin scanner ; testproc end ;
  whilesy : begin scanner ; whileproc end ;
  ifsy   : begin scanner ; ifproc end ;
  repeatsy : begin scanner ; repeatproc end ;
  gotosy : begin scanner ; mustbe(id) end ;
    .
    .
    .
```

```
id      : begin
            scanner ;
            if have(colon)
               then command
               else begin
                      mustbe(asgnop) ;
                      expr ;
                    end ;
          end ;
  otherwise error('illegal symbol') ;
end (* case *)
```

In the above program when the terminal symbol that preceeds a command is encountered it is thrown away by calling procedure scanner and then the appropriate routine is called to parse that certain command. However in some cases like the one that the first terminal is 'id', the next symbol has to be examined in order to identify the type of the command and then call the related routine to parse it.

The structure of any procedure called corresponds to the related production in the BNF form. For example procedure 'testproc' that parses a test command has the following structure :

```
expr ;
mustbe(thensy) ;
command ;
mustbe(orsy) ;
command ;
```

First a call is made to procedure 'expr' that parses any BCPL expression in order to parse the boolean expression after the test symbol. Then a check is made for the symbol then using procedure mustbe since this is the only symbol that can follow the boolean expression. After the check procedure command is called to parse any BCPL command that follows the symbol then. This is the recursivenature of this compiling technique.

Another important part of the syntax anlyser is the set of routines that parse expressions. These are written in a structure that corresponds to their BNF form. The structure of the first procedure of them whose BNF form is :

<expr> ::= <expr0>{-> <expr> , <expr>}

is given here for reference :

```
expr0 ;
if have(condsy)
   then begin
          expr ;
          mustbe(comma) ;
          expr ;
        end;
```

All the way through the scanning process, routines that parse declarations call routine 'enterid' to enter any name encountered, into the symbol table. Routines that parse commands and expressions on the other hand, call routine 'searchid' to check whether a name encountered exists within the symbol table. Otherwise they take corrective actions.

5.3 Code Generation

Any routine of the syntax analyser, after recognizing a certain construct, generates the appropriate code while parsing that construct. The code is generated by making different calls to procedure 'gencode' that writes the appropriate assembler instructions into file 'code'. Here again procedure 'testproc' will be considered to show how code is generated :

```
expr ;
mustbe(thensy) ;
gencode('      MI   X1,LAB1 ') ;
gencode('      ZR   X1,LAB2 ') ;
gencode('      JP   LAB3    ') ;
gencode('LAB1  ZR   X1,LAB4 ') ;
gencode('      JP   LAB3    ') ;
gencode('LAB4  NO           ') ;
command ;
gencode('      JP   LAB3    ') ;
mustbe(orsy) ;
gencode('LAB2  NO           ') ;
command ;
gencode('LAB3  NO           ') ;
```

Thus the code generated from this procedure is the following :

```
        .
     expr
        .
      MI   X1,LAB1
      ZR   X1,LAB2
      JP   LAB3
LAB1  ZR   X1,LAB4
      JP   LAB3
LAB4  NO
        .
     command 1
        .
      JP   LAB3
LAB2  NO
        .
     command 2
        .
LAB3  NO
```

Code for the boolean expression and the two commands are generated within the related routines. The result of any expression is placed within register X1. Therefore register X1 is tested at the beginning to see if it is a minus zero ie. a bit pattern of all ones, which is the value true. Then control goes to label LAB4 where the first command is executed and then a jump is performed to the end of the code that is LAB3. If register X1 is a plus zero which is the value false control goes to label LAB2 where the second command is executed. Otherwise if the value in register X1 is neither a minus zero nor a plus zero control goes to label LAB3, so that neither of the two commands are executed.

VI. ERROR DIAGNOSIS AND RECOVERY

It is an unfortunate fact that the user will always submit programs that are syntactically incorrect. The translator must react to such programs and furthermore must be able to deal with any number of possible kinds of errors.

Fortunately LL grammars and especially LL(1) grammars have good error detection properties [3]. The program is scanned top to bottom left to right. For every input symbol the translator has a goal symbol which may be one of a number of symbols. If the input is not one of the goal symbols an error has been detected. Therefore in an LL(1) grammar where we look only one symbol ahead, we can discover whether or not the input symbol is legal before moving on.

The question is what attempt should the translator make to remedy the situation. A first solution that comes to mind is to give up. However in practice it is usually never acceptablesince it may make the debugging of a large program a very trying exercise. Therefore we are forced to consider errorrecovery.

The underlying philosophy of error recovery is to change an incorrect program to the nearest syntactically correct one. However a compromise has to be done, and design a simple scheme that is low in cost in terms of time and code and will still recover from the majority of errors.

The most common types of errors are :

i. The insertion of extra symbols
ii. The omission of symbols
iii. The replacement of symbols
iv. the transposition of two symbols

In this implementation two techniques of error recovery schemes have been incorporated. Both schemes are relatively simple and can be coded inexpensively. The first scheme is that of Ammann's [3] he used in his Pascal compiler and the second was invented by Turner [6] for use in a compiler for the applicative language SASL. Both of these compilers were LL(1) and both were implementedusing recursive descent technique.

6.1 Ammann's Error Recovery Sheme [3]

The basis of the scheme is to divide the legal symbols of the language into two disjoint sets, the set of relevant and the set of irrelevant symbols.

Every procedure that parses a non-terminal in the grammar is given as parameter the set of relevant symbols. If an error occurs, depending on the recovery strategy, the parsing procedure may skip all the input symbols up to the first relevant one. The parsing procedure also passes on the set of relevant symbols to any other parsing procedure that it calls. For example to translate the while command that has the following syntax :

while <expr> do <command>

the relevant set of symbols is passed to procedure 'whileproc' that will translate that command. Before whileproc calls procedure 'expr' to translate the boolean expression following the keyword while, it adds the symbol 'do' to the set of relevant symbols and passes it to procedure expr. If an error is detected at that procedure, the error recovery will not skip past the do symbol. After control comes back procedure whileproc resumes normal service from that point as nothing has happend.

The worst case would be when this recovery scheme recovers at the wrong point. For example when procedure expr finds another symbol while skipping which is from the relevant set but that is not a do.

Two procedures support the recovery scheme of Ammann in the translator. These are procedure 'error' and procedure 'skip'. The first one issues an appropriate error message and the second one skips to the first relevant

symbol. Following is a simplified version of procedure skip :

```
procedure skip(relevantset) ;
begin
  while not (token in relevantset) do scanner ;
end ; (* skip *)
```

6.2 Turner's Error Recovery Scheme [7]

Turner's recovery scheme depends on the use of three primitives, namely 'have','check' that has been implemented with name 'mustbe', and 'complain' that has been implemented as 'report'.

These primitives have been implemented as routines in the following structure :

```
procedure mustbe(symbol) ;
begin
  if recovering
     then begin
            while token <> symbol do scanner ;
            scanner ;
            recovering := false ;
          end
     else if token = symbol
            then scanner
            else report(symbol) ;
end ; (* mustbe *)

procedure report(symbol) ;
begin
  if not recovering
     then begin
            writeln('error',token,' found where ',symbol,'expected') ;
            recovering := true ;
          end ;
end ; (* report *)
```

The boolean variable recovering, initially false is set true when an error has been detected and we are trying to recover from it. If we are recovering from an error and procedure mustbe is called with a goal symbol, then the input is skipped until the goal symbol is found in the input. If we are not recovering and procedure mustbe is called, then either the symbol in the input is the goal symbol or we have an error in which case we call procedure report to report the error and put the translator in recovery mode. In such a situation the required goal symbol is assumed to be there and translation proceeds.

Finally the primitive 'have' that is actually a boolean function is used to see if a goal symbol is present or not in the input. The structure of this function is the following :

```
function have(symbol) : boolean ;
begin
  if token = symbol
     then begin
            scanner ;
            have := true ;
          end
     else have := false ;
end ; (* have *)
```

Thus the translator can check the current token with a specific goal symbol and take a certain action if it is there, or another action if it is not.

The worst case in this recovery would be if the second symbol in error which we are recovering on does not occur again in the input stream. In such a case the whole of the subsequent text is skipped.

Appendix I provides a translation listing of an erroneous BCPL program with the related error messages.

VII. RUN TIME CONSIDERATIONS

The code generated by the translator has to be supported by some stacks where space for variables at run-time will be provided. This is needed because BCPL routines may be called recursively and therefore the number of variables at run-time may may increase or decrease dynamically. Also some spare space has to be set aside for overhead tasks at run-time.

Furthermore a number of routines, that will provide the interface of the BCPL program with the operating system are needed. The structure of the assembly code generated has the following form :

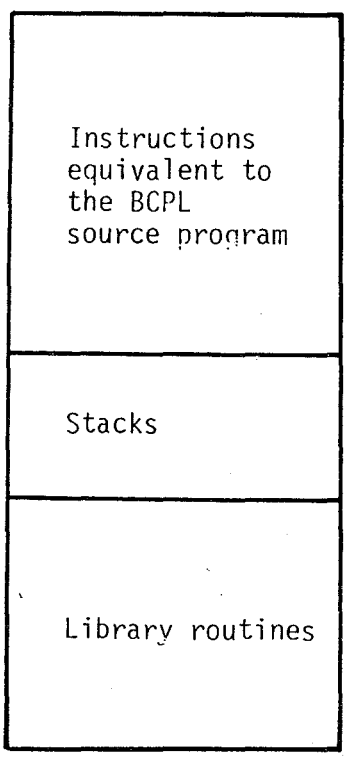


Figure 7.1 Structure of the code generated by the translator

The next sections explain how these stacks are utilized and manipulated, how registers are organized at run-time, and describe the library routines.

7.1 Run-time Stacks

The execution of the assembler code generated depends on four stacks. These are the 'run-time data' stack,'procedure-name' stack, 'for-loop' stack and 'register' stack. Though it would be better to have a single stack providing space for all of these four stacks they have been separated to simplify their usage and manipulation at run-time.

7.1.1 Run-time Data Stack (STACK$)

This is the largest stack among the four, and provides storage for the data words of the assembler program being executed. At the start of the execution the data words required by the main BCPL program are allocated on the stack. Then whenever a procedure is invoked the required number of data words for that procedure is appended to the end of the previous data words, and this goes on as other procedures are called recursively. The stack is expended in this way until the last procedure invoked finishes. At that time the data words that have been allocated for that procedure are removed from the stack. In this way a dynamic allocation and deallocation of the data words, or in other terms the 'data segments', for each procedure invoked is achieved.

As the base address of the data segments vary during run-time variable addressing is non-trivial. Section 7.4 describes variable addressing. This way of organizing data guarantees maximum storage economy : every data segment exists only during the execution of the procedure to which it belongs. It is created at procedure entry and discarded at exit. Both of these stack operations can be implemented at a relatively low expense.

To allow stacking and unstacking of data segments, a link is needed which is called 'dynamic link' (DL) and chains every data segment to its immediate predecessor in the stack. Variable addressing is done

through a second link which is called 'static link' (SL) and chains only those data segments which - according to the scope rules of BCPL - are currently accessible.

SL as well as DL are incorporated into the head of every data segment as the first and second words respectively. These links are manipulated by two library routines called R$ENTRY and R$EXIT at entry and exit of procedures respectively. Figure 7.3 illustrates the organization of the run-time data stack. In this figure BASE is the base address of the most recently created data segment. It is the head of the two chains and is hence used both for addressing variables and for the two stack manipulations. Stacking also requires another pointer (NEXT) defining the base address of the data segment to be stacked [5]. Section 7.4 describes where and how these two pointers are kept.

Two extra words have to be allocated somewhere in order to store the return address (RA) for that procedure and the return value (RV) if the data segment belongs to a function. These two words have been also allocated at the beginning of each data segment below SL and DL as follows :

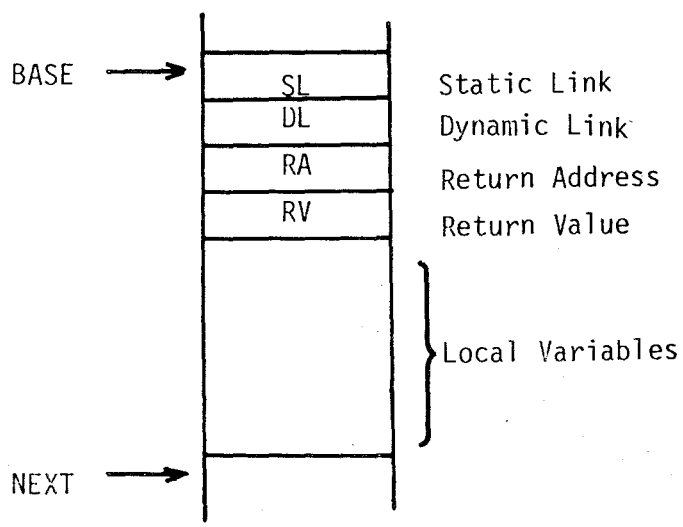
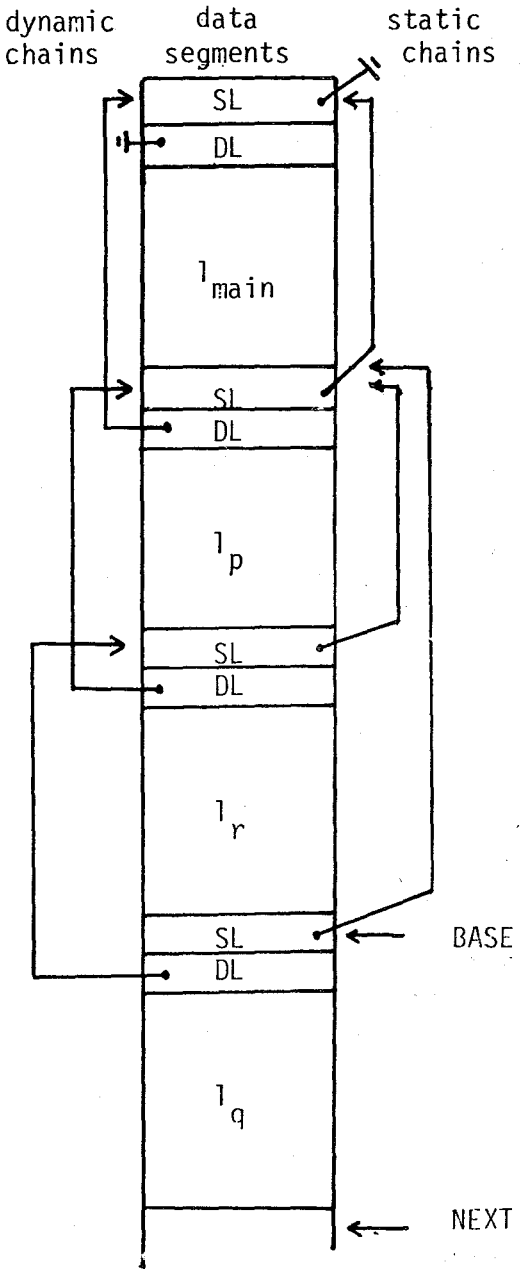


Figure 7.2 Representation of a data segment within the run-time data stack

```
      .
      .
      .
let   P() be
      let  Q()  be
           .
           .
           .
      and  R()  be
           .
           .
           Q() ;
           .
           .          ;
      .
      .
      R() ;
      .
      .          ;
      .    // main program
      .
      .
      P() ;
      .
      .
      .
finish
```

(a)



(b)

Figure 7.3 Run-time data stack

(a) A BCPL program

(b) Corresponding stack (growing downwards) of data data segments belonging to the calling sequence : main -> P -> R -> Q, where l_{main}, l_p, l_r, l_q are the data segments for local variables of these procedures

7.1.2 Procedure-name Stack (N$STACK)

This stack is used to push the name of any procedure invoked. Again a new entry is made at procedure entry and discarded at procedure exit. This stack becomes useful in case of a run-time error, when a library routine called TRACE$B is invoked to trace back the stack and dump the names of the active procedures in the following form :

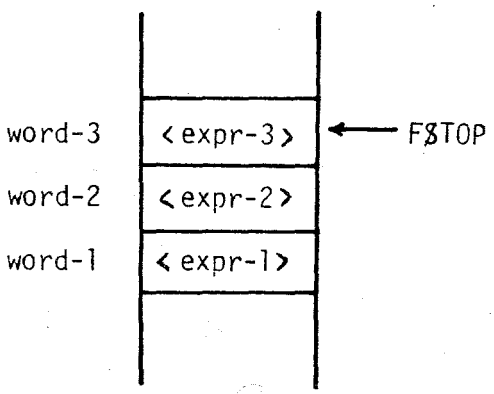
```
run-time error explanation
ERROR OCCURED AT PROCEDURE  zzz NEAR LINE nn
  ... PROCEDURE zzz WAS CALLED FROM PROCEDURE yyy
  ... PROCEDURE yyy WAS CALLED FROM PROCEDURE xxx
            .
            .
            .
  ... PROCEDURE xxx WAS CALLED FROM MAIN PROGRAM
```

7.1.3 For-loop Stack (F$STACK)

This stack is used to implement for-loops of the BCPL language. It is especially useful in cases of nested loops. Three words from the stack are allocated for each loop, and these words are deallocated when the execution of the loop terminates. As an example the following for-loop :

for i:=<expr 1> to <expr 2> by <expr 3> do <command>

necessitates the following allocation of words within the stack :



where word-1 is used to place the starting value of the loop which in this case is the value of <expr 1>, word-2 is used for the last value of that loop that is <expr 2>, and word-3 is the step count ie. the value of <expr 3>.

At the beginning of the loop a test is made to see whether the loop can be entered or not. If yes then the value within the first word is placed into the data word associated with variable 'i' and the loop is entered. At the end of the loop the value of the first word is incremented or decremented by adding the value in the third word to it and control goes to the beginning of the loop.

The user can use and even alter the value of 'i' within the body of the loop since this value is restored when the loop is entered from the first word allocated for that loop in the stack. It is even possible and acceptable to have nested loops all with the same index variable. This way of organization makes things easy both at translation and execution time.

Four different and small library routines called FOR$1, FOR$2, FOR$3 and FOR$4 are used to manipulate and make possible the usage of for-loops.

7.1.4 Register Stack (R$STACK)

Since function calls are available only from within expressions something has to be done inorder to save the contents of registers used thus far for the evaluation of the expression before jumping to any function. Therefore whenever a function call is encountered during the evaluation of an expression, a call is made to a library routine called SAV$REG to save registers X1 through X5 into the register stack. Upon return from the function another library routine, RES$REG is invoked to restore the contents of those registers and the evaluation of the expression continues.

This enables the economization of registers that are small in number, and makes all registers from X1 to X5 available at function entry to that function.

7.2 Size of the Stacks and Translator Directives

The allocation of the appropriate size of the stacks is not transparent to the user since he may frequently encounter one of the following run-time error messages :

RUN-TIME DATA STACK OVERFLOW
PROCEDURE-NAME STACK OVERFLOW
FOR-LOOP STACK OVERFLOW
REGISTER STACK OVERFLOW

The first two run-time errors are usually obtained if the BCPL program is highly recursive. The third error occurs when a large number of nested for-loops exists. The last one is issued when the program is highly recursive in terms of function calls.

The translator by default provides the following sizes for the above four stacks :

Run-time data stack	:	3072 words
Procedure-name stack	:	128 words
For-loop stack	:	48 words
Register stack	:	128 words

This reveals the fact that even if the run-time data stack is not exhausted only 128 procedures or functions may be active at a time. Also only 16 nested for-loops are allowed and the depth of recursion for functions is not more than 25.

However four different translator directives exist to define larger or even smaller sizes for these stacks. These are the following :

#S size for the run-time data stack
#N size for the procedure-name stack
#F size for the foor-loop stack
#R size for the register stack

Each of them starts with a # in column one, and column two muste be one of the letters : S,N,F or R. The 'size' is a decimal number that can be placed anywhere on the same line and specifies the maximum length for the related stack. The translator does not check the size, so the user must bear in mind that the total program size can not exceed the maximum field length specified by the installation, and total program size is the sum of : instruction words generated, size of constants table, size of required library routines, plus the size of the four stacks. All of these values are given as 'program statistics' at the end of translation listing.

Two extra translator directives exist : #L+ or #L- that turns the translation listing on and off, and #P that forces the translator to eject a page on the listing.

7.3 CDC CYBER 170/815 Registers

CDC CYBER 170 series, model 815 maintains 24 operational registers namely A0 to A7, X0 to X7 and B0 to B7. Registers X0 to X7 are 60 bits long and are used to hold data while registers B0 to B7 are 18 bits long and their major purpose is to hold either addresses or be used as increment registers with the exception of B0 that is hardwarewise set to zero. A or address registers are also 18 bits long and they have a special way of use in the sense that they form pairs with their corresponding X registers except A0 that is independent of X0 and vice versa. Whenever an address is set into any of A1 to A5 registers the data at that memory location is transferred to the corresponding X register, while an address, set into one of A6 or A7 registers has the effect of storing the data hold by the corresponding X register to that memory location. Thus A0 can be used to hold any address without affecting the contents of register X0.

7.4 Run-time Organization of Registers

Since registers represent a scarce resource they have to be used efficiently by the translator so that it does not easily run out of registers during translation, and successfully translate any BCPL program.

However the danger mentioned above is always present in long and nested expressions. Therefore the translation of such expressions will be terminated with a message : 'expression too complex, simplify it' .

The evaluation of an expression depends on X registers since the length of data within the computer is 60 bits long. The algorithm adopted to handle the allocation of X registers is a simple one and mainly depends on registers X1 to X5 because only these registers have the possibility of being loaded from memory and evaluating an expression basically means get data from memory and process it. To achieve this objective registers X1 to X5 have been organized as a stack, and whenever the translator requires a register a routine called 'getreg' provides the next available one if any exists, otherwise the translator has run out of registers due to a highly nested expression. On the other hand whenever the translator finishes with a register - that is always the last one obtained - a routine called 'freereg' is invoked to release that register and make it available for future requests.

Registers X0,X6 and X7 are used for intermediate results in evaluating parts of expressions and unlike registers X1 to X5 they are not required to be allocated and deallocated. Register X6 is particularly used to store the results of evaluated expressions into memory.

Register A0 is used to hold addresses temporarily, and registers A1 to A7 are utilized together with X1 to X7 as pairs in evaluating expressions.

Registers B0 to B7 are of special use since most of them keep addresses that are pointers to data segments within the run-time data stack. As mentioned above B0 is hardwarewise set to zero. B1 is set to one softwarewise by an instruction at the beginning of the generated assembler code. Therefore these two registers can be used as the values zero and one in setting Xregisters and in this way some optimization in the code generated is achieved. To illustrate this optimization the instruction : SXi 1 is a 30 bit one while the instruction : SXi B1 is a 15 bit one, and obviously the values zero and one are the most frequently used values.

Register B2 holds the address of the first word of the run-time data stack which is also the main program data segment start address. Register B3 holds the current data segment start address which is a pointer to the data area of the currently executed BCPL routine at run-time

or translated at translation time.

Register B4 holds next available location address of the run-time data stack. A simple formula to calculate contents of B4 can be : (B3)+4+l_c where the four words are the header words of the current data segment and l_c is the number of data words or local variables of the routine beingexecuted. Figure 7.4 illustrates this organization.

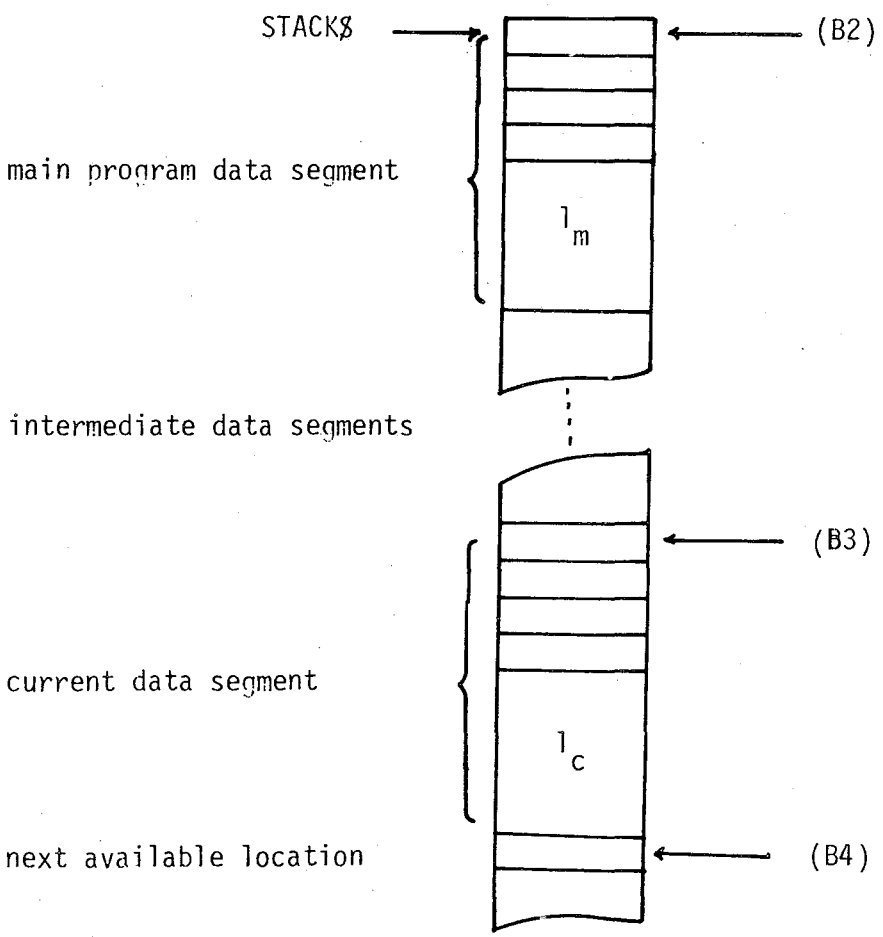


Figure 7.4 Run-time organization of B2,B3 and B4 registers

l_m : data words for main program local variables

l_c : data words for current routine local variables

Register B5 has been devoted to hold the line number of the BCPL program being executed. Its value is used to report the position of a run-time error in terms of BCPL source code program line number in the following form :

```
run-time error explanation
ERROR OCCURED AT MAIN PROGRAM NEAR LINE n
```

or

```
run-time error explanation
ERROR OCCURED AT routine name NEAR LINE n
```

Register B6 and B7 are used to hold addresses, offsets to be added on addresses, shift counts to be used in shifting data within X registers left or right, and for other minor objectives. In particular B6 is used to hold the start address of a data segment other than the current one or the main program data segment since pointers to them are in registers B3 and B2 respectively. This makes it possible for a routine to address non-local variables. Thus to address a variable that belongs to the nth previous data segment the following code is generated (the contents of the first word of a data segment header is the static link) :

```
1.  SAxtop  B3
2.  SAxtop  Xxtop
3.  SAxtop  Xxtop
      .
      .
      .
n.  SAxtop  Xxtop
    SB6     Xxtop
```

where Xxtop is an X register from X1 to X5 obtained before the above code is generated and released afterwards. In this way any variable within that data segment can be referenced by an instruction of the form :

```
SAi  B6+3+v
```

where v is the ordinal place of that variable within that data segment.

VIII. THE BCPL LIBRARY

A number of assembler routines have been prepared to provide for the interface of any BCPL program with the operating system. The translator after finishing the translation of the BCPL program, reads the file where these routines reside (BCPLLIB) and appends those that are required at the end of the generated assembler code. These routines, in turn, utilize certain system supplied routines from a system library file called RELOPL which are entered at assembly time by COMPASS. Therefore these two files have to be local when the BCPL program is translated and then the translated code is assembled.

Library routines of BCPL are mainly used for input and output, for run-time error handling, for expanding and shrinking the run-time data stack, for storing and restoring contents of registers, and for various other purposes. The listing of the whole BCPL library can be found in Appendix C.

8.1 Input / Output Routines

Input / output routines of the BCPL language of this thesis are not standard and apply only for this implementation. They have been designed to resemble those of the Pascal language, so they are quite simple and easy to use. There are a total of six routines, two of which provide for input and four for output. All make use of the standard input and output files. Therefore input and output for other files are not supported. However the experienced programmer may write his own routines to manipulate files other than these two, and call them from his BCPL program by inserting the appropriate assembler instructions into the code generated by the translator, using the facility of the keyword compass.

8.1.1 Routine READLN

This routine is used to read the next input line from file input into a buffer. The length of the buffer is eight words long so it provides for 80 characters that is the length of a card image.

The BCPL keyword is : readln

The code generated to call this routine is : RJ RD$LN where RD$LN is the entry point of that library routine and RJ means 'return jump' to the specified entry .

If the end of file is reached an appropriate run-time error is issued and the program is aborted.

8.1.2 Routine READCH

This routine returns the next character from the input buffer if any exists. Otherwise calls routine RD$LN to fill the buffer and then it extracts the first character and returns it.

The BCPL form is : readch (<name>)

The code generated to call this routine is : RJ RD$CH where RD$CH is the entry point of the library routine. The character to be returned is placed into register X1 left justified and with zero fill.

8.1.3 Routine WRITELN

This routine dumps the contents of the output buffer to file output as a line image. The length of the buffer is 13 words thus it provides space for 130 characters to be written on a line. However the first byte or character is used for cariage control and is not printed but has the specified effect defined by the operating system if it is a valid carriage control character.

The BCPL keyword is : writeln

The code generated to call that routine is : RJ WR$LN where WR$LN is the entry point of the library routine.

8.1.4 Routine WRITESTR

This routine is used to place a string into the output buffer starting at the next available byte. If the buffer is full then routine WR$LN is used to dump the buffer to file output and empty it. Then the string is placed to the buffer.

The BCPL form is : writestr (<expr1>{:<expr2>} ,{<expr1>{:<expr2>}}*)

The code generated to call this routine is : RJ WR$STR where WR$STR is the entry point of the library routine.

In the above form <expr1> is an expression that evaluates into a bit string that is interpreted as a ten character left justified string. The second expression, <expr2> , separated by a colon from the first one is the format designator indicating the number of characters to be printed. If the format designator is not present then a value of one is assumed. More than one string can be output at a time.

At entry to the routine register X1 holds the string and register X2 holds the format designator.

8.1.5 Routine WRITEN

This routine is used to place the decimal equivalent of a bit string into the output buffer. Again if no empty byte exists in the buffer routine WR$LN is called to dump its contents and empty it.

The BCPL form is : writen (<expr>{,<expr>}*)

The code generated to call this routine is : RJ WR$NUM where WR$NUM is the entry point of that library routine.

In the above form <expr> is an expression that evaluates into a bit string to be interpreted as a number. This routine converts this number to a character string by calling some system supplied routines and then places it to the output buffer. More than one decimal values can be output at a time.

At entry to this routine register X1 contains the value of the expression.

8.1.6 Routine WRITEO

It was inevitable to have a facility for printing out the octal equivalent of bit strings. Such a facility is made necessary by two facts. First BCPL supports the octal type, and secondly it is a language used for systems programming where shifting and masking of bit strings and manipulating addresses are possible.

The BCPL form is : writeo (<expr>{,<expr>}*)

The code generated to call this routine is : RJ WR$O where WR$O is the entry point of that library routine.

In the above form <expr> is an expression that evaluates to a bit string to be interpreted as a number. This routine converts a given number to an octal character string by calling system supplied routines and places it to the output buffer. More than one octal values can be handled at a time. At entry to this routine register X1 contains the value of the expression.

8.2 Routines that Support Execution of For-loops

The treatment of for-loops is unusual as explained in section 7.1.3. A stack is used to achieve this way of organizing for-loops. This is the for-loop stack or F$STACK in terms of assembler code generated. Four different small routines are utilized to manipulate this stack and the three values of the for-loop : initial, last and step value. Description of these routines follows :

i. The first routine is called at the beginning, before the code for the loop starts. This routine is used to allocate three words in the for-loop stack and report an error if the stack overflows.
   The code generated is : RJ FOR$1 where FOR$1 is the entry point of that library routine.

ii. The second routine is used to test the step counter and report an error if it is an invalid one.
   The code generated is : RJ FOR$2 where FOR$2 is the entry point of that library routine.

At entry to this routine register X1 contains the value of the step counter if it was specified on the for command, otherwise a value of one is assumed. This fact is indicated by two values, one and zero respectively, within register B7. Then the value of the step counter in X1 is compared with the limiting values of the loop, that is the initial and last value. An error is reported and the program is aborted if the step counter does not confirm.

iii. The third routine is called at the beginning of the loop and returns information of whether the loop can be entered or not. This is indicated by using register B7 as a flag. If the loop can continue then the value zero, otherwise the value one is set into register B7. Register B7 is checked at the beginning of the loop. If it contains zero the loop is entered otherwise a jump is performed to the next instruction after the loop. The code generated is : RJ FOR$3 where FOR$3 is the entry point of that library routine.

iv. The fourth and last routine is called at the end of the loop and upon return a jump is performed to the beginning of the loop. This routine increments the initial value of the loop that resides in the first word by the step value in the third word.
The code generated is : RJ FOR$4 where FOR$4 is the entry point of that library routine.

8.3 Routines for Entering and Exiting Procedures

Two routines are used to manipulate the run-time data stack and procedure-namestack when entering and exiting procedures. Section 7.4 describes how the registers that are manipulated by these routines have been organized.

8.3.1 Routine to Enter a Procedure

This routine is used to allocate the required number of data words for a procedure entered, manipulate the dynamic link (DL), manipulate the

static link (SL) according to the type of call, change the pointer for the next available location in the run-time data stack, and place the procedure name into the procedure-name stack.

The code generated is : RJ R$ENTRY where R$ENTRY is the entry point of that library routine.

At entry to this routine, register X1 contains the name of the routine,register X7 contains the number of data words that the new procedure requires and B7 contains the call type in the following form :

0 - for calling itself or calling its outer one or calling one from its own level

1 - for calling one from its inner procedures

First the name of the procedure is placed to the procedure-name stack (N$STACK) if there is an empty location, otherwise an error is issued and the program is aborted. Then the dynamic link is created by using the contents of register B3 which points to the data segment of the calling procedure :

DL ← B3

The static link is created by taking account the call type that register B7 shows :

if B7 = 0 then SL is set to point the same place where the previous data segment's SL is pointing : SL ← (B3).

if B7 = 1 then S1 is set to point the starting address of the previous data segment : SL ← B3 .

Then register B3 is set to point at this new data segment by using contents of register B4 :

B3 ← B4

B4 is also updated to point 4+X7 words further then it used to :

B4 ← B4+4+X7

where the first four locations are used for the header of the data segment and the X7 locations are those required by the procedure . If at that time B4 exceeds the length of the stack an error is issued to inform the overflow condition and the program is aborted.

8.3.2 Routine to Exit a Procedure

This routine does just the reverse of routine R$ENTRY but the algorithm is not as complicated as before. First the stack top pointer of procedure-name stack is decremented. Then it manipulates registers B3 and B4 so that the stack shrinks. B4 is set to point to the data segment of the routine we are going to exit and B3 is set to point to the data segment of the previous routine that was the caller of the former one :

B4 ← B3
B3 ← (B3+1)

where B3+1 is the DL of the last routine.

The code generated is : RJ R$EXIT where R$EXIT is the entry point of that library routine.

8.4 Routines for Saving and Restoring Registers

Whenever a function call is encountered during the execution of an expression registers used so far for the evaluation of the expression have to be saved before the function is called. These registers have also to be restored after control returns back from the function inorder to continue with the evaluation of the expression. Two library routines perform these tasks.

i. The first routine saves registers X1 to X5 into the register stack (R$STACK). If the stack overflows a run-time error is issued and the program is aborted.
The code generated is : RJ SAV$REG where SAV$REG is the entry point of that library routine.

ii. The second routine is called upon return from the function and restores registers X1 to X5 from the stack. In this way the evaluation of the expression can continue.
The code generated is : RJ RES$REG where RES$REG is the entry point of that library routine.

8.5 Other Run-time Routines

Three more library routines are utilized at run-time. Two of them are for reporting run-time error messages and the third is used to execute a halt command issued by the user.

i. The first one is for reporting an error message in case of a switchon into command that does not have a default label and the value of the switchon expression does not match with any of the case labels. After the error is issued the program is aborted.
The code generated is : RJ CASE$ER where CASE$ER is the entry point of that library routine.

ii. The second routine is used to report an illegal left or right shift count and abort the program. A shift count is illegal if it is less than zero or greater than 60.
The code generated is : RJ SHF$ERR where SHF$ERR is the entry point of that library routine.

iii. The third routine is called in case a user halt command is encountered. This routine gives a run-time message : 'PROGRAM HALTED VIA A HALT COMMAND' and aborts the program.
The code generated is : RJ HALT$ where HALT$ is the entry point of that library routine.

8.6 Reporting the Place of a Run-time Error

As described in section 7.4 register B5 has been dedicated to keep the source line of the program being executed. The procedure-name stack on the other hand holds the names of all the active routines. Therefore

all the information to determine the place of a run-time error is available.

Whenever a run-time error is issued by any of the library routines, another routine is called before the program is aborted. This routine reports the line number of the run-time error using contents of register B5. Then it traces back the procedure-name stack and prints the names of the active procedures together with the names of the procedures that called them. The listing starts with the last active procedure and continues till the main program. Thus a minimal debugging facility is provided.

The code to call that routine is : RJ TRACE$B where TRACE$B is the entry point of that library routine.

IX. CASE STUDIES

Five different problems have been programmed using the BCPL language and executed after they have been translated by the translator that is the subject of this thesis. Each problem is of a different nature and they have been used to test the translator and the code it generates.

9.1 Towers of Hanoi

9.1.1 Definition

The first problem is the 'Towers of Hanoi' problem whose initial setup is shown in figure 9.1. Three pegs A,B and C exist. Five disks of differing diameters are placed on peg A so that the larger disk is always below a smaller one. The object is to move them to peg C using peg B as auxiliary. Only the top disk of any peg may be moved to another peg and a larger disk may never rest on a smaller one.

9.1.2 Solution

A recursive solution has been stated for the problem as follows [8]:

To move n disks from A to C using B as auxiliary
1. if n = 1 then move the single disk from A to C and stop
2. move the top n-1 disks from A to B using C as auxiliary
3. move the remaining disk from A to C
4. move the n-1 disks from B to C using A as auxiliary

The problem has been solved with n = 5 ie. five disks though it works with a practically infinite number of disks. The results verify with those in the book of Tenenbaum and Augenstein [8]. A listing of the BCPL program and results have been included in Appendix D.

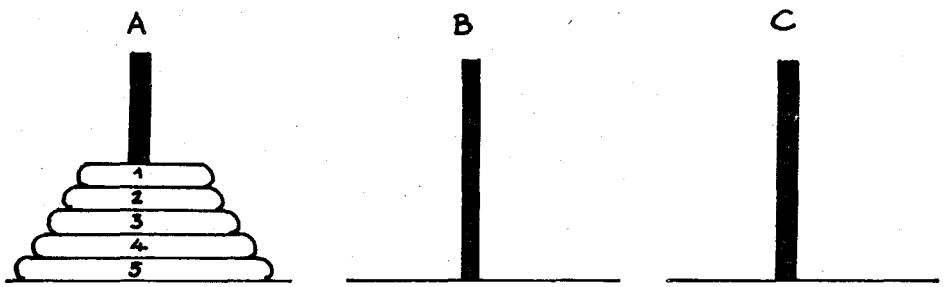


Figure 9.1 Initial set-up of the Towers of Hanoi problem

9.2 Tree Differentiation

9.2.1 Definition

This is a problem of finding the derivative of a formula with respect to the variable x. Programs for algebraic differentiation were among the first symbol manipulation routines ever written for computers; they were written as early as 1952 [9].

9.2.2 Solution

The idea behind the solution algorithm is to traverse a right threaded binary tree, that contains the original formula, in postorder, forming the derivative of each node as we go until eventually the entire derivative has been calculated. Using postorder means, we arrive at an operator node in the tree after its operands - left subtree and right subtree if exists - have been differentiated. As the algorithm proceeds

a new tree that represents the derivative of the formula is built up by creating new nodes or by making copies of subtrees of the original tree.

The problem and the differentiation algorithm has been taken from Knuth [9]. The following formula is the object of the problem :

$$3 * \ln (x+1) - \frac{a}{x^2}$$

The postorder form of the above formula is the following :

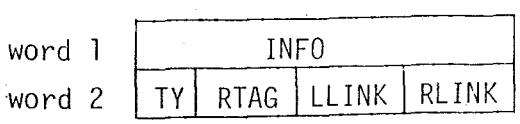
3 x 1 + L * a x 2 ↑ / -

and the corresponding right threaded binary tree can be found in Figure 9.2(a)

The algorithm is capable of calculating the derivative D(y) for any formula y composed of the following operators and rules :

D(x) = 1
D(a) = 0 , if a is a constant or a variable ≠ x
D(ln u)= D(u)/u
D(-u) = -D(u)
D(u+v) = D(u)+D(v)
D(u-v) = D(u)-D(v)
D(u*v) = D(u)*v + u*D(v)
D(u/v) = D(u)/v - (u*D(v))/(v↑2)
D(u↑v) = D(u)*(v*(u↑(v-1)))+((ln u*D(v))*(u↑v)

Any of the two right threaded binary trees is composed of nodes having the following structure :



The first word of any node contains the information : either a 60 bit integer or a ten character variable name or a right justifed single character represerting any valid operator :

↑ (for exponentiation),/,*,-,+,N(for negation),L(for natural logarithm)

In the second word the TY field shows the type of the information contained in the first word. Valid entries of this field are the following :

0 for a 60 bit integer constant
1 for any variable
2 for natural logarithm (L)
3 for negation (N)
4 for addition (+)
5 for subtraction (-)
6 for multiplication (*)
7 for division (/)
8 for exponentiation (↑)

The above sequence also corresponds to the priority of operators that the differentiation algorithm has to take into account.

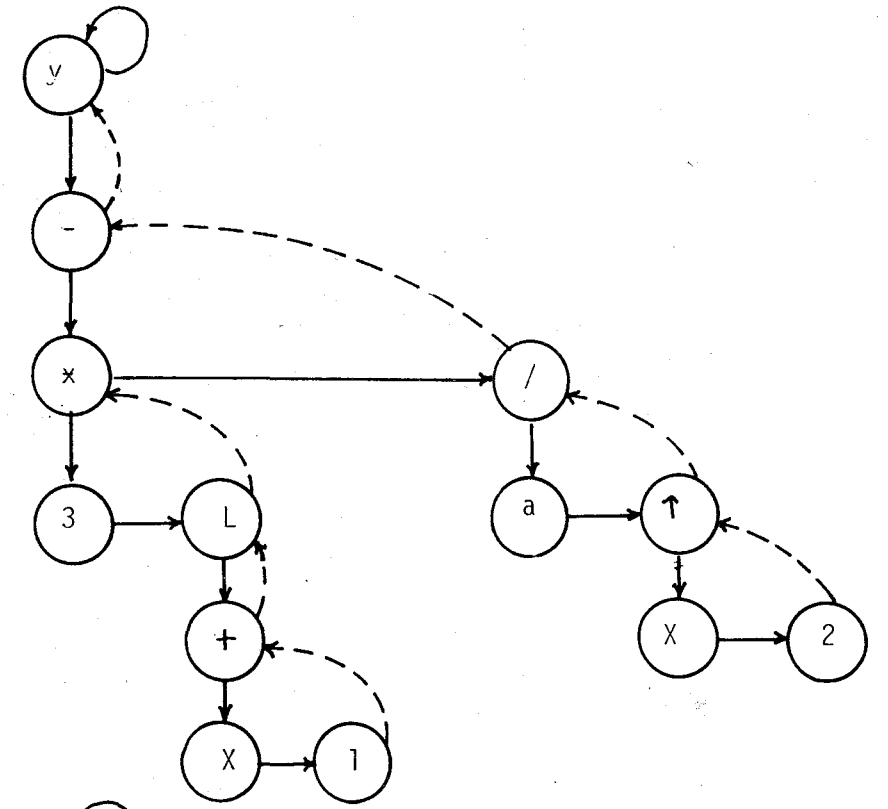
The RTAG field indicates whether the RLINK (right link) field is a thread pointing to the postorder successor of that node or an actual right link pointing to the right subtree of that node. In the first case RTAG contains zero while in the latter it contains the value one.

The LLINK field is just a pointer to the left subtree of that node if it has a left subtree, otherwise this field contains zero.

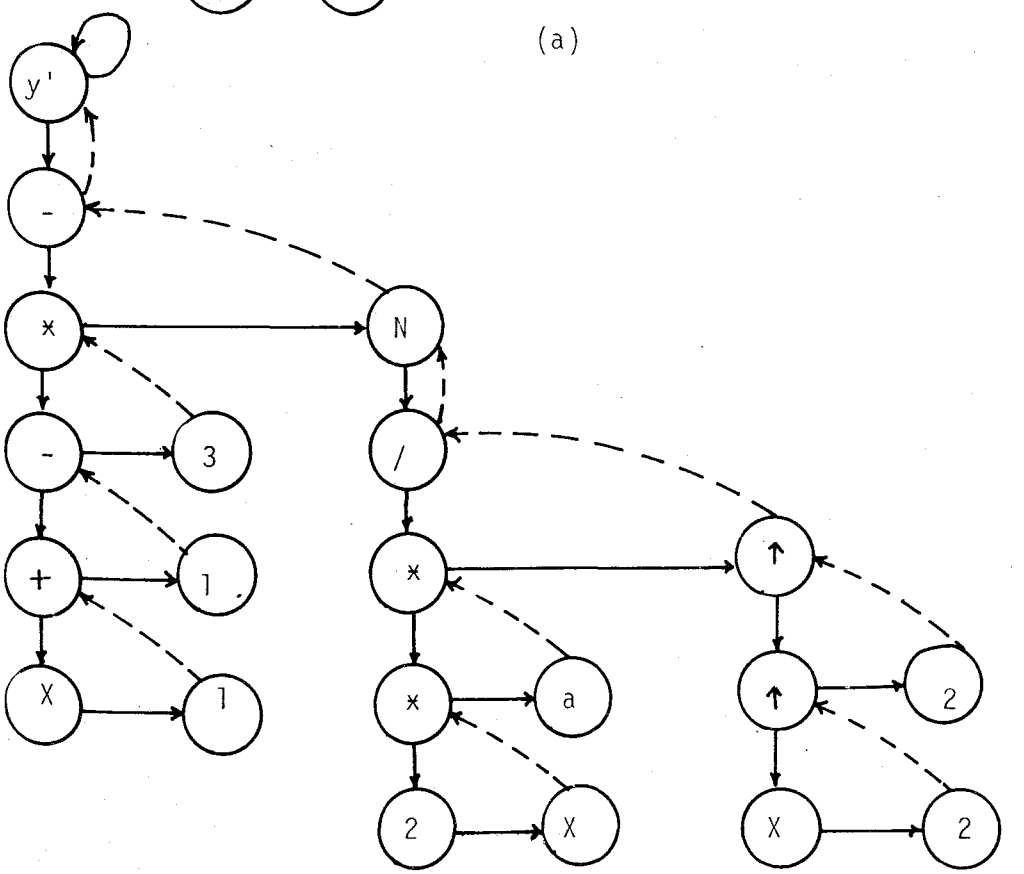
Both the LLINK and RLINK fields contain real addresses within the programs field length. This is an advantage of the BCPL language to be able to manipulate addresses and use them as pointers.

The listing of the program together with its input data and results can be found in Appendix E. The resulting postorder form of the differentiated expression is the following :

3 1 x 1 + / * a 2 x * * x 2 ↑ 2 ↑ / N -



(a)



(b)

Figure 9.2 Right-threaded tree representation of expressions
(a) right-threaded binary tree of original formula
(b) right-threaded binary tree of diffirentiated formula

which corresponds to the formula :

$$3\,\frac{1}{x+1} - N\,\frac{a2x}{(x^2)^2}$$

that is the derivative of the original formula. The right threaded binary tree of the differentiated formula has been given in Figure 9.2(b) for convenience.

The program has been tested with other more complex formulaes and expected normal results have been obtained.

9.3 The Eight Queens

9.3.1 Definition

The problem here is to place eight queens on a chess-board in such a way that no queen is threatening the cell that any other queen resides. Also all valid positionings have to be found. The number of them is known to be 92 while the number of all possible placements is $\binom{64}{8}$ that is 4 426 165 368.

9.3.2 Solution

The algorithm is a simple one that tries all possible placements of the eight queens on the board. However since this would require a large amount of computer time, to minimize the number of trials some simple assumptions have been made :

i. Only one queen can exist on a certain row.
   This implies that each queen can be placed only on a certain row and this minimizes the number of possible placements to 2 377 216 which is eight to the power of eight.

ii. Only one queen can exist on a certain column and on the two diagonals that pass from that cell.
   This implies that after queen n has been placed in row n the number of cells that queen n+1 can be placed on row n+1

is $0 \leq$ # of cells < 8. So the above number is reduced to a smaller one. Figude 9.3 illustrates this fact.

The listing of the program for this problem and all 92 solutions can be found in Appendix F.

row of								number of places
queen 1	Q	.	.	.	.	.	.	.	8
queen 2	.	.	Q	.	.	.	.	.	6
queen 3	.	.	.	.	Q	.	.	.	4
queen 4	.		.	.	.	.			3
queen 5	.		.		.	.	.		
queen 6	.	.	.		.	.	.	.	
queen 7	.		.		.		.	.	
queen 8	.		.		.			.	

Figure 9.3 The eight queens problem (Q represents a queen, dots represent threatened cell by previously placed queens)

9.4 The number PI (π)

9.4.1 Definition

In this problem the purpose is to calculate the number PI with a very high precission after the decimal point. This number of digits after the decimal point have been taken to be 125.

9.4.2 Solution

The number PI has been calculated using the following series :

$$PI = 3 + 6 \sum_{n=1}^{\infty} \frac{(0.5)^{2n+1} \left\lceil_{k=1}^{n} (2k-1)\right.}{(2n+1) \left\lceil_{k=1}^{n} 2k\right.}$$

Since no computer system has a word large enough to keep a real number in such a high precission an array of words where each one will keep a certain amount of digits will have to be used. An integer array would be more suitable for this purpose . Also since BCPL does not have the type real, calculation of these series has to be made with an integer array.

In the BCPL program a 30 element vector has been used. Each element provides storage for five consequtive digits after the decimal point and so a total storage for 150 digits is available. However only the first 25 elements of the vector have been used to display the result because the values in the last elements will highly depend on the digits to the right of the 30th element and on a larger number of terms of the series calculated.

A listing of the program and its results can be found in Appendix G. The program has used the first 215 terms of the series to calculate the number PI displayed. Also the first 40 digits of the number verify to those given by Knuth [9].

9.5 Quick Sort

9.5.1 Definition

In this problem the purpose is to sort 1000 integer numbers using quick sort algorithm. The program has to be written recursively and the sort time has to be calculated.

9.5.2 Solution

The quick sort algorithm has been taken from Tenenbaum and Augenstein [8] and is the following :

Let X be an array of n (here n=1000) the number of elements in the array to be sorted. Choose an element A from a specific position within the array (eg. it can be thefirst element so that A=X(1)). Suppose that the elements of X are rearranged so that A is placed into position j and the following conditions hold :

i. Each of the elements in positions 1 through j-1 is less than or equal to A.

ii. Each of the elements in positions j+1 through n is greater than or equal to A.

Notice that if these two conditions hold for a particular A and j then A remains in position j when the array is completely sorted. If the process is repeated with subarrays X(1) through X(j-1) and X(j+1) through X(n) and any subarray created by the process in successive iterations, the final result is a sorted file.

The next requirement, that of finding the sort time, can be solved by issuing a system request to get the time before and after sort and obtain the difference. Since BCPL does not provide such a facility this could be achieved by inserting two extra assembler codes using the keyword compass as follows :

```
COMPASS     TIME  B2+v
```

where B2+v is the address of a variable from the main program that system will place there the accumulated central processor time used by the job. Thus the sort time obtained is approximately 950 milliseconds. Appendix H provides the listing and results of this program.

X. CONCLUSION

In this thesis, the implementation details of a BCPL translator have been described. Five case studies explained in Chapter IX have been included to prove the translators capability in terms of recognizing BCPL programs correctly and in terms of producing correct assembler code. In each case, study, problems of a different nature have been solved to show the power of the BCPL language implemented. Finally a syntactically incorrect BCPL program has been given to the translator inorder to see its error recovery canability.

The simplicity of implementing the Recursive Descent technique has been stated implicitly in Chapter V. The translator has also been programmed in a high level language : Pascal. Therefore it can be modified easily, so some flexibility exists in adding extra features to the BCPL language which the translator currently does not recognize. These could be constructs of BCPL which have not been implemented or other useful tools to make systems programming easier.

The keyword compass added to the syntax of the BCPL language implemented has been proved to be useful in case study five where the time of sorting 1000 numbers with quick sort is sought. Thus facilities of the assembler that do not exist in BCPL can be employed.

Improvements also can be made to make both the translation and execution of a BCPL program efficient. Translation time can be shortened by rewriting the translator in a more efficient and effective way. Execution time on the other hand can be decreased by rewriting the BCPL library inorder to have more efficient input / output routines since these are the most time consuming parts of a BCPL program during run-time. Also different techniques can be adopted while writing the translator's code emitting constructs and the logic of them to produce better and more efficient code.

Furthermore the external feature of BCPL can be added to this implementation inorder to be able to write better systems programs. All of them can be done as a future project.

APPENDICES

APPENDIX A. SYNTAX OF THE IMPLEMENTED BCPL LANGUAGE

Character set of the language :

A through Z
0 through 9
* / + - : ; , = < > ( ) [ ] ! $ # " blank

Precedence of operators listed in order of highest to towest :

( )	Parenthesis, beginning with innermost pair
lv,rv	Address manipulation, from left to right
lshift,rshift	Left and right shift, from left to right
*,/,rem	Multiplication, division, and remainder, from left to right
+,-	Additon and subtruction, from left to right
=,<>,<,<=,>,>=	Relational operators, from left to right
not	Complement
land	Logical and
lor	Logical or
eqv	Logical equivalence
neqv	Logical exclusive or or not equivalence

Following is the syntax of the BCPL language in BNF notation as implemented in this thesis :

<letter>::= A/B/C/ ... /Z
<digit >::= 0/1/2/ ... /9
<odigit>::= 0/1/2/ ... /7
<number>::= <digit>{<digit>}* / $<odigit>{<odigit>}*
<string>::= "<char>{<char>}$_0^9$"
<name >::= <letter>{<letter>/<digit>}*
<const >::= <number>/<manifest>/<string>/ true / false
<relop >::= = / <> / > / >= / < / <=

```
<addop>::= +/-
<mulop>::= * / / / rem
<shfop>::= lshift / rshift
<addrop>::=lv / rv
<program>::=<block>
<block>::= {<declaration>;}* {<command>;}* finish
<declaration>::=static <declbody> /
                manifest <declbody> /
                let <declist> {and <declist>}*
<declbody>::= [<def> {;<def>}*]
<def> ::=<name> : <const>
<declist>::= <name>(<fpl>) = <funcbody> /
             <name>(<fpl>) be <procbody> /
             <name> = table <const>{,<const>}* /
             <name> = vec <const>
<funcbody>::= valof [<block>] /
              valof <command> /
              <expr>
<procbody>::= [<block>] / <command>
<fpl> ::= <namelist> / <empty>
<namelist>:: <name> {,<name>}*
<command>:: goto <name> /
            if <expr> do <command> /
            while <expr> do <command> /
            unless <expr> do <command> /
            repeat <command> until <expr> /
            test <expr> then <command> or <command> /
            return /
            resultis <expr> /
            for <name>:= <expr> to <expr> {by <expr>} do <command> /
            switchon <expr> into <casebody> /
            [<compound command>] /
            <name> ( ) /
            <name> ( <elist> ) /
            <name>:= <expr> /
```

```
        <addrop> <name> := <expr> /
        <name>!<name> :=<expr> /
        <name>!<const> := <expr> /
        <name> : <command> /
         readln  /
         readch (<name>)  /
         writeln  /
         writestr (<expr>{:<expr>} { ,<expr>{:<expr>}}* )  /
         writen (<expr>{,<expr>}*)  /
         writeo (<expr>{,<expr>}*)  /
         compass  <compass assembler instruction> /
         halt  /
        <empty>
<compound command> :: <command>{; <command>}* /  empty
<casebody> :: { case <const> : <command> }*
              { default      : <command> }
                endcase
<elist> :: <expr> {,<expr>}*
<expr>  :: <expr0> { -> <expr>,<expr>}
<expr0> :: <expr1> { neqv <expr1>}*
<expr1> :: <expr2> { eqv  <expr2>}*
<expr2> :: <expr3> { lor  <expr3>}*
<expr3> :: <expr4> { land <expr4>}*
<expr4> :: { not }<expr5> { <relop> <expr5>}*
<expr5> :: <expr6> { <addop><expr6>}*
<expr6> :: {<addop>} <expr7> {<mulop>< expr7>}*
<expr7> :: <expr8 > {<shfop> <expr8>}
<expr8> :: {<addrop>} <name>  / <expr9>
<expr9> :: <name>  /
           <const> /
           <name>!<name>  /
           <name>!<const> /
           <name> ( )     /
           <name> ( <elist> )  /
            ( <expr> )
```

In the above syntax

/ means an alternate follows

$\{\}$ means repeats zero or more times

$\{\}^{*}$ means repeats zero to infinit times

$\{\}_{i}^{j}$ means repeats i to j times

Following are the reserved keywords of the BCPL language :

and	halt	resultis	to
be	if	return	unless
by	into	rem	until
case	land	lshift	valof
compass	let	lv	vec
default	lor	rshift	while
do	neqv	rv	writeln
endcase	not	static	writen
eqv	or	switch	writeo
finish	readch	table	writestr
for	readln	test	then
goto	repeat		

APPENDIX B. SOURCE LISTING OF THE TRANSLATOR

```
 SOURCE LISTING OF BCPL2                                    PASCAL/170 1.0  567


000004      1 (* BCPL *)
000004      2 (*$W99999D*)(*$R+*)
000004      3
000004      4
000004      5     (*************************************************************************
000004      6      *                                                                       *
000004      7      *   AN  IMPLEMENTATION  OF  A  BCPL  TRANSLATOR  USING  RECURSIVE       *
000004      8      *                      DESCENT  TECHNIQUE                               *
000004      9      *                                                                       *
000004     10      *                                                                       *
000004     11      *   THIS   TRANSLATOR  HAS  BEEN  WRITTEN  USING  THE  RECURSIVE       *
000004     12      *   DESCENT TECHNIQUE . IT ACCEPTS BCPL SOURCE STATEMENTS AND           *
000004     13      *   PRODUCES CDC COMPASS ASSEMBLER CODE TO BE COMPILED BY COMPASS       *
000004     14      *   AND EXECUTED . IT HAS BEEN PREPARED FOR THE FULFILLMENT OF          *
000004     15      *   MASTER DEGREE IN COMPUTER ENGENEERING DEPARTMENT BOGAZICI            *
000004     16      *   UNIVERSITY - 1984 .                                                 *
000004     17      *                                                                       *
000004     18      *   PREPARED BY  :  VASIL KADIFELI                                      *
000004     19      *                                                                       *
000004     20      *   THE TRANSLATOR CAN BE CALLED BY :                                   *
000004     21      *                                                                       *
000004     22      *   GET,BCPL/UN=LIBRARY.                                                *
000004     23      *   GET,BCPLLIB/UN=LIBRARY.                                             *
000004     24      *   ATTACH,RELOPL/UN=INS,PN=PACK1.                                      *
000004     25      *   BCPL,IN,LIST,CODE,LIB.   OR   BCPL(IN,LIST,CODE,LIB).               *
000004     26      *   REWIND,CODE.                                                        *
000004     27      *   COMPASS,I=CODE.                                                     *
000004     28      *   LGO.                                                                *
000004     29      *                                                                       *
000004     30      *   WHERE FILES :                                                       *
000004     31      *                                                                       *
000004     32      *      IN   : INPUT FILE OF BCPL SOURCE STATEMENTS (INPUT)              *
000004     33      *      LIST : TRANSLATOR LISTING FILE OF BCPL PROG.(OUTPUT)             *
000004     34      *      CODE : FILE FOR ASSEMBLER CODE GENERATED    (CODE)               *
000004     35      *      LIB  : BCPL LIBRARY FILE , NOT TO BE USED                        *
000004     36      *             BY THE USER.                         (BCPLLIB)           *
000004     37      *                                                                       *
000004     38      *************************************************************************)
000004     39
000004     40 PROGRAM BCPL (INPUT/,OUTPUT,CODE,BCPLLIB/) ;
000074     41
000074     42 LABEL 1111 , 2222 , 3333 ;
000074     43
000074     44 CONST
000074     45
000074     46   FIRSTCHAR = 'A' ; LASTCHAR = ';' ;
000074     47   NOOFKW = 50 ;
000074     48   SMAX = 20 ; ZER = 0 ;
000074     49   TITLE1 = '   SOURCE LISTING OF BCPL PROGRAM ' ;
000074     50   TITLE2 = '*** BCPL TRANSLATOR / LEVEL 1 ***'  ;
000074     51   TITLE3 = ' PAGE' ;
000074     52
000074     53 TYPE
000074     54
000074     55   CTP       =  ^ CONSTANT ;
000074     56   IDP       =  ^ IDENT ;
000074     57
000074     58   IDCLASS   = (     KONS,        LABL,        VARS,
000074     59                  PROCFUNC,       PROG                ) ;
000074     60
```

```
000074     61   CSTCLASS = (       INT,        STR                ) ;
000074     62
000074     63   CONSTANT  =  PACKED RECORD
000074     64                  NEXT  : CTP ;
000074     65                  ADDR  :  0..MAXINT  ;
000074     66                  CASE  KLASS  :  CSTCLASS  OF
000074     67                    INT  : ( VALU      :  PACKED ARRAY [1..20] OF CHAR ) ;
000074     68                    STR  : ( STRLEN    :  0..10 ;
000074     69                             STRVALU   :  PACKED ARRAY [1..10]
000074     70                                          OF CHAR  ) ;
000074     71               END ;
000074     72
000074     73
000074     74   IDENT  =  PACKED  RECORD
000074     75               NAME  :  PACKED ARRAY [1..10] OF CHAR ;
000074     76               ADDRESS : 0..MAXINT ;
000074     77               LLINK ,
000074     78               RLINK :  IDP      ;
000074     79               NEXT  :  IDP      ;
000074     80               CASE   KLASS  : IDCLASS  OF
000074     81                 KONS     :  ( VALU     :  CTP          ) ;
000074     82                 LABL     :  ( LADDR    :  0..MAXINT ;
000074     83                               DECLARED :  BOOLEAN      ) ;
000074     84                 VARS     :  ( VADDR    :  0..MAXINT ;
000074     85                               TAB      :  BOOLEAN   ;
000074     86                               VEC      :  BOOLEAN   ;
000074     87                               VALLINK  :  CTP       ;
000074     88                               NOOFELEM :  0.. MAXINT  ) ;
000074     89                 PROCFUNC:  ( P        :  BOOLEAN   ;
000074     90                               PFADDR   :  INTEGER   ;
000074     91                               PARNO    :  0..MAXINT ;
000074     92                               PPTR     :  IDP          ) ;
000074     93                 PROG     :  ( )
000074     94             END ;
000074     95
000074     96   SYMBOL =
000074     97          ( ADDOP     ,      FINISHSY ,      NOTSY     ,      SWITCHSY ,
000074     98            ADDROP    ,      FORSY    ,      NUMBER    ,      TABLESY  ,
000074     99            ANDSY     ,      GOTOSY   ,      ORSY      ,      TESTSY   ,
000074    100            ASGNOP    ,      ID       ,      OTHERSY   ,      THENSY   ,
000074    101            BESY      ,      IFSY     ,      RBRACK    ,      TOSY     ,
000074    102            BYSY      ,      INTOSY   ,      READSY    ,      UNLESSY  ,
000074    103            CASESY    ,      KONST    ,      RELOP     ,      UNTILSY  ,
000074    104            COLON     ,      LAND     ,      REPEATSY  ,      VALOFSY  ,
000074    105            COMMA     ,      LBRACK   ,      RESULTSY  ,      VECAP    ,
000074    106            HALTSY    ,      LETSY    ,      RETURNSY  ,      VECSY    ,
000074    107            CONDSY    ,      LOR      ,      RPARENT   ,      WHILESY  ,
000074    108            DEFAULTSY,       LPARENT  ,      SEMICOLON,       WRITESY  ,
000074    109            DOSY      ,      MANISY   ,      SHFOP     ,      COMPASS  ,
000074    110            ENDCASESY,       MULOP    ,      STATSY    ,
000074    111            EQVSY     ,      NEQVSY   ,      STRING                    ) ;
000074    112
000074    113   SETOFSYS = SET OF SYMBOL ;
000074    114
000074    115   OPERATOR =
000074    116          ( ASTR      ,      L        ,      MINUS     ,      RDIV     ,
000074    117            C         ,      LEOP     ,      N         ,      REM      ,
000074    118            EQOP      ,      LSHF     ,      NEOP      ,      RSHF     ,
000074    119            GEOP      ,      LTOP     ,      O         ,      RVOP     ,
000074    120            GTOP      ,      LVOP     ,      PLUS      ,      S          ) ;
```

```
000074   121
000074   122   TOKENCLASS = (  OPRTR,    ADDRESS,      OTHER                ) ;
000074   123
000074   124   TOKENS    =  PACKED RECORD
000074   125                   CLASS  :  SYMBOL
000074   126                   CASE  TOKENCLASS  OF
000074   127                     OPRTR   : ( OP : OPERATOR ) ;
000074   128                     ADDRESS : ( CSTADDR : CTP         ) ;
000074   129                     OTHER   : ()
000074   130                 END ;
000074   131
000074   132   SEARCHLEVEL = (  SALEV,  (* SEARCH ALL LEVELS          *)
000074   133                    SCLEV   (* SEARCH ONLY CURRENT LEVEL *)     ) ;
000074   134
000074   135   ENTERLEVEL  = (  ECLEV,  (* ENTER AT CURRENT LEVEL     *)
000074   136                    ENLEV   (* ENTER AT A NEW LEVEL       *)     ) ;
000074   137
000074   138   CHARSET = SET OF '$'..'9' ;
000074   139
000074   140   NUMARRAY  = PACKED ARRAY [1..20] OF CHAR ;
000074   141
000074   142 VAR
000074   143
000074   144   DIGITS,  (* DECIMAL DIGITS   *)
000074   145   ODIGITS, (* OCTAL   DIGITS   *)
000074   146   IDCHARS  (* ID CHARACTERS    *) : CHARSET ;
000077   147
000077   148   RESTSYMBOLS : PACKED ARRAY [ CHAR ] OF TOKENS ;
000177   149
000177   150   CTSTART , CTLAST  : CTP ;
000201   151   CLEVPTR , STSTART : IDP ;
000203   152   IDPTR,IDP1  : IDP ;
000205   153   IDSTR       : PACKED ARRAY [1..10] OF CHAR    ;
000206   154   DAYMESS     : PACKED ARRAY [1..24] OF CHAR    ;
000211   155   MAIN        : PACKED ARRAY [1..12] OF CHAR    ;
000213   156
000213   157   KW          : PACKED ARRAY [1..NOOFKW] OF
000213   158                   PACKED RECORD
000213   159                     NAME  :  PACKED ARRAY [1..10] OF CHAR ;
000213   160                     CLASS :  SYMBOL   ;
000213   161                     OP    :  OPERATOR ;
000213   162                   END ;
000357   163
000357   164   MAXINTEGER , MAXOCTAL : NUMARRAY ;
000363   165
000363   166   PROCADDR , FUNCADDR , CONSTADDR : INTEGER ;
000366   167   DSTSIZE , FSTSIZE , RSTSIZE , NSTSIZE , LIBSIZE : INTEGER ;
000373   168
000373   169   CH : CHAR ;
000374   170   CHCNT,CARDCNT,ERRINX,LINECNT,PAGECNT : INTEGER ;
000401   171   INBUF : PACKED ARRAY [1..80] OF CHAR ;
000411   172   SYMSTART : INTEGER ; ERRPOS : CHAR ;
000413   173
000413   174   RECOVERING,UNCLOSED,SKIPPING,OPTIMIZE,LISTING : BOOLEAN ;
000420   175   TOPNODE : BOOLEAN ;
000421   176
000421   177   TIME1,TIME2 : REAL ;
000423   178   DAT , TIM : ALFA ;
000425   179
000425   180   I,ERRCNT,TOTERR : INTEGER ;
```

```
000430   181   ERRBUF : PACKED ARRAY [1..80] OF CHAR ;
000440   182
000440   183   REPORTARR : PACKED ARRAY [SYMBOL]
000440   184                  OF PACKED ARRAY [1..10] OF CHAR ;
000532   185
000532   186   LSYS      : SETOFSYS ; (* SET OF LEGAL SYMBOLS TO SKIP *)
000533   187
000533   188   TOKEN : TOKENS ;
000534   189
000534   190   CODE    : TEXT ; (* FILE TO PLACE THE CODE GENERATED *)
000570   191   BCPLLIB : TEXT ; (* FILE WHERE BCPLLIB RESIDES *)
000624   192   LIB : PACKED ARRAY [1..6] OF BOOLEAN ;
000625   193
000625   194   CASECNT , LABCNT , JUMPCNT , LEVCNT , XTOP : INTEGER ;
000632   195   IBANK : INTEGER ; (* KEEPS THE NUMBER OF INSTRUCTION GENERATED *)
000633   196
000633   197 (*  F O R W A R D    D E C L A R A T I O N S  *)
000633   198
000633   199
000633   200 PROCEDURE GENCODE(LAB:INTEGER ; LABTYPE : CHAR ;
000004   201                   INSTR : INTEGER ; I,J,K : INTEGER ;
000010   202                   KK : INTEGER )                          ; FORWARD ;
000011   203 PROCEDURE GETREG                                          ; FORWARD ;
000002   204 PROCEDURE FREEREG                                         ; FORWARD ;
000002   205 PROCEDURE COMPUTELEVEL(L:INTEGER)                         ; FORWARD ;
000003   206 PROCEDURE LINKCONST(C:CTP;VAR C1:CTP)                     ; FORWARD ;
000004   207 PROCEDURE MUSTBE(S:SYMBOL)                                ; FORWARD ;
000003   208 PROCEDURE REPORT(S:SYMBOL)                                ; FORWARD ;
000003   209 FUNCTION  HAVE(S:SYMBOL) : BOOLEAN                       ; FORWARD ;
000004   210 PROCEDURE SKIP(LSYS : SETOFSYS)                          ; FORWARD ;
000003   211 PROCEDURE DECLARATION(VAR SUCDECLARATION : BOOLEAN)       ; FORWARD ;
000003   212 PROCEDURE DECLIST                                         ; FORWARD ;
000002   213 PROCEDURE VECTOR (P : IDP)                                ; FORWARD ;
000003   214 PROCEDURE TABLE (P : IDP)                                 ; FORWARD ;
000003   215 PROCEDURE CST ( P : IDP )                                 ; FORWARD ;
000003   216 PROCEDURE STATIC                                          ; FORWARD ;
000002   217 PROCEDURE MANIFEST                                        ; FORWARD ;
000002   218 PROCEDURE SDEF                                            ; FORWARD ;
000002   219 PROCEDURE COMMAND(LSYS : SETOFSYS)                        ; FORWARD ;
000003   220 PROCEDURE IFPROC(LSYS : SETOFSYS)                         ; FORWARD ;
000003   221 PROCEDURE WHILEPROC(LSYS : SETOFSYS)                      ; FORWARD ;
000003   222 PROCEDURE REPEATPROC(LSYS : SETOFSYS)                     ; FORWARD ;
000003   223 PROCEDURE UNLESSPROC(LSYS : SETOFSYS)                     ; FORWARD ;
000003   224 PROCEDURE TESTPROC(LSYS : SETOFSYS)                       ; FORWARD ;
000003   225 PROCEDURE RETURNPROC(LSYS : SETOFSYS)                     ; FORWARD ;
000003   226 PROCEDURE RESULTPROC(LSYS : SETOFSYS)                     ; FORWARD ;
000003   227 PROCEDURE EXPR(LSYS : SETOFSYS)                           ; FORWARD ;
000003   228 PROCEDURE EXPRO(LSYS : SETOFSYS)                          ; FORWARD ;
000003   229 PROCEDURE EXP1(LSYS : SETOFSYS)                           ; FORWARD ;
000003   230 PROCEDURE EXP2(LSYS : SETOFSYS)                           ; FORWARD ;
000003   231 PROCEDURE EXP3(LSYS : SETOFSYS)                           ; FORWARD ;
000003   232 PROCEDURE EXP4(LSYS : SETOFSYS)                           ; FORWARD ;
000003   233 PROCEDURE EXP5(LSYS : SETOFSYS)                           ; FORWARD ;
000003   234 PROCEDURE EXP6(LSYS : SETOFSYS)                           ; FORWARD ;
000003   235 PROCEDURE EXP7(LSYS : SETOFSYS)                           ; FORWARD ;
000003   236 PROCEDURE EXP8(LSYS : SETOFSYS)                           ; FORWARD ;
000003   237 PROCEDURE EXP9(LSYS : SETOFSYS)                           ; FORWARD ;
000003   238 PROCEDURE BLOCK(LSYS : SETOFSYS)                          ; FORWARD ;
000003   239 PROCEDURE SWITCHPROC(LSYS : SETOFSYS)                     ; FORWARD ;
000003   240 PROCEDURE FPL (LSYS : SETOFSYS)                           ; FORWARD ;
```

```
000003   241 PROCEDURE PROCBODY(LSYS : SETOFSYS ; C : CTP)        ; FORWARD ;
000004   242 PROCEDURE FUNCBODY(LSYS : SETOFSYS ; C : CTP)        ; FORWARD ;
000004   243 PROCEDURE ELIST(LSYS : SETOFSYS ; PNO : INTEGER)     ; FORWARD ;
000004   244 PROCEDURE FORPROC(LSYS : SETOFSYS)                   ; FORWARD ;
000003   245 PROCEDURE MDEF                                      ; FORWARD ;
000002   246 PROCEDURE ERRCR(I:INTEGER)                          ; FORWARD ;
000003   247 PROCEDURE SEARCHID(LEVEL:SEARCHLEVEL)               ; FORWARD ;
000003   248 FUNCTION  PREVLEVPTR( R : IDP ) : IDP               ; FORWARD ;
000004   249 PROCEDURE FREENODE( R : IDP )                       ; FORWARD ;
000003   250 PROCEDURE FREELEVEL                                 ; FORWARD ;
000002   251 PROCEDURE BUILDPARAM                                ; FORWARD ;
000002   252 PROCEDURE ENTERID (LEVEL : ENTERLEVEL )             ; FORWARD ;
000003   253 PROCEDURE READCHPROC                                ; FORWARD ;
000002   254 PROCEDURE WRITESTRPROC (LSYS : SETOFSYS )           ; FORWARD ;
000003   255 PROCEDURE WRITENUMPROC (LSYS : SETOFSYS )           ; FORWARD ;
000003   256 PROCEDURE WRITEOPROC   (LSYS : SETOFSYS )           ; FORWARD ;
000003   257 PROCEDURE DUMPCONST                                 ; FORWARD ;
000002   258 PROCEDURE ENTERLIB                                  ; FORWARD ;
000002   259 PROCEDURE INITVALU                                  ; FORWARD ;
000002   260
000002   261
000002   262 (*****************************************
000002   263  *                                       *
000002   264  *  THIS FUNCTION CONVERTS A STRING OF   *
000002   265  *  DECIMAL OR OCTAL DIGITS INTO THEIR   *
000002   266  *  EQUIVALENT INTEGER VALUE .           *
000002   267  *                                       *
000002   268  *****************************************)
000002   269
000002   270 FUNCTION CONV( A : NUMARRAY ) : INTEGER ;
000006   271
000006   272 VAR I,J,VAL,KAT : INTEGER ;
000012   273
000012   274 BEGIN (* CONV *)
000012   275   IF A[1]=' '
000012   276     THEN KAT:=10
000016   277     ELSE KAT:=8 ;
000022   278   I:=1 ;
000024   279   VAL:=0 ;
000025   280   WHILE A[I]=' ' DO I:=I+1 ;
000041   281   FOR J:=I TO 20 DO VAL:=VAL*KAT+ORD(A[J])-ORD('0') ;
000063   282   CONV:=VAL ;
000065   283 END ; (* CONV *)
000077   284
000077   285 (*********************************************************
000077   286  *                                                       *
000077   287  *  THIS PROCEDURE INITIALIZES TABLES AND VARIABLES      *
000077   288  *  TO BE USED BY THE TRANSLATOR . INITIALIZATION        *
000077   289  *  TIME HAS NOT BEEN INCLUDED INTO TRANSLATION TIME .   *
000077   290  *                                                       *
000077   291  *********************************************************)
000077   292
000077   293 PROCEDURE INITIALIZE;
000002   294
000002   295 VAR I: INTEGER ;
000003   296
000003   297 BEGIN
000003   298
000003   299   NEW(IDPTR) ;                (* PREPARE SYMBOL TABLE HEADER *)
000006   300   IDPTR ^. KLASS := PROG ;
```

```
000016   301   IDPTR^.ADDRESS := 0 ;
000022   302   IDPTR ^. LLINK := NIL ;
000027   303   IDPTR ^. RLINK := NIL ;
000034   304   IDPTR ^. NEXT  := NIL ;
000042   305   IDPTR ^. NAME  := '          ' ;
000047   306   STSTART        := IDPTR ;
000051   307   CLEVPTR        := STSTART ;
000052   308   IDPTR          := NIL   ;
000053   309
000053   310   NEW(CTSTART) ;                (* PREPARE CONSTANTS TABLE HEADER *)
000055   311   WITH CTSTART ^ DO
000062   312     BEGIN
000062   313       NEXT:=NIL ;
000064   314       ADDR:=0   ;
000066   315       KLASS:=INT;
000070   316       VALU:='7777777777777777777' ;
000073   317     END ;
000073   318   CTLAST         := CTSTART ;
000074   319
000074   320   (* INITIALIZE BUFFERS AND VARIABLES *)
000074   321
000074   322   FOR I := 1 TO 80 DO ERRBUF[I] := ' ' ;
000116   323
000116   324   PROCADDR := 0 ;
000117   325   FUNCADDR := 0 ;
000120   326   CONSTADDR:= 0 ;
000121   327   LABCNT   := 0 ;
000123   328   CASECNT  := 0 ;
000124   329   JUMPCNT  := 0 ;
000125   330   XTOP     := 0 ;
000127   331   IBANK    := 0 ;
000130   332   DSTSIZE  := 3072 ;
000131   333   FSTSIZE  :=   48 ;
000133   334   RSTSIZE  :=  128 ;
000134   335   NSTSIZE  :=  128 ;
000135   336   LIBSIZE  :=    0 ;
000136   337   LISTING  := TRUE ;
000140   338
000140   339   REWRITE(CODE) ;        (* OPEN CODE FILE AND LIBRARY FILE *)
000141   340   RESET(BCPLLIB);
000143   341
000143   342   DATE(DAT) ;
000145   343   TIME(TIM) ;
000147   344
000147   345   DAYMESS := 'TRANSLATING            ' ;
000153   346   MAIN    := 'MAIN PROGRAM' ;
000156   347
000156   348   DIGITS := [ '0'..'9' ] ;              (* DECIMAL DIGITS SET       *)
000157   349   ODIGITS:= [ '0'..'7' ] ;              (* OCTAL   DIGITS SET       *)
000161   350   IDCHARS:= DIGITS + [ 'A'..'Z' ] ;     (* IDENTIFIER CHARCTERS SET *)
000164   351
000164   352   MAXINTEGER := '  576460752303423487' ; (* MAX DECIMAL INTEGER      *)
000167   353   MAXOCTAL   := '7777777777777777777' ; (* MAX OCTAL   INTEGER      *)
000172   354
000172   355   CHCNT :=89 ;
000173   356   ERRCNT := 0 ;
000175   357   CARDCNT := 0 ;
000176   358   ERRINX := 0 ;
000177   359   LINECNT := 0 ;
000200   360   PAGECNT := 1 ;
```

```
000201   361   ERRPOS  := '0' ;
000202   362   TOTERR := 0 ;
000204   363
000204   364   SKIPPING       := FALSE ;
000205   365   RECOVERING     := FALSE ;
000207   366   UNCLOSED       := FALSE ;
000210   367   OPTIMIZE       := TRUE  ;
000211   368
000211   369   FOR I:=1 TO 6 DO LIB[I]:=FALSE ;
000225   370
000225   371   (* SOME SINGLE CHARACTER SYMBOLS OF THE BCPL LANGUAGE *)
000225   372
000225   373   CH     := FIRSTCHAR ;
000226   374   REPEAT
000226   375     WITH RESTSYMBOLS [CH] DO CLASS := OTHERSY ;
000234   376     CH := SUCC(CH) ;
000237   377   UNTIL CH = LASTCHAR ;
000240   378   RESTSYMBOLS [CH] .CLASS := OTHERSY ;
000244   379
000244   380   RESTSYMBOLS [ ';' ] .CLASS := SEMICOLON ;
000247   381   RESTSYMBOLS [ '(' ] .CLASS := LPARENT   ;
000252   382   RESTSYMBOLS [ ')' ] .CLASS := RPARENT   ;
000255   383   RESTSYMBOLS [ '[' ] .CLASS := LBRACK    ;
000260   384   RESTSYMBOLS [ ']' ] .CLASS := RBRACK    ;
000262   385   RESTSYMBOLS [ ',' ] .CLASS := COMMA     ;
000265   386   WITH RESTSYMBOLS [ '!' ] DO
000265   387     BEGIN CLASS := VECAP   ;              END;
000270   388   WITH RESTSYMBOLS [ '*' ] DO
000270   389     BEGIN CLASS := MULOP   ; OP := ASTR   END;
000275   390   WITH RESTSYMBOLS [ '+' ] DO
000275   391     BEGIN CLASS := ADDOP   ; OP := PLUS   END;
000301   392   WITH RESTSYMBOLS [ '=' ] DO
000301   393     BEGIN CLASS := RELOP   ; OP := EQOP   END;
000306   394
000306   395   (* CREATE KEYWORDS TABLE *)
000306   396
000306   397   WITH KW [ 1] DO
000306   398     BEGIN
000306   399       NAME := 'AND       ' ; CLASS := ANDSY       ;
000312   400     END ;
000312   401   WITH KW [ 2] DO
000312   402     BEGIN
000312   403       NAME := 'BE        ' ; CLASS := BESY        ;
000315   404     END ;
000315   405   WITH KW [ 3] DO
000315   406     BEGIN
000315   407       NAME := 'BY        ' ; CLASS := BYSY        ;
000321   408     END ;
000321   409   WITH KW [ 4] DO
000321   410     BEGIN
000321   411       NAME := 'CASE      ' ; CLASS := CASESY      ;
000325   412     END ;
000325   413   WITH KW [ 5] DO
000325   414     BEGIN
000325   415       NAME := 'COMPASS   ' ; CLASS := COMPASS     ;
000331   416     END ;
000331   417   WITH KW [ 6] DO
000331   418     BEGIN
000331   419       NAME := 'COND      ' ; CLASS := CONDSY      ;
000335   420     END ;
```

```
000335   421   WITH KW [ 7] DO
000335   422     BEGIN
000335   423       NAME := 'DEFAULT    ' ; CLASS := DEFAULTSY    ;
000341   424     END ;
000341   425   WITH KW [ 8] DO
000341   426     BEGIN
000341   427       NAME := 'DO         ' ; CLASS := DOSY         ;
000345   428     END ;
000345   429   WITH KW [ 9] DO
000345   430     BEGIN
000345   431       NAME := 'EQV        ' ; CLASS := EQVSY        ;
000351   432     END ;
000351   433   WITH KW [10] DO
000351   434     BEGIN
000351   435       NAME := 'ENDCASE    ' ; CLASS := ENDCASESY    ;
000355   436     END ;
000355   437   WITH KW [11] DO
000355   438     BEGIN
000355   439       NAME := 'FALSE      ' ; CLASS := NUMBER       ;
000361   440     END ;
000361   441   WITH KW [12] DO
000361   442     BEGIN
000361   443       NAME := 'FINISH     ' ; CLASS := FINISHSY     ;
000364   444     END ;
000364   445   WITH KW [13] DO
000364   446     BEGIN
000364   447       NAME := 'FOR        ' ; CLASS := FORSY        ;
000370   448     END ;
000370   449   WITH KW [14] DO
000370   450     BEGIN
000370   451       NAME := 'GOTO       ' ; CLASS := GOTOSY       ;
000374   452     END ;
000374   453   WITH KW [15] DO
000374   454     BEGIN
000374   455       NAME := 'HALT       ' ; CLASS := HALTSY       ;
000400   456     END ;
000400   457   WITH KW [16] DO
000400   458     BEGIN
000400   459       NAME := 'IF         ' ; CLASS := IFSY         ;
000404   460     END ;
000404   461   WITH KW [17] DO
000404   462     BEGIN
000404   463       NAME := 'INTO       ' ; CLASS := INTOSY       ;
000410   464     END ;
000410   465   WITH KW [18] DO
000410   466     BEGIN
000410   467       NAME := 'LAND       ' ; CLASS := LAND         ;
000414   468     END ;
000414   469   WITH KW [19] DO
000414   470     BEGIN
000414   471       NAME := 'LET        ' ; CLASS := LETSY        ;
000420   472     END ;
000420   473   WITH KW [20] DO
000420   474     BEGIN
000420   475       NAME := 'LOR        ' ; CLASS := LOR          ;
000424   476     END ;
000424   477   WITH KW [21] DO
000424   478     BEGIN
000424   479       NAME := 'LSHIFT     ' ; CLASS := SHFOP        ; OP := LSHF   ;
000432   480     END ;
```

```
000432   481   WITH KW [22] DO
000432   482     BEGIN
000432   483       NAME := 'LV        ' ; CLASS := ADDROP      ; OP := LVOP   ;
000440   484     END ;
000440   485   WITH KW [23] DO
000440   486     BEGIN
000440   487       NAME := 'MANIFEST  ' ; CLASS := MANISY      ;
000444   488     END ;
000444   489   WITH KW [24] DO
000444   490     BEGIN
000444   491       NAME := 'NEQV      ' ; CLASS := NEQVSY      ;
000450   492     END ;
000450   493   WITH KW [25] DO
000450   494     BEGIN
000450   495       NAME := 'NOT       ' ; CLASS := NOTSY       ;
000454   496     END ;
000454   497   WITH KW [26] DO
000454   498     BEGIN
000454   499       NAME := 'OR        ' ; CLASS := ORSY        ;
000460   500     END ;
000460   501   WITH KW [27] DO
000460   502     BEGIN
000460   503       NAME := 'READCH    ' ; CLASS := READSY      ; OP := C      ;
000466   504     END ;
000466   505   WITH KW [28] DO
000466   506     BEGIN
000466   507       NAME := 'READLN    ' ; CLASS := READSY      ; OP := L      ;
000473   508     END ;
000473   509   WITH KW [29] DO
000473   510     BEGIN
000473   511       NAME := 'REM       ' ; CLASS := MULOP       ; OP := REM    ;
000501   512     END ;
000501   513   WITH KW [30] DO
000501   514     BEGIN
000501   515       NAME := 'REPEAT    ' ; CLASS := REPEATSY    ;
000505   516     END ;
000505   517   WITH KW [31] DO
000505   518     BEGIN
000505   519       NAME := 'RESULTIS  ' ; CLASS := RESULTSY    ;
000511   520     END ;
000511   521   WITH KW [32] DO
000511   522     BEGIN
000511   523       NAME := 'RETURN    ' ; CLASS := RETURNSY    ;
000515   524     END ;
000515   525   WITH KW [33] DO
000515   526     BEGIN
000515   527       NAME := 'RSHIFT    ' ; CLASS := SHFOP       ; OP := RSHF   ;
000523   528     END ;
000523   529   WITH KW [34] DO
000523   530     BEGIN
000523   531       NAME := 'RV        ' ; CLASS := ADDROP      ; OP := RVOP   ;
000531   532     END ;
000531   533   WITH KW [35] DO
000531   534     BEGIN
000531   535       NAME := 'STATIC    ' ; CLASS := STATSY      ;
000535   536     END ;
000535   537   WITH KW [36] DO
000535   538     BEGIN
000535   539       NAME := 'SWITCHON  ' ; CLASS := SWITCHSY    ;
000541   540     END ;
```

```
000541   541   WITH KW [37] DO
000541   542     BEGIN
000541   543       NAME := 'TABLE     ' ; CLASS := TABLESY     ;
000545   544     END ;
000545   545   WITH KW [38] DO
000545   546     BEGIN
000545   547       NAME := 'TEST      ' ; CLASS := TESTSY      ;
000551   548     END ;
000551   549   WITH KW [39] DO
000551   550     BEGIN
000551   551       NAME := 'THEN      ' ; CLASS := THENSY      ;
000555   552     END ;
000555   553   WITH KW [40] DO
000555   554     BEGIN
000555   555       NAME := 'TO        ' ; CLASS := TOSY        ;
000561   556     END ;
000561   557   WITH KW [41] DO
000561   558     BEGIN
000561   559       NAME := 'TRUE      ' ; CLASS := NUMBER      ;
000565   560     END ;
000565   561   WITH KW [42] DO
000565   562     BEGIN
000565   563       NAME := 'UNLESS    ' ; CLASS := UNLESSY     ;
000571   564     END ;
000571   565   WITH KW [43] DO
000571   566     BEGIN
000571   567       NAME := 'UNTIL     ' ; CLASS := UNTILSY     ;
000575   568     END ;
000575   569   WITH KW [44] DO
000575   570     BEGIN
000575   571       NAME := 'VALOF     ' ; CLASS := VALOFSY     ;
000601   572     END ;
000601   573   WITH KW [45] DO
000601   574     BEGIN
000601   575       NAME := 'VEC       ' ; CLASS := VECSY       ;
000605   576     END ;
000605   577   WITH KW [46] DO
000605   578     BEGIN
000605   579       NAME := 'WHILE     ' ; CLASS := WHILESY     ;
000611   580     END ;
000611   581   WITH KW [47] DO
000611   582     BEGIN
000611   583       NAME := 'WRITELN   ' ; CLASS := WRITESY     ; OP := L     ;
000617   584     END ;
000617   585   WITH KW [48] DO
000617   586     BEGIN
000617   587       NAME := 'WRITEN    ' ; CLASS := WRITESY     ; OP := N     ;
000625   588     END ;
000625   589   WITH KW [49] DO
000625   590     BEGIN
000625   591       NAME := 'WRITEO    ' ; CLASS := WRITESY     ; OP := O     ;
000633   592     END ;
000633   593   WITH KW [50] DO
000633   594     BEGIN
000633   595       NAME := 'WRITESTR  ' ; CLASS := WRITESY     ; OP := S     ;
000641   596     END ;
000641   597
000641   598   (* ERROR REPORTING ARRAY TABLE *)
000641   599
000641   600
```

```
000641    601    REPORTARR [COMMA      ] := '     ,      ' ;
000643    602    REPORTARR [DOSY       ] := '    DO      ' ;
000645    603    REPORTARR [TOSY       ] := '    TO      ' ;
000647    604    REPORTARR [BYSY       ] := '    BY      ' ;
000650    605    REPORTARR [SEMICOLON  ] := '     ;      ' ;
000652    606    REPORTARR [UNLESSY    ] := '  UNLESS    ' ;
000654    607    REPORTARR [FINISHSY   ] := '  FINISH    ' ;
000656    608    REPORTARR [FORSY      ] := '   FOR      ' ;
000660    609    REPORTARR [UNTILSY    ] := '   UNTIL    ' ;
000662    610    REPORTARR [COLON      ] := '     :      ' ;
000664    611    REPORTARR [GOTOSY     ] := '   GOTO     ' ;
000666    612    REPORTARR [VALOFSY    ] := '   VALOF    ' ;
000670    613    REPORTARR [ASGNOP     ] := '     :=     ' ;
000672    614    REPORTARR [IFSY       ] := '     IF     ' ;
000674    615    REPORTARR [VECSY      ] := '    VEC     ' ;
000676    616    REPORTARR [STATSY     ] := '  STATIC    ' ;
000700    617    REPORTARR [INTOSY     ] := '   INTO     ' ;
000702    618    REPORTARR [ENDCASESY  ] := ' ENDCASE    ' ;
000704    619    REPORTARR [MANISY     ] := ' MANIFEST   ' ;
000706    620    REPORTARR [LETSY      ] := '    LET     ' ;
000710    621    REPORTARR [EQVSY      ] := '    EQV     ' ;
000712    622    REPORTARR [REPEATSY   ] := '  REPEAT    ' ;
000714    623    REPORTARR [HALTSY     ] := '   HALT     ' ;
000716    624    REPORTARR [LAND       ] := '    LAND    ' ;
000717    625    REPORTARR [NOTSY      ] := '    NOT     ' ;
000721    626    REPORTARR [NEQVSY     ] := '   NEQV     ' ;
000723    627    REPORTARR [LOR        ] := '    LOR     ' ;
000725    628    REPORTARR [VECAP      ] := '     !      ' ;
000727    629    REPORTARR [CRSY       ] := '     OR     ' ;
000731    630    REPORTARR [ANDSY      ] := '    AND     ' ;
000733    631    REPORTARR [RESULTSY   ] := ' RESULTIS   ' ;
000735    632    REPORTARR [READSY     ] := '  READ..    ' ;
000737    633    REPORTARR [BESY       ] := '    BE      ' ;
000741    634    REPORTARR [RETURNSY   ] := '  RETURN    ' ;
000743    635    REPORTARR [LPARENT    ] := '     (      ' ;
000745    636    REPORTARR [SWITCHSY   ] := ' SWITCHON   ' ;
000747    637    REPORTARR [RPARENT    ] := '     )      ' ;
000751    638    REPORTARR [CASESY     ] := '   CASE     ' ;
000753    639    REPORTARR [TABLESY    ] := '   TABLE    ' ;
000755    640    REPORTARR [WRITESY    ] := ' WRITE...   ' ;
000757    641    REPORTARR [LBRACK     ] := '     [      ' ;
000761    642    REPORTARR [CONDSY     ] := '     ->     ' ;
000763    643    REPORTARR [TESTSY     ] := '   TEST     ' ;
000765    644    REPORTARR [RBRACK     ] := '     ]      ' ;
000767    645    REPORTARR [DEFAULTSY  ] := '  DEFAULT   ' ;
000771    646    REPORTARR [THENSY     ] := '   THEN     ' ;
000773    647    REPORTARR [ID         ] := 'IDENTIFIER' ;
000775    648    REPORTARR [WHILESY    ] := '   WHILE    ' ;
000777    649    REPORTARR [ADDOP      ] := '  + OR -    ' ;
001001    650    REPORTARR [ADDROP     ] := ' LV OR RV   ' ;
001003    651    REPORTARR [KONST      ] := ' CONSTANT   ' ;
001005    652    REPORTARR [MULOP      ] := ' * / REM    ' ;
001007    653    REPORTARR [NUMBER     ] := '  NUMBER    ' ;
001011    654    REPORTARR [OTHERSY    ] := '??????????' ;
001013    655    REPORTARR [RELOP      ] := '  RELOP     ' ;
001015    656    REPORTARR [SHFOP      ] := ' L/R SHIFT' ;
001017    657    REPORTARR [STRING     ] := '  STRING    ' ;
001021    658
001021    659    CH := ' ' ;
001023    660 END ; (* INITIALIZE *)
```

```
001214   661
001214   662 (*************************************************************************
001214   663  *                                                                       *
001214   664  *  THIS ROUTINE ENTERS A CONSTANT INTO THE CONSTANTS TABLE IF IT        *
001214   665  *  HAS NOT BEEN ENTERED YET . THE CONSTANTS TABLE HAS BEEN ORGANIZED    *
001214   666  *  AS A LINKED LIST , THUS ENTRIES ARE MADE TO THE END OF THE TABLE .   *
001214   667  *  IF THE CONSTANT HAS ALLREADY BEEN ENTERED THEN FINDS ITS ENTRY .     *
001214   668  *  THE RETURNED VALUE IS A POINTER TO THAT ENTRY .                      *
001214   669  *                                                                       *
001214   670  *************************************************************************)
001214   671
001214   672 PROCEDURE LINKCONST ;
000004   673
000004   674 VAR P,Q : CTP ; FOUND : BOOLEAN ; KL : CSTCLASS ;
000010   675
000010   676 BEGIN (* LINKCONST *)
000010   677   KL:=C ^. KLASS ;
000013   678   P := CTSTART ;
000015   679   Q := P ;
000016   680   FOUND := FALSE ;
000017   681   WHILE (P<>NIL) AND (NOT FOUND) DO
000023   682     BEGIN
000023   683       Q:=P ;
000024   684       IF Q ^. KLASS = KL
000030   685         THEN BEGIN
000032   686                CASE KL OF
000033   687                  INT : IF Q ^. VALU = C ^. VALU
000043   688                          THEN FOUND:=TRUE ;
000053   689                  STR : IF Q ^. STRLEN = C ^. STRLEN
000063   690                          THEN IF Q ^. STRVALU = C ^. STRVALU
000075   691                                  THEN FOUND:=TRUE ;
000101   692                  OTHERWISE ERROR(49) ;
000107   693                END ; (* CASE *)
000107   694              END ;
000107   695       P:=Q ^. NEXT ;
000114   696     END ;
000115   697   IF FOUND
000115   698     THEN BEGIN
000116   699            C1:=Q ;
000120   700            DISPOSE(C) ;
000122   701          END
000122   702     ELSE BEGIN
000123   703            C ^. NEXT := NIL ;
000130   704            CTLAST ^. NEXT := C ;
000135   705            CONSTADDR:=CONSTADDR+1 ;
000137   706            C ^. ADDR := CONSTADDR ;
000147   707            CTLAST := C ;
000150   708            C1 := C ;
000151   709          END ;
000151   710 END ; (* LINKCONST *)
000167   711
000167   712   (* THIS PROCEDURE PRINTS THE TITLE LINE ON EACH PAGE *)
000167   713   (* OF THE SYMBOLIC LISTING OF THE BCPL PROGRAM       *)
000167   714
000167   715 PROCEDURE PRINTTITLE ;
000002   716
000002   717 BEGIN (* PRINTTITLE *)
000002   718   WRITELN('1') ;
000012   719   WRITELN ;
000014   720   WRITELN(TITLE1,' ':17,TITLE2,' ':17,DAT,TIM,TITLE3,PAGECNT:3) ;
```

```
000062   721   WRITELN ;
000064   722   WRITELN ;
000066   723   LINECNT := 0 ;
000070   724   PAGECNT := PAGECNT+1 ;
000071   725 END ; (* PRINTTITLE *)
000104   726
000104   727 (*************************************************************************
000104   728  *                                                                       *
000104   729  *  THIS IS THE LEXICAL ANALYZER ROUTINE FOR THE BCPL LANGUAGE .  *
000104   730  *  IT HAS BEEN ORGANIZED IN A FORM VERY CLOSE TO A STATE TABLE . *
000104   731  *                                                                       *
000104   732  *************************************************************************)
000104   733
000104   734 PROCEDURE SCANNER ;
000002   735
000002   736 LABEL 1,4,5,6 ;
000002   737 VAR   FINISHED , ERR , DIG : BOOLEAN ;
000005   738       I , LGTH , J , K     : INTEGER ;
000011   739       STRBUF               : PACKED ARRAY [1..10] OF CHAR ;
000012   740       P                    : CTP         ;
000013   741       VAL                  : PACKED ARRAY [1..20] OF CHAR ;
000015   742
000015   743   (*  PRINTS ERROR MARKERS BELOW THE ERRONEOUS LINE *)
000015   744
000015   745   PROCEDURE PRINTERRORS ;
000002   746
000002   747   VAR I : INTEGER ;
000003   748
000003   749   BEGIN (* PRINTERRORS *)
000003   750     WRITE(' ':16) ;
000010   751     FOR I := 1 TO 80 DO
000012   752       BEGIN
000014   753         WRITE(ERRBUF[I]) ;
000033   754         ERRBUF[I]:=' '   ;
000047   755       END ;
000053   756     WRITELN ;
000054   757     TOTERR := TOTERR + ERRCNT ;
000056   758     ERRCNT := 0 ;
000057   759     LINECNT := LINECNT+1 ;
000061   760     IF LINECNT=60
000061   761       THEN PRINTTITLE ;
000064   762     ERRPOS := '0' ;
000066   763   END ; (* PRINTERRORS *)
000072   764
000072   765   (* PRINTS THE LINE THAT HAS JUST BEEN TRANSLATED *)
000072   766
000072   767   PROCEDURE PRINTLINE ;
000002   768
000002   769   VAR I : INTEGER ;
000003   770
000003   771   BEGIN (* PRINTLINE *)
000003   772     WRITE('      ',CARDCNT:5,'     ') ;
000022   773     FOR I := 1 TO 80 DO WRITE(INBUF[I]) ;
000050   774     WRITELN ;
000051   775     LINECNT:=LINECNT+1 ;
000053   776     IF LINECNT=60 THEN PRINTTITLE ;
000056   777   END ; (* PRINTLINE *)
000064   778
000064   779   (* READS A NEW LINE , MAKES THE PREVIOUS LINE AND ITS *)
000064   780   (* ERROR MARKERS LINE BE PRINTED IF ANY EXIST         *)
```

```
000064   781
000064   782   PROCEDURE READLINE ;
000002   783
000002   784   VAR I,BUFLEN,SIZE : INTEGER ;
000005   785
000005   786   BEGIN (* READLINE *)
000005   787     CHCNT := 0 ;
000006   788     IF (CARDCNT>0) AND LISTING THEN PRINTLINE ;
000012   789     IF (ERRCNT>0) AND (NOT LISTING) THEN PRINTLINE ;
000017   790     IF ERRCNT>0 THEN PRINTERRORS ;
000022   791     READLN ;
000024   792     IF EOF
000024   793       THEN IF CARDCNT = 0
000026   794              THEN GOTO 1111
000032   795              ELSE GOTO 2222
000035   796       ELSE BEGIN
000036   797              BUFLEN := 0 ;
000040   798              WHILE (NOT EOLN) AND (BUFLEN < 80) DO
000043   799                BEGIN
000043   800                  BUFLEN := BUFLEN+1 ;
000045   801                  READ(INBUF[BUFLEN])  ;
000062   802                END ;
000063   803              I := BUFLEN+1 ;
000065   804              WHILE I <= 80 DO
000070   805                BEGIN
000070   806                  INBUF[I] := ' ' ;
000102   807                  I := I + 1 ;
000104   808                END ;
000105   809              WRITE(CODE,'*') ;
000112   810              I := 1 ;
000114   811              WHILE I <= 80 DO
000117   812                BEGIN
000117   813                  IF INBUF[I] <> ' '
000125   814                    THEN WRITE(CODE,INBUF[I]) ;
000146   815                  I := I + 1 ;
000150   816                END ;
000151   817              WRITELN(CODE) ;
000153   818            END ;
000153   819     IF INBUF[1]='#'
000153   820       THEN BEGIN
000156   821              IF (INBUF[2]='P') AND LISTING
000160   822                THEN PRINTTITLE ;
000163   823              IF INBUF[2] IN ['S','F','R','N','L']
000163   824                THEN
000167   825                  IF INBUF[2]<>'L'
000167   826                    THEN BEGIN
000171   827                           I:=2 ;
000173   828                           REPEAT
000173   829                             I:=I+1 ;
000175   830                           UNTIL (INBUF[I] IN DIGITS) OR (I=80) ;
000211   831                           SIZE:=0 ;
000212   832                           WHILE (I<=80) AND (INBUF[I] IN DIGITS) DO
000226   833                             BEGIN
000226   834                               SIZE:=SIZE*10+ORD(INBUF[I])-ORD('0')    ;
000241   835                               I:=I+1 ;
000243   836                             END ;
000244   837                           IF SIZE>0
000244   838                             THEN BEGIN
000246   839                                    IF INBUF[2]='S' THEN DSTSIZE:=SIZE ;
000253   840                                    IF INBUF[2]='F' THEN FSTSIZE:=SIZE ;
```

```
000260   841                                    IF INBUF[2]='R' THEN RSTSIZE:=SIZE ;
000265   842                                    IF INBUF[2]='N' THEN NSTSIZE:=SIZE ;
000272   843                                  END
000272   844                            END
000272   845                       ELSE IF (INBUF[3]='-')
000275   846                               THEN LISTING := FALSE
000276   847                               ELSE IF (INBUF[3]='+')
000302   848                                       THEN LISTING := TRUE ;
000304   849               CH:=' ' ;
000306   850               CHCNT:=100 ;
000307   851               CARDCNT:=CARDCNT+1 ;
000311   852             END
000311   853       ELSE BEGIN
000312   854               CARDCNT:=CARDCNT+1 ;
000314   855               GENCODE(-1,' ',610,5,0,0,CARDCNT) ;
000323   856             END ;
000323   857   END ; (* READLINE *)
000334   858
000334   859   (* SUPPLIES SCANNER WITH THE NEXT CHARACTER OF THE INPUT STREAM *)
000334   860
000334   861   FUNCTION NEXTCHAR : CHAR ;
000003   862
000003   863   BEGIN (* NEXTCHAR *)
000003   864     IF CHCNT >= 80
000006   865       THEN NEXTCHAR := ' '
000011   866       ELSE BEGIN
000013   867               CHCNT := CHCNT+1 ;
000015   868               NEXTCHAR := INBUF[CHCNT] ;
000027   869             END ;
000027   870   END ; (* NEXTCHAR *)
000032   871
000032   872   (*  THIS ROUTINE SEARCHES KEYWORDS TABLE VIA BINARY SEARCH *)
000032   873   (*  TO FIND WHETHER A NAME ENCOUNTERED IS A KEYWORD OR NOT *)
000032   874
000032   875   PROCEDURE RESERVED ;
000002   876
000002   877   VAR LOW,HIGH,MID : INTEGER ;
000005   878       FOUND        : BOOLEAN ;
000006   879       P            : CTP       ;
000007   880
000007   881   BEGIN (* RESERVED *)
000007   882     LOW := 1 ;
000006   883     HIGH := NOOFKW ;
000007   884     FOUND := FALSE ;
000011   885     WHILE (LOW <= HIGH) AND (NOT FOUND) DO  (* BINARY SEARCH *)
000015   886       BEGIN
000015   887         MID := (LOW+HIGH) DIV 2 ;
000017   888         IF IDSTR = KW[MID].NAME
000022   889           THEN FOUND := TRUE
000024   890           ELSE IF IDSTR < KW[MID].NAME
000030   891                  THEN HIGH := MID-1
000033   892                  ELSE LOW  := MID+1
000035   893       END ;
000040   894     IF FOUND
000040   895       THEN BEGIN
000041   896               IF KW[MID].CLASS = NUMBER
000044   897                 THEN BEGIN
000047   898                        TOKEN.CLASS := NUMBER ;
000051   899                        IF KW[MID].NAME = 'TRUE      '
000054   900                          THEN TOKEN.CSTADDR := CTSTART
```

```
000056   901                          ELSE BEGIN
000061   902                                 NEW(P) ;
000063   903                                 P ^. KLASS := INT ;
000072   904                                 P ^. VALU  := '                    0' ;
000100   905                                 IF (NOT RECOVERING) AND OPTIMIZE
000101   906                                   THEN LINKCONST(P,P) ;
000105   907                                 TOKEN.CSTADDR := P ;
000110   908                               END ;
000110   909                       END
000110   910                   ELSE BEGIN
000111   911                          TOKEN.CLASS := KW[MID].CLASS ;
000120   912                          CASE TOKEN.CLASS OF
000121   913                            READSY,
000121   914                            WRITESY,
000121   915                            MULOP,
000121   916                            ADDOP,
000121   917                            SHFOP,
000121   918                            ADDROP : TOKEN.OP := KW[MID].OP ;
000131   919                            OTHERWISE ;
000167   920                          END ;
000167   921                       END
000167   922              END
000167   923        ELSE TOKEN.CLASS := ID ;
000173   924    END ; (* RESERVED *)
000212   925
000212   926  BEGIN (* SCANNER *)
000212   927    REPEAT
000004   928      WHILE CH = ' ' DO
000007   929        BEGIN
000007   930          IF CHCNT >= 80
000007   931            THEN READLINE
000011   932            ELSE BEGIN
000014   933                   CHCNT:=CHCNT+1 ;
000016   934                   CH:=INBUF[CHCNT] ;
000030   935                 END ;
000030   936        END ;
000031   937
000031   938      SYMSTART := CHCNT ;
000033   939      FINISHED := TRUE  ;
000035   940
000035   941      CASE CH OF
000036   942
000036   943      'A','B','C','D','E','F','G','H','I','J', (* IDENTIFIER OR KEYWORD *)
000036   944      'K','L','M','N','O','P','Q','R','S','T',
000036   945      'U','V','W','X','Y','Z' :
000036   946
000036   947        BEGIN
000036   948          I := 1 ;
000040   949          IDSTR[I] := CH ;
000050   950          CH := NEXTCHAR ;
000055   951          DIG := FALSE   ;
000056   952          WHILE (CH IN IDCHARS) DO
000061   953            BEGIN
000061   954              IF I < 10
000061   955                THEN BEGIN
000064   956                       I := I+1 ;
000065   957                       IF CH IN DIGITS THEN DIG := TRUE ;
000071   958                       IDSTR[I] := CH ;
000102   959                     END ;
000102   960              CH := NEXTCHAR ;
```

```
000107   961          END ;
000110   962        IF I < 10 THEN FOR I := I+1 TO 10 DO IDSTR[I] := ' ' ;
000127   963        IF NOT DIG
000127   964          THEN BEGIN
000131   965                 RESERVED ;
000133   966                 IF TOKEN.CLASS=COMPASS
000133   967                   THEN BEGIN
000137   968                          WHILE CHCNT<80 DO
000142   969                            BEGIN
000142   970                              CHCNT:=CHCNT+1 ;
000143   971                              CH:=INBUF[CHCNT] ;
000156   972                              WRITE(CODE,CH) ;
000163   973                            END ;
000164   974                          WRITELN(CODE) ;
000166   975                          CH:=' ';
000170   976                          FINISHED:=FALSE ;
000171   977                        END
000171   978               END
000171   979          ELSE TOKEN.CLASS := ID ;
000175   980      END ;
000176   981
000176   982    '0','1','2','3','4','5','6','7','8','9' : (* INTEGER CONSTANT *)
000176   983
000176   984      BEGIN
000176   985        VAL := '                    ' ;
000201   986        I := 1 ;
000202   987        REPEAT
000202   988          VAL[I] := CH ;
000220   989          CH := NEXTCHAR ;
000224   990          I := I+1 ;
000226   991        UNTIL ( NOT (CH IN DIGITS)) OR ( I>18) ;
000231   992        IF CH IN DIGITS
000231   993          THEN BEGIN
000233   994                 ERROR(23) ;
000235   995                 REPEAT
000235   996                   CH:=NEXTCHAR ;
000242   997                 UNTIL NOT(CH IN DIGITS) ;
000244   998                 VAL := MAXINTEGER ;
000247   999               END
000247  1000          ELSE BEGIN
000250  1001                 J := I-1 ;
000252  1002                 K := 20 ;
000253  1003                 WHILE J>0 DO
000255  1004                   BEGIN
000255  1005                     VAL[K] := VAL[J] ;
000302  1006                     J := J-1 ;
000303  1007                     K := K-1 ;
000304  1008                   END ;
000305  1009                 FOR I:=1 TO K DO VAL[I]:=' ' ;
000327  1010                 I:=3 ;
000330  1011                 WHILE I<=20 DO
000333  1012                   BEGIN
000333  1013                     IF VAL[I] = ' ' THEN GOTO 1 ;
000345  1014                     IF VAL[I] > MAXINTEGER[I]
000363  1015                       THEN BEGIN
000367  1016                              ERROR(23) ;
000371  1017                              VAL := MAXINTEGER ;
000374  1018                              GOTO 1 ;
000375  1019                            END ;
000375  1020                     I := I+1 ;
```

```
000377  1021                    END ;
000400  1022                END ;
000400  1023       1 :
000400  1024          TOKEN . CLASS := NUMBER ;
000403  1025          NEW(P) ;
000405  1026          P ^ .KLASS := INT ;
000414  1027          P ^ .VALU := VAL ;
000422  1028          IF (NOT RECOVERING) AND OPTIMIZE
000423  1029          THEN LINKCONST ( P,P ) ;
000427  1030          TOKEN.CSTADDR:=P;
000432  1031        END ;
000433  1032
000433  1033      ':' : (* COLON OR ASSIGN SYMBOL *)
000433  1034
000433  1035        BEGIN
000433  1036          CH := NEXTCHAR ;
000440  1037          IF CH = '='
000440  1038            THEN BEGIN
000442  1039                   TOKEN . CLASS := ASGNOP ;
000445  1040                   CH := NEXTCHAR ;
000451  1041                 END
000451  1042            ELSE TOKEN . CLASS := COLON ;
000455  1043        END ;
000456  1044
000456  1045      '<' : (* LESSTHEN OR LESSTHENOREQUAL OR NOTEQUAL OPERATOR *)
000456  1046
000456  1047        BEGIN
000456  1048          CH := NEXTCHAR ;
000463  1049          TOKEN . CLASS := RELOP ;
000466  1050          IF CH = '='
000466  1051            THEN BEGIN
000470  1052                   TOKEN . OP := LEOP ;
000472  1053                   CH := NEXTCHAR ;
000477  1054                 END
000477  1055            ELSE IF CH = '>'
000500  1056                   THEN BEGIN
000502  1057                          TOKEN . OP := NEOP ;
000505  1058                          CH := NEXTCHAR ;
000511  1059                        END
000511  1060                   ELSE TOKEN . OP := LTOP ;
000515  1061        END ;
000516  1062
000516  1063      '>' : (* GREATERTHEN OR GREATERTHENOREQUAL OPERATOR *)
000516  1064
000516  1065        BEGIN
000516  1066          CH := NEXTCHAR ;
000523  1067          TOKEN . CLASS := RELOP ;
000526  1068          IF CH = '='
000526  1069            THEN BEGIN
000530  1070                   TOKEN . OP := GEOP ;
000532  1071                   CH := NEXTCHAR ;
000537  1072                 END
000537  1073            ELSE TOKEN . OP := GTOP ;
000543  1074        END ;
000544  1075
000544  1076      '$' : (* OCTAL KONSTANT *)
000544  1077
000544  1078        BEGIN
000544  1079          CH := NEXTCHAR ;
000551  1080          VAL := '                    ' ;
```

```
000554  1081          IF NOT (CH IN ODIGITS)
000556  1082            THEN BEGIN
000557  1083                   ERROR(1) ;
000560  1084                   IF CH IN DIGITS
000560  1085                    THEN REPEAT
000563  1086                           CH := NEXTCHAR ;
000570  1087                         UNTIL NOT (CH IN DIGITS) ;
000572  1088                 END
000572  1089            ELSE BEGIN
000573  1090                   I:=1 ;
000575  1091                   REPEAT
000575  1092                     VAL[I] := CH ;
000613  1093                     CH := NEXTCHAR ;
000617  1094                     I:=I+1 ;
000621  1095                   UNTIL ( NOT (CH IN ODIGITS)) OR (I>20) ;
000624  1096                   IF CH IN ODIGITS
000624  1097                     THEN BEGIN
000626  1098                            ERROR(23) ;
000630  1099                            REPEAT
000630  1100                              CH:=NEXTCHAR ;
000635  1101                            UNTIL NOT (CH IN DIGITS) ;
000637  1102                            VAL:=MAXOCTAL ;
000642  1103                          END
000642  1104                     ELSE IF CH IN DIGITS
000643  1105                            THEN BEGIN
000645  1106                                   ERROR(1) ;
000647  1107                                   REPEAT
000647  1108                                     CH:= NEXTCHAR ;
000654  1109                                   UNTIL NOT(CH IN DIGITS) ;
000656  1110                                   VAL := MAXOCTAL ;
000661  1111                                 END
000661  1112                            ELSE BEGIN
000662  1113                                   J:=I-1 ;
000664  1114                                   K:=20 ;
000665  1115                                   WHILE J>0 DO
000667  1116                                     BEGIN
000667  1117                                       VAL[K] := VAL[J] ;
000714  1118                                       K:=K-1 ;
000715  1119                                       J:=J-1 ;
000716  1120                                     END ;
000717  1121                                   FOR I:=1 TO K DO VAL[I]:='0' ;
000741  1122                                 END ;
000741  1123                 END ;
000741  1124          TOKEN . CLASS := NUMBER ;
000744  1125          NEW(P) ;
000746  1126          P ^ .KLASS := INT ;
000755  1127          P ^ .VALU := VAL ;
000763  1128          IF (NOT RECOVERING) AND OPTIMIZE
000764  1129            THEN LINKCONST ( P,P ) ;
000770  1130          TOKEN.CSTADDR:=P;
000773  1131        END ;
000774  1132
000774  1133      '-' : (* MINUS OR COND SYMBOL *)
000774  1134
000774  1135        BEGIN
000774  1136          TOKEN . CLASS := ADDOP ;
000777  1137          CH := NEXTCHAR ;
001004  1138          IF CH = '>'
001004  1139            THEN BEGIN
001006  1140                   TOKEN . CLASS := CONDSY ;
```

```
001011  1141                  CH := NEXTCHAR ;
001015  1142                END
001015  1143            ELSE TOKEN . OP := MINUS ;
001021  1144        END ;
001022  1145
001022  1146      ';','(',')','[',']',',','!','*','+','=' : (* REST SYMBOLS *)
001022  1147
001022  1148        BEGIN
001022  1149          TOKEN := RESTSYMBOLS [CH] ;
001026  1150          CH := NEXTCHAR ;
001033  1151        END ;
001034  1152
001034  1153      '/' : (* SLASH OR COMMENT *)
001034  1154
001034  1155        BEGIN
001034  1156          CH := NEXTCHAR ;
001041  1157          IF CH = '/'
001041  1158            THEN BEGIN
001043  1159                   READLINE ;
001044  1160                   FINISHED := FALSE ;
001046  1161                   CH := NEXTCHAR ;
001052  1162                 END
001052  1163            ELSE BEGIN
001053  1164                   TOKEN . CLASS := MULOP ;
001056  1165                   TOKEN . OP    := RDIV  ;
001061  1166                 END ;
001061  1167        END ;
001062  1168
001062  1169      '"' : (* CHARACTER STRINGS *)
001062  1170
001062  1171        BEGIN
001062  1172          LGTH := 0 ;
001064  1173          FOR I := 1 TO 10 DO STRBUF[I] := ' ' ;
001100  1174          ERR := FALSE ;
001101  1175          UNCLOSED := TRUE ;
001103  1176
001103  1177          CHCNT:=CHCNT+1 ;
001105  1178          IF CHCNT > 80
001105  1179            THEN BEGIN
001107  1180                   READLINE ;
001110  1181                   CHCNT:=1 ;
001112  1182                 END ;
001112  1183          IF INBUF[CHCNT] = '"' THEN GOTO 6 ;
001125  1184          LGTH := 1 ;
001127  1185          STRBUF[LGTH] := INBUF[CHCNT] ;
001147  1186
001147  1187      4 :
001147  1188          CHCNT:=CHCNT+1 ;
001151  1189          IF CHCNT > 80
001151  1190            THEN BEGIN
001153  1191                   READLINE ;
001155  1192                   CHCNT:=1 ;
001157  1193                 END ;
001157  1194          IF INBUF[CHCNT] = '"' THEN GOTO 6 ;
001172  1195      5 :
001172  1196          LGTH:=LGTH+1 ;
001174  1197          IF (LGTH > 10) AND (NOT ERR)
001177  1198            THEN BEGIN
001200  1199                   SYMSTART:=CHCNT ;
001202  1200                   ERROR(36) ;
```

```
001203  1201                    ERR := TRUE ;
001205  1202                  END ;
001205  1203          IF NOT ERR THEN STRBUF[LGTH] := INBUF[CHCNT] ;
001230  1204          GOTO 4 ;
001231  1205      6 :
001231  1206          CHCNT:=CHCNT+1 ;
001233  1207          IF CHCNT > 80
001233  1208            THEN BEGIN
001235  1209                   READLINE ;
001237  1210                   CHCNT:=1 ;
001241  1211                 END ;
001241  1212          IF INBUF[CHCNT] = '"' THEN GOTO 5 ;
001254  1213
001254  1214      (* END OF STRING DETECTION *)
001254  1215
001254  1216          CH:=INBUF[CHCNT] ;
001270  1217          UNCLOSED := FALSE ;
001271  1218          IF LGTH = 0
001271  1219            THEN BEGIN
001273  1220                   ERROR(3) ;
001275  1221                   LGTH := 1 ;
001277  1222                   STRBUF[LGTH] := ' ' ;
001306  1223                 END ;
001306  1224          IF ERR THEN LGTH:=10 ;
001311  1225          TOKEN . CLASS := STRING ;
001314  1226          NEW(P) ;
001316  1227          P ^ .KLASS := STR    ;
001325  1228          P ^ .STRLEN := LGTH  ;
001334  1229          FOR I:=1 TO 10 DO P ^ .STRVALU[I] := STRBUF[I];
001362  1230          IF (NOT RECOVERING) AND OPTIMIZE
001363  1231            THEN LINKCONST ( P,P ) ;
001367  1232          TOKEN.CSTADDR:=P;
001372  1233        END ;
001373  1234
001373  1235      OTHERWISE (* ILLEGAL SYMBOL OF THE LANGUAGE *)
001436  1236
001436  1237        BEGIN
001436  1238          ERROR(4) ;
001440  1239          CH := NEXTCHAR ;
001445  1240          FINISHED := FALSE ;
001446  1241        END ;
001446  1242      END ; (* CASE *)
001446  1243
001446  1244    UNTIL FINISHED ;
001450  1245
001450  1246 END ; (* SCANNER *)
001476  1247
001476  1248       (*  E R R O R     R E C O V E R Y     R O U T I N E S  *)
001476  1249
001476  1250 (*****************************************************************
001476  1251  *                                                               *
001476  1252  *  THIS ROUTINE GETS NEXT TOKEN IF THE CURRENT TOKEN IS THE      *
001476  1253  *  REQUIRED ONE OTHERWISE STARTS ERROR RECOVERY                  *
001476  1254  *                                                               *
001476  1255  *****************************************************************)
001476  1256
001476  1257 PROCEDURE MUSTBE ;
000003  1258
000003  1259 BEGIN (* MUSTBE *)
000003  1260   IF RECOVERING
```

```
000004  1261      THEN BEGIN
000006  1262             SKIPPING := TRUE ;
000007  1263             WHILE TOKEN.CLASS <> S DO SCANNER ;
000014  1264             SKIPPING := FALSE ;
000016  1265             SCANNER ;
000017  1266             RECOVERING := FALSE ;
000021  1267           END
000021  1268      ELSE IF TOKEN.CLASS = S
000022  1269             THEN SCANNER
000025  1270             ELSE REPORT(S) ;
000032  1271 END ; (* MUSTBE *)
000036  1272
000036  1273 (*****************************************************************
000036  1274  *                                                               *
000036  1275  *  THIS FUNCTION RETURNS TRUE AND ADVANCES TO THE NEXT TOKEN    *
000036  1276  *  IF THE CURRENT TOKEN IS TEH EXPECTED ONE , OTHERWISE         *
000036  1277  *  RETURNS FALSE .                                              *
000036  1278  *                                                               *
000036  1279  *****************************************************************)
000036  1280
000036  1281 FUNCTION HAVE ;
000006  1282
000006  1283 BEGIN (* HAVE *)
000006  1284   IF TOKEN.CLASS = S
000006  1285     THEN BEGIN
000011  1286             SCANNER ;
000012  1287             HAVE := TRUE ;
000014  1288           END
000014  1289     ELSE HAVE := FALSE ;
000017  1290 END ; (* HAVE *)
000024  1291
000024  1292 (*****************************************************
000024  1293  *                                                   *
000024  1294  *  THIS ROUTINE REPORTS AN ERROR DURING RECOVERY    *
000024  1295  *                                                   *
000024  1296  *****************************************************)
000024  1297
000024  1298 PROCEDURE REPORT ;
000003  1299
000003  1300 BEGIN (* REPORT *)
000003  1301   IF NOT RECOVERING
000004  1302     THEN BEGIN
000006  1303            IF ERRBUF[SYMSTART] = ' '
000016  1304              THEN BEGIN
000021  1305                     ERRCNT:=ERRCNT+1 ;
000022  1306                     IF ERRPOS='9'
000022  1307                       THEN ERRPOS:='*'
000025  1308                       ELSE IF ERRPOS='*'
000027  1309                              THEN
000031  1310                              ELSE ERRPOS:=SUCC(ERRPOS) ;
000035  1311                   END
000035  1312              ELSE TOTERR:=TOTERR+1 ;
000040  1313            ERRBUF[SYMSTART] := ERRPOS ;
000055  1314            WRITE('    ** POSITION',ERRCNT:3) ;
000066  1315            WRITE(' ** ERROR 37 ** ') ;
000073  1316            WRITE(REPORTARR [TOKEN.CLASS],' FOUND WHERE ') ;
000110  1317            WRITE(REPORTARR [S],' EXPECTED ') ;
000124  1318            WRITELN ;
000126  1319            RECOVERING := TRUE ;
000130  1320            LINECNT:=LINECNT+1 ;
```

```
000132  1321            IF LINECNT=60 THEN PRINTTITLE ;
000134  1322          END
000134  1323 END ; (* REPORT *)
000147  1324
000147  1325 (*********************************************************
000147  1326  *                                                       *
000147  1327  *  THIS ROUTINE SKIPS UNTIL AN ACCEPTABLE TOKEN IS      *
000147  1328  *  FOUND TO RECOVER PREVIOUS ERROR .                    *
000147  1329  *                                                       *
000147  1330  *********************************************************)
000147  1331
000147  1332 PROCEDURE SKIP ;
000003  1333
000003  1334 BEGIN (* SKIP *)
000003  1335   SKIPPING := TRUE ;
000006  1336   WHILE NOT (TOKEN.CLASS IN LSYS) DO SCANNER ;
000014  1337   SKIPPING := FALSE ;
000016  1338   ERROR(6) ;
000017  1339   XTOP:=0 ;
000021  1340 END ; (* SKIP *)
000023  1341
000023  1342 (*************************************************************
000023  1343  *                                                           *
000023  1344  *  THIS IS THE STARTING POINT OF DECLARATION ROUTINES       *
000023  1345  *  CALLED AT THE BEGINNING OF A BCPL BLOCK LIKE A           *
000023  1346  *  PROCEDURE , FUNCTION , OR MAIN PROGRAM .                 *
000023  1347  *                                                           *
000023  1348  *************************************************************)
000023  1349
000023  1350 PROCEDURE DECLARATION ;
000003  1351
000003  1352 BEGIN (* DECLARATION *)
000003  1353   CASE TOKEN.CLASS OF
000007  1354
000007  1355     LETSY   :  (* FUNCTION OR ROUTINE DECLARATION *)
000007  1356       BEGIN
000007  1357         SCANNER ;
000010  1358         DECLIST ;
000011  1359         WHILE HAVE(ANDSY) DO DECLIST ;
000017  1360         SUCDECLARATION := TRUE ;
000021  1361       END ;
000022  1362
000022  1363     STATSY  :  (* STATIC DECLARATION *)
000022  1364       BEGIN
000022  1365         SCANNER ;
000023  1366         STATIC ;
000024  1367         SUCDECLARATION := TRUE ;
000026  1368       END ;
000027  1369
000027  1370     MANISY  :  (* MANIFEST DECLARATION *)
000027  1371       BEGIN
000027  1372         SCANNER ;
000030  1373         MANIFEST ;
000031  1374         SUCDECLARATION := TRUE ;
000033  1375       END ;
000034  1376
000034  1377     OTHERWISE  SUCDECLARATION := FALSE ;
000052  1378
000052  1379   END ; (* CASE *)
000052  1380 END ; (* DECLARATION *)
```

```
000056  1381
000056  1382 (*************************************************************************
000056  1383  *                                                                       *
000056  1384  *  THIS ROUTINE RECOGNIZES THE DEFINITION OF A PROCEDURE ,               *
000056  1385  *  A FUNCTION , A VECTOR , OR A TABLE AND STARTS RELATED ACTION .        *
000056  1386  *                                                                       *
000056  1387  *************************************************************************)
000056  1388
000056  1389
000056  1390 PROCEDURE DECLIST ;
000002  1391
000002  1392 VAR P,PA,PC : IDP ; C : CTP ; S : PACKED ARRAY [1..10] OF CHAR ;
000007  1393
000007  1394 BEGIN (* FUNCTION OR ROUTINE DECLARATION *)
000007  1395   IF TOKEN.CLASS <> ID
000004  1396     THEN BEGIN
000010  1397            ERROR(5) ;
000011  1398            SKIP([SEMICOLON,ANDSY]) ;
000013  1399          END
000013  1400     ELSE BEGIN
000014  1401            S:=IDSTR ;
000016  1402            SEARCHID(SALEV);
000021  1403            PA := IDPTR ;
000023  1404            SEARCHID(SCLEV) ;
000026  1405            PC := IDPTR ;
000030  1406            SCANNER ;
000031  1407            CASE TOKEN.CLASS OF
000034  1408            LPARENT :
000034  1409              BEGIN  (*LPARENT *)
000034  1410                IF PA <> NIL
000034  1411                  THEN BEGIN
000037  1412                         ERROR(18) ;
000040  1413                         SKIP([SEMICOLON,ANDSY]) ;
000042  1414                       END
000042  1415                  ELSE BEGIN
000043  1416                         ENTERID(ENLEV) ;
000046  1417                         NEW(C) ;
000050  1418                         WITH C^ DO
000055  1419                           BEGIN
000055  1420                             KLASS := STR ;
000060  1421                             STRVALU:=S ;
000061  1422                             STRLEN := 10;
000064  1423                           END ;
000064  1424                         LINKCONST(C,C) ;
000065  1425                         SCANNER ;
000066  1426                         FPL([SYS+[RPARENT]) ;
000071  1427                         MUSTBE(RPARENT) ;
000074  1428                         IF TOKEN.CLASS = BESY
000074  1429                           THEN BEGIN
000077  1430                                  CLEVPTR ^. P := TRUE ;
000106  1431                                  CLEVPTR ^. PFADDR := PROCADDR+1 ;
000113  1432                                  PROCADDR := CLEVPTR ^. PFADDR ;
000120  1433                                  SCANNER ;
000121  1434                                  PROCBODY(LSYS,C) ;
000124  1435                                END
000124  1436                           ELSE IF TOKEN.CLASS = RELOP
000125  1437                                  THEN
000130  1438                                  BEGIN
000130  1439                                    IF TOKEN.OP = EQOP
000130  1440                                      THEN SCANNER
```

```
000132  1441                                        ELSE ERROR(8) ;
000136  1442                                      CLEVPTR ^. P := FALSE ;
000145  1443                                      CLEVPTR ^. PFADDR :=FUNCADDR+1;
000152  1444                                      FUNCADDR := FUNCADDR + 1 ;
000153  1445                                      FUNCBODY(LSYS,C) ;
000156  1446                                    END
000156  1447                                    ELSE
000157  1448                                    BEGIN
000157  1449                                      ERROR(7) ;
000161  1450                                      SKIP([SEMICOLON,ANDSY]) ;
000163  1451                                      FREELEVEL ;
000164  1452                                    END ;
000164  1453                          END ;
000164  1454              END ; (* LPARENT *)
000165  1455            RELOP : (* VECTOR OR TABLE DEFINITION *)
000165  1456              BEGIN
000165  1457                IF PC <> NIL
000165  1458                  THEN BEGIN
000170  1459                         ERROR(14) ;
000171  1460                         SKIP([SEMICOLON,ANDSY]) ;
000173  1461                       END
000173  1462                  ELSE BEGIN
000174  1463                         ENTERID(ECLEV) ;
000177  1464                         P := IDPTR ;
000201  1465                         IF TOKEN.CLASS = RELOP
000201  1466                           THEN IF TOKEN.OP = EQOP
000205  1467                                  THEN SCANNER
000207  1468                                  ELSE ERROR(8) ;
000213  1469                         IF TOKEN.CLASS = VECSY
000213  1470                           THEN VECTOR(P)
000220  1471                           ELSE IF TOKEN.CLASS = TABLESY
000222  1472                                  THEN TABLE(P)
000226  1473                                  ELSE BEGIN
000227  1474                                         ERROR(9) ;
000231  1475                                         SKIP([SEMICOLON,ANDSY]) ;
000233  1476                                       END ;
000233  1477                       END ;
000233  1478              END ; (* RELOP *)
000234  1479            OTHERWISE BEGIN
000252  1480                        ERROR(10) ;
000254  1481                        SKIP([SEMICOLON,ANDSY]) ;
000256  1482                      END ; (* OTHERWISE *)
000256  1483            END ; (* CASE *)
000256  1484          END ; (* ELSE *)
000256  1485 END ; (* DECLIST *)
000273  1486
000273  1487 (*******************************************************
000273  1488  *                                                     *
000273  1489  *  THIS ROUTINE HANDLES THE FORMAL PARAMETERS OF A    *
000273  1490  *  PROCEDURE DECLARATION                              *
000273  1491  *                                                     *
000273  1492  *******************************************************)
000273  1493
000273  1494 PROCEDURE FPL ;
000003  1495
000003  1496 BEGIN (* FPL *)
000003  1497   IF TOKEN.CLASS = ID
000004  1498     THEN BEGIN
000010  1499            SEARCHID(SCLEV) ;
000013  1500            IF IDPTR = NIL
```

```
000013  1501                THEN BEGIN
000016  1502                       ENTERID(ECLEV) ;
000021  1503                       IDPTR ^. KLASS := VARS ;
000030  1504                       IDPTR ^. VADDR := CLEVPTR ^. ADDRESS ;
000044  1505                       IDPTR ^. TAB := FALSE ;
000053  1506                       IDPTR ^. VEC := FALSE ;
000060  1507                       IDPTR ^. VALLINK := NIL ;
000065  1508                       IDPTR ^. NOOFELEM := 0 ;
000074  1509                       BUILDPARAM ;
000075  1510                       SCANNER ;
000076  1511                     END
000076  1512                ELSE BEGIN
000077  1513                       ERROR(14) ;
000101  1514                       SCANNER ;
000102  1515                     END
000102  1516             END
000102  1517      ELSE IF TOKEN.CLASS <> RPARENT
000103  1518              THEN BEGIN
000106  1519                     ERROR(5);
000107  1520                     SKIP([COMMA,RPARENT]) ;
000111  1521                   END ;
000111  1522    WHILE HAVE(COMMA) DO
000115  1523      BEGIN
000115  1524        IF TOKEN.CLASS = ID
000115  1525          THEN BEGIN
000121  1526                 SEARCHID(SCLEV) ;
000124  1527                 IF IDPTR = NIL
000124  1528                   THEN BEGIN
000127  1529                          ENTERID(ECLEV) ;
000132  1530                          IDPTR ^. KLASS := VARS ;
000141  1531                          IDPTR ^. VADDR := CLEVPTR ^. ADDRESS ;
000155  1532                          IDPTR ^. TAB := FALSE ;
000164  1533                          IDPTR ^. VEC := FALSE ;
000171  1534                          IDPTR ^. VALLINK := NIL ;
000176  1535                          IDPTR ^. NOOFELEM := 0 ;
000205  1536                          BUILDPARAM ;
000206  1537                          SCANNER ;
000207  1538                        END
000207  1539                   ELSE BEGIN
000210  1540                          ERROR(14) ;
000212  1541                          SCANNER ;
000213  1542                        END
000213  1543               END
000213  1544          ELSE BEGIN
000214  1545                 ERROR(5) ;
000216  1546                 SKIP([COMMA,RPARENT]) ;
000220  1547               END ;
000220  1548      END ; (* WHILE *)
000221  1549 END ; (* FPL *)
000224  1550
000224  1551 (*********************************************************
000224  1552  *                                                       *
000224  1553  *  THIS ROUTINE LINKS TOGETHER THE FORMAL PARAMETERS    *
000224  1554  *  RECOGNIZED BY PROCEDURE FPL                          *
000224  1555  *                                                       *
000224  1556  *********************************************************)
000224  1557
000224  1558 PROCEDURE BUILDPARAM ;
000002  1559
000002  1560 VAR P,Q : IDP     ;
```

```
000004  1561
000004  1562 BEGIN (* BUILDPARAM *)
000004  1563   CLEVPTR ^. PARNO := CLEVPTR ^. PARNO + 1 ;
000020  1564   P := CLEVPTR ^. PPTR ;
000025  1565   IF P = NIL
000025  1566     THEN BEGIN
000027  1567            CLEVPTR ^. PPTR := IDPTR ;
000036  1568            IDPTR ^. NEXT := NIL ;
000045  1569          END
000045  1570     ELSE BEGIN
000046  1571       WHILE P <> NIL DO
000051  1572         BEGIN
000051  1573           Q := P ;
000052  1574           P := P ^. NEXT ;
000057  1575         END ;
000060  1576       Q ^. NEXT := IDPTR ;
000070  1577       IDPTR ^. NEXT := NIL ;
000076  1578     END ;
000076  1579 END ; (* BUILDPARAM *)
000104  1580
000104  1581 (*************************************************************
000104  1582  *                                                           *
000104  1583  *  THIS ROUTINE HANDLES THE DECLARATION OF A BCPL VECTOR     *
000104  1584  *                                                           *
000104  1585  *************************************************************)
000104  1586
000104  1587 PROCEDURE VECTOR ;
000003  1588
000003  1589 BEGIN (* VECTOR *)
000003  1590   SCANNER ;
000005  1591   IF TOKEN.CLASS = ID
000005  1592     THEN BEGIN
000011  1593            SEARCHID(SALEV) ;
000014  1594            IF IDPTR <> NIL
000014  1595              THEN IF IDPTR ^ . KLASS = KONS
000023  1596                     THEN IF IDPTR ^ .VALU ^. KLASS = INT
000035  1597                            THEN WITH P ^ DO
000043  1598                                   BEGIN
000043  1599                                     KLASS := VARS ;
000046  1600                                     VADDR := CLEVPTR ^. ADDRESS ;
000056  1601                                     VEC   := TRUE ;
000061  1602                                     TAB   := FALSE;
000063  1603                                     VALLINK := NIL ;
000064  1604                                     NOOFELEM:=CONV(IDPTR ^.VALU ^.VALU)+1;
000104  1605                                     CLEVPTR ^. ADDRESS := CLEVPTR ^.
000114  1606                                           ADDRESS + NOOFELEM ;
000120  1607                                     SCANNER ;
000121  1608                                   END
000121  1609                            ELSE BEGIN
000122  1610                                   ERROR(11) ;
000124  1611                                   FREENODE(P) ;
000126  1612                                   SCANNER ;
000127  1613                                 END
000127  1614                     ELSE BEGIN
000130  1615                            ERROR(11) ;
000132  1616                            FREENODE(P);
000134  1617                            SCANNER ;
000135  1618                          END
000135  1619              ELSE BEGIN
000136  1620                     ERROR(13);
```

```
000140  1621                    FREENODE(P);
000142  1622                    SCANNER ;
000143  1623                  END
000143  1624          END
000143  1625      ELSE IF TOKEN.CLASS = NUMBER
000144  1626             THEN WITH P ^ DO
000153  1627                  BEGIN
000153  1628                    KLASS := VARS ;
000156  1629                    VADDR := CLEVPTR ^. ADDRESS ;
000166  1630                    VEC    := TRUE ;
000167  1631                    TAB    := FALSE ;
000171  1632                    VALLINK := NIL ;
000173  1633                    NOOFELEM := CONV(TOKEN.CSTADDR ^.VALU)+ 1 ;
000210  1634                    CLEVPTR ^. ADDRESS := CLEVPTR^.ADDRESS+NOOFELEM ;
000223  1635                    SCANNER ;
000224  1636                  END
000224  1637             ELSE BEGIN
000225  1638                    ERROR(11) ;
000227  1639                    FREENODE(P) ;
000231  1640                    IF TOKEN.CLASS <> SEMICOLON THEN SCANNER ;
000235  1641                  END ;
000235  1642 END ; (* VECTOR *)
000241  1643
000241  1644 (*********************************************
000241  1645  *                                           *
000241  1646  *  THIS ROUTINE HANDLES A TABLE DEFINITON    *
000241  1647  *                                           *
000241  1648  *********************************************)
000241  1649
000241  1650
000241  1651
000241  1652 PROCEDURE TABLE ;
000003  1653
000003  1654 VAR C : CTP ; LINKED : BOOLEAN ;
000005  1655
000005  1656   (* DROPS THE ENTRY OF THE TABLE FROM THE SYMBOL TABLE *)
000005  1657
000005  1658   PROCEDURE DROPLINK ( DROP : IDP ) ;
000003  1659
000003  1660     PROCEDURE FOLLOWIDP ( R : IDP ) ;
000003  1661
000003  1662     BEGIN (* FOLLOWIDP *)
000003  1663       IF R <> NIL
000004  1664         THEN BEGIN
000006  1665                IF R ^. LLINK = DROP
000012  1666                  THEN R ^. LLINK := NIL
000021  1667                  ELSE FOLLOWIDP(R ^. LLINK) ;
000030  1668                IF R ^. RLINK = DROP
000035  1669                  THEN R ^. RLINK := NIL
000043  1670                  ELSE FOLLOWIDP ( R ^. RLINK ) ;
000052  1671              END ;
000052  1672     END ; (* FOLLOWIDP *)
000056  1673
000056  1674   BEGIN (* DROPLINK *)
000056  1675     FOLLOWIDP ( CLEVPTR ) ;
000006  1676   END ; (* DROPLINK *)
000012  1677
000012  1678 BEGIN (* TABLE *)
000012  1679   OPTIMIZE := FALSE ;
000006  1680   WITH P ^ DO
```

```
000012  1681      BEGIN
000012  1682        KLASS := VARS ;
000015  1683        VADDR := CLEVPTR ^. ADDRESS ;
000025  1684        VEC   := FALSE ;
000027  1685        TAB   := TRUE ;
000031  1686        VALLINK := NIL ;
000033  1687        NOOFELEM := 1 ;
000035  1688      END ;
000035  1689    SCANNER ;
000036  1690    CST(P) ;
000040  1691    WHILE HAVE(COMMA) DO CST(P) ;
000047  1692    IF P ^. VALLINK = NIL
000054  1693      THEN BEGIN
000055  1694             ERROR (19) ;
000057  1695             DROPLINK ( P ) ;
000061  1696             DISPOSE ( P ) ;
000063  1697             CLEVPTR ^. ADDRESS := CLEVPTR ^. ADDRESS -1 ;
000076  1698           END
000076  1699      ELSE CLEVPTR ^. ADDRESS := CLEVPTR ^. ADDRESS+P ^. NOOFELEM ;
000116  1700    OPTIMIZE := TRUE ;
000120  1701  END ; (* TABLE *)
000130  1702
000130  1703  (*****************************************************************
000130  1704   *                                                               *
000130  1705   *  THIS ROUTINE RECOGNIZES AND ENTERS VALUES OF A TABLE INTO    *
000130  1706   *  CONSTANTS TABLE AND LINKS THEM TOGETHER AS A LIST .          *
000130  1707   *                                                               *
000130  1708   *****************************************************************)
000130  1709
000130  1710  PROCEDURE CST ;
000003  1711
000003  1712  VAR C : CTP ; I : INTEGER ;
000005  1713
000005  1714    PROCEDURE TABVALUNTER(C:CTP) ;
000003  1715
000003  1716    BEGIN (* TABVALUNTER *)
000003  1717      CTLAST ^. NEXT := C ;
000011  1718      C ^. NEXT      := NIL ;
000016  1719      CONSTADDR      := CONSTADDR+1 ;
000020  1720      C ^. ADDR      := CONSTADDR ;
000030  1721      CTLAST         := C ;
000031  1722      IF P ^. VALLINK = NIL
000036  1723        THEN P ^. VALLINK := CTLAST ;
000044  1724    END ; (* TABVALUNTER *)
000050  1725
000050  1726  BEGIN (* CST *)
000050  1727    IDPTR := NIL ;
000006  1728    IF TOKEN.CLASS IN [NUMBER,STRING]
000006  1729      THEN BEGIN
000011  1730             P ^. NOOFELEM := P ^. NOOFELEM + 1 ;
000026  1731             TABVALUNTER(TOKEN.CSTADDR) ;
000030  1732             SCANNER ;
000031  1733           END
000031  1734      ELSE IF TOKEN.CLASS = ID
000032  1735             THEN BEGIN
000035  1736                    SEARCHID(SALEV) ;
000040  1737                    IF IDPTR <> NIL
000040  1738                      THEN IF IDPTR ^ .KLASS = KONS
000047  1739                             THEN BEGIN
000051  1740                                    NEW (C) ;
```

```
000053  1741                                    C ^. NEXT := NIL ;
000060  1742                                    CONSTADDR := CONSTADDR + 1 ;
000062  1743                                    C ^. ADDR := CONSTADDR ;
000072  1744                                    C ^. KLASS:= IDPTR^.VALU^.KLASS;
000112  1745                                    CASE C ^. KLASS OF
000117  1746                                    INT : C^.VALU:=IDPTR^.VALU^.VALU;
000136  1747                                    STR : BEGIN
000136  1748                                            C^.STRLEN:=IDPTR ^.VALU ^.
000152  1749                                                       STRLEN ;
000156  1750                                            FOR I:=1 TO C^.STRLEN DO
000164  1751                                              C ^. STRVALU[I] :=
000176  1752                                              IDPTR^.VALU^.STRVALU[I];
000225  1753                                          END ;
000226  1754                                    END ; (* CASE *)
000232  1755                                    LINKCONST(C,C) ;
000234  1756                                    P ^. NOOFELEM := P ^. NOOFELEM + 1 ;
000252  1757                                    TABVALUNTER(C) ;
000254  1758                                    SCANNER ;
000255  1759                                  END
000255  1760                              ELSE BEGIN
000256  1761                                    ERROR (12) ;
000260  1762                                    SCANNER ;
000261  1763                                  END
000261  1764                         ELSE BEGIN
000262  1765                                    ERROR(13) ;
000264  1766                                    ERROR(12) ;
000266  1767                                    SCANNER ;
000267  1768                                  END ;
000267  1769                    END
000267  1770               ELSE BEGIN
000270  1771                      ERROR(12) ;
000272  1772                      SKIP([COMMA,ANDSY]) ;
000274  1773                    END ;
000274  1774 END ; (* CST *)
000306  1775
000306  1776 (*************************************************************************
000306  1777  *                                                                       *
000306  1778  *  THIS ROUTINE HANDLES THE DEFINITION OF SIMPLE VARIABLES AND          *
000306  1779  *  THEIR INITIAL VALUES BY REPEATEDLY CALLING PROCEDURE SDEF .          *
000306  1780  *                                                                       *
000306  1781  *************************************************************************)
000306  1782
000306  1783 PROCEDURE STATIC ;
000002  1784
000002  1785 BEGIN (* STATIC *)
000002  1786   MUSTBE (LBRACK) ;
000007  1787   SDEF ;
000010  1788   WHILE (TOKEN.CLASS=SEMICOLON) OR (NOT(TOKEN.CLASS=RBRACK)) DO
000016  1789     BEGIN
000016  1790       IF NOT HAVE(SEMICOLON)
000021  1791         THEN BEGIN
000023  1792                ERROR(24) ;
000024  1793                IF NOT(TOKEN.CLASS = ID) THEN SCANNER ;
000030  1794              END ;
000030  1795       SDEF ;
000031  1796     END ;
000032  1797   MUSTBE (RBRACK) ;
000035  1798 END ; (*STATIC *)
000037  1799
000037  1800
```

```
000037  1801 (*****************************************************************
000037  1802  *                                                               *
000037  1803  *  THIS ROUTINE RECOGNIZES A NEW VARIABLE  TOGETHER WITH ITS    *
000037  1804  *  INITIAL VALUE AND INTRODUCES THEM INTO SYMBOL AND             *
000037  1805  *  CONSTANTS TABLE .                                            *
000037  1806  *                                                               *
000037  1807  *****************************************************************)
000037  1808
000037  1809 PROCEDURE SDEF ;
000002  1810
000002  1811 VAR P : IDP     ;
000003  1812     INERROR : BOOLEAN ;
000004  1813     C : CTP     ;
000005  1814
000005  1815 BEGIN (* SDEF *)
000005  1816   INERROR := FALSE ;
000006  1817   IF TOKEN.CLASS = ID
000006  1818     THEN BEGIN
000011  1819            SEARCHID (SCLEV) ;
000014  1820            IF IDPTR = NIL
000014  1821              THEN
000017  1822              BEGIN
000017  1823                ENTERID(ECLEV) ;
000022  1824                P := IDPTR ;
000024  1825                SCANNER ;
000025  1826                MUSTBE(COLON) ;
000030  1827                IF TOKEN.CLASS IN [NUMBER,STRING]
000030  1828                  THEN
000034  1829                    BEGIN
000034  1830                      WITH P ^ DO
000041  1831                        BEGIN
000041  1832                          KLASS := VARS ;
000044  1833                          VADDR := CLEVPTR ^. ADDRESS ;
000054  1834                          TAB := FALSE ;
000057  1835                          VEC := FALSE ;
000060  1836                          VALLINK := TOKEN.CSTADDR ;
000062  1837                          NOOFELEM := 0 ;
000064  1838                        END ;
000064  1839                      SCANNER ;
000065  1840                    END
000065  1841                  ELSE
000066  1842                    IF TOKEN.CLASS = ID
000066  1843                      THEN
000071  1844                      BEGIN
000071  1845                        SEARCHID(SALEV) ;
000074  1846                        IF IDPTR <> NIL
000074  1847                          THEN
000077  1848                            IF IDPTR ^. KLASS = KONS
000103  1849                              THEN
000105  1850                              BEGIN
000105  1851                                NEW(C) ;
000107  1852                                WITH C ^ DO
000114  1853                                  BEGIN
000114  1854                                    NEXT := NIL ;
000115  1855                                    KLASS:=IDPTR^.VALU^.KLASS ;
000132  1856                                    CASE KLASS OF
000133  1857                                      INT :
000133  1858                                        VALU:=IDPTR ^.VALU ^.VALU ;
000147  1859                                      STR :
000147  1860                                        BEGIN
```

```
000147  1861                                        STRLEN:=IDPTR^.VALU^.STRLEN;
000163  1862                                        STRVALU:=IDPTR^.VALU^.STRVALU;
000175  1863                                      END ;
000176  1864                                    END ;
000202  1865                                  END ;
000202  1866                                LINKCONST(C,C) ;
000204  1867                                WITH P ^ DO
000211  1868                                  BEGIN
000211  1869                                    KLASS := VARS ;
000214  1870                                    VADDR := CLEVPTR ^. ADDRESS ;
000224  1871                                    VEC := FALSE ;
000226  1872                                    TAB := FALSE ;
000230  1873                                    NOOFELEM := 0 ;
000233  1874                                    VALLINK := C ;
000234  1875                                  END ;
000234  1876                                SCANNER ;
000235  1877                              END
000235  1878                              ELSE BEGIN
000236  1879                                     ERROR(12) ;
000240  1880                                     INERROR:= TRUE ;
000242  1881                                     SKIP([SEMICOLON,RBRACK]) ;
000244  1882                                   END
000244  1883                         ELSE BEGIN
000245  1884                                ERROR(13) ;
000247  1885                                INERROR := TRUE ;
000251  1886                                SKIP([SEMICOLON,RBRACK]) ;
000253  1887                              END ;
000253  1888                      END
000253  1889                      ELSE BEGIN
000254  1890                             ERROR(12) ;
000256  1891                             INERROR := TRUE ;
000260  1892                             SKIP([SEMICOLON,RBRACK]) ;
000262  1893                           END ;
000262  1894              END
000262  1895              ELSE BEGIN
000263  1896                     ERROR(14) ;
000265  1897                     SKIP([SEMICOLON,RBRACK]) ;
000267  1898                   END ;
000267  1899          END
000267  1900      ELSE BEGIN
000270  1901             ERROR(5);
000272  1902             SKIP([SEMICOLON,RBRACK]) ;
000274  1903           END ;
000274  1904    IF INERROR
000274  1905      THEN BEGIN
000276  1906             NEW(C) ;
000300  1907             WITH C ^ DO
000305  1908               BEGIN
000305  1909                 NEXT := NIL ;
000306  1910                 KLASS := INT ;
000311  1911                 VALU := '                    0' ;
000313  1912               END ;
000313  1913             LINKCONST(C,C) ;
000315  1914             WITH P ^ DO
000322  1915               BEGIN
000322  1916                 KLASS := VARS;
000325  1917                 VADDR := CLEVPTR ^. ADDRESS ;
000335  1918                 VEC := FALSE ;
000337  1919                 TAB := FALSE ;
000341  1920                 NOOFELEM := 0 ;
```

```
000344  1921                VALLINK := C ;
000345  1922              END ;
000345  1923          END ;
000345  1924 END ; (* SDEF *)
000361  1925
000361  1926 (*************************************************************************
000361  1927  *                                                                       *
000361  1928  *  THIS ROUTINE HANDLES THE DEFINITION OF COMPILE TIME MANIFEST          *
000361  1929  *  CONSTANTS BY REPEATEDLY CALLING PROCEDURE MDEF .                      *
000361  1930  *                                                                       *
000361  1931  *************************************************************************)
000361  1932
000361  1933 PROCEDURE MANIFEST ;
000002  1934
000002  1935 BEGIN (* MANIFEST *)
000002  1936   MUSTBE(LBRACK);
000007  1937   MDEF ;
000010  1938   WHILE (TOKEN.CLASS=SEMICOLON) OR (NOT(TOKEN.CLASS=RBRACK)) DO
000016  1939     BEGIN
000016  1940       IF NOT HAVE(SEMICOLON)
000021  1941         THEN BEGIN
000023  1942                ERROR(24) ;
000024  1943                IF NOT (TOKEN.CLASS = ID) THEN SCANNER ;
000030  1944              END ;
000030  1945       MDEF ;
000031  1946     END ;
000032  1947   MUSTBE (RBRACK) ;
000035  1948 END ; (* MANIFEST *)
000037  1949
000037  1950 (*************************************************************************
000037  1951  *                                                                       *
000037  1952  *  THIS ROUTINE RECOGNIZES A NEW MANIFEST CONSTANT TOGETHER WITH        *
000037  1953  *  ITS COMPILE TIME VALUE AND INTRODUCES THEM INTO SYMBOL AND           *
000037  1954  *  CONSTANTS TABLE .                                                    *
000037  1955  *                                                                       *
000037  1956  *************************************************************************)
000037  1957
000037  1958 PROCEDURE MDEF ;
000002  1959
000002  1960 VAR P : IDP      ;
000003  1961     C : CTP      ;
000004  1962     INERROR : BOOLEAN ;
000005  1963
000005  1964 BEGIN (* MDEF *)
000005  1965   INERROR := FALSE ;
000006  1966   IF TOKEN.CLASS = ID
000006  1967     THEN BEGIN
000011  1968            SEARCHID(SCLEV) ;
000014  1969            IF IDPTR = NIL
000014  1970              THEN
000017  1971              BEGIN
000017  1972                ENTERID(ECLEV) ;
000022  1973                P := IDPTR ;
000024  1974                SCANNER ;
000025  1975                MUSTBE (COLON) ;
000030  1976                IF TOKEN.CLASS IN [NUMBER,STRING]
000030  1977                  THEN
000034  1978                  BEGIN
000034  1979                    P ^. KLASS := KONS ;
000043  1980                    P ^. VALU := TOKEN.CSTADDR ;
```

```
000052  1981                    SCANNER ;
000053  1982                  END
000053  1983                  ELSE
000054  1984                    IF TOKEN.CLASS = ID
000054  1985                      THEN
000057  1986                      BEGIN
000057  1987                        SEARCHID(SALEV) ;
000062  1988                        IF IDPTR <> NIL
000062  1989                          THEN
000065  1990                            IF IDPTR ^. KLASS = KONS
000071  1991                              THEN
000073  1992                              BEGIN
000073  1993                                P ^. KLASS := KONS ;
000102  1994                                P ^. VALU := IDPTR ^. VALU ;
000115  1995                                SCANNER ;
000116  1996                              END
000116  1997                              ELSE
000117  1998                              BEGIN
000117  1999                                ERROR(11) ;
000121  2000                                INERROR := TRUE ;
000123  2001                                NEW(C) ;
000125  2002                                C ^. NEXT := NIL ;
000132  2003                                C ^. KLASS := INT ;
000140  2004                                C ^. VALU := '                    0' ;
000147  2005                                P ^. KLASS := KONS ;
000156  2006                                P ^. VALU := C ;
000164  2007                              END
000164  2008                            ELSE BEGIN
000165  2009                                   ERROR(13) ;
000167  2010                                   INERROR := TRUE ;
000171  2011                                   NEW(C) ;
000173  2012                                   C ^. NEXT := NIL ;
000200  2013                                   C ^. KLASS := INT ;
000206  2014                                   C ^. VALU := '                    0' ;
000215  2015                                   P ^. KLASS := KONS ;
000224  2016                                   P ^. VALU := C ;
000232  2017                                 END
000232  2018                      END
000232  2019                      ELSE BEGIN
000233  2020                             ERROR(11) ;
000235  2021                             INERROR:= TRUE ;
000237  2022                             NEW(C) ;
000241  2023                             C ^. NEXT := NIL ;
000246  2024                             C ^. KLASS := INT ;
000254  2025                             C ^. VALU := '                    0' ;
000263  2026                             P ^. KLASS := KONS ;
000272  2027                             P ^. VALU := C ;
000300  2028                           END ;
000300  2029              END
000300  2030              ELSE
000301  2031                BEGIN
000301  2032                  ERROR(14) ;
000303  2033                  INERROR:= TRUE ;
000305  2034                END ;
000305  2035          END
000305  2036      ELSE BEGIN
000306  2037             ERROR(5) ;
000310  2038             INERROR:= TRUE ;
000312  2039          END ;
000312  2040    IF INERROR
```

```
000312  2041      THEN BEGIN
000314  2042             SKIP([ SEMICOLON,RBRACK]) ;
000316  2043
000316  2044           END ;
000316  2045 END ; (* MDEF *)
000332  2046
000332  2047       (*  T R A N S L A T I O N     R O U T I N E S  *)
000332  2048
000332  2049 (*************************************************************************
000332  2050  *                                                                       *
000332  2051  *  THIS ROUTINE IS THE STARTING POINT FOR THE TRANSLATION OF ANY  *
000332  2052  *  BCPL TYPE STATEMENT .                                                *
000332  2053  *                                                                       *
000332  2054  *************************************************************************)
000332  2055
000332  2056 PROCEDURE COMMAND ;
000003  2057
000003  2058 VAR P,Q : IDP     ; CT :CTP ; LEVEL,LEVEL1,TYPEOFCALL : INTEGER ;
000011  2059     A : OPERATOR ; ABOOL,V,NAME,INERROR,TOP : BOOLEAN ; OPE : OPERATOR ;
000020  2060
000020  2061 BEGIN (* COMMAND *)
000020  2062   CASE TOKEN.CLASS OF
000007  2063
000007  2064   ADDROP : (* DIRECT INDIRECT ADDRESSING *)
000007  2065     BEGIN
000007  2066       A:=TOKEN.OP ;
000013  2067       INERROR := FALSE ;
000014  2068       V := FALSE ;
000016  2069       IF A = LVOP
000016  2070         THEN BEGIN
000020  2071                ERROR(41) ;
000021  2072                ABOOL:=FALSE ;
000023  2073              END
000023  2074         ELSE ABOOL:=TRUE ;
000025  2075       SCANNER ;
000026  2076       IF TOKEN.CLASS <> ID
000026  2077         THEN
000032  2078         BEGIN
000032  2079           ERROR(5) ;
000033  2080           SKIP(LSYS) ;
000035  2081         END
000035  2082         ELSE
000036  2083         BEGIN
000036  2084           SEARCHID(SALEV) ;
000041  2085           IF IDPTR = NIL
000041  2086             THEN
000044  2087             BEGIN
000044  2088               ERROR(13) ;
000045  2089               SKIP(LSYS) ;
000047  2090             END
000047  2091             ELSE
000050  2092               IF IDPTR^.KLASS <> VARS
000054  2093                 THEN
000057  2094                 BEGIN
000057  2095                   ERROR(5) ;
000060  2096                   SKIP(LSYS) ;
000062  2097                 END
000062  2098                 ELSE
000063  2099                 BEGIN
000063  2100                   P:=IDPTR ;
```

```
000065  2101                LEVEL:=LEVCNT ;
000067  2102                SCANNER ;
000070  2103                IF HAVE(VECAP)
000073  2104                  THEN
000074  2105                  BEGIN
000074  2106                    V:=TRUE ;
000076  2107                    IF TOKEN.CLASS = ID
000076  2108                      THEN
000101  2109                      BEGIN
000101  2110                        SEARCHID(SALEV) ;
000104  2111                        IF IDPTR = NIL
000104  2112                          THEN
000107  2113                          BEGIN
000107  2114                            ERROR(13) ;
000110  2115                            SKIP(LSYS) ;
000112  2116                            INERROR:=TRUE ;
000114  2117                          END
000114  2118                          ELSE
000115  2119                            IF IDPTR^.KLASS = VARS
000121  2120                              THEN
000124  2121                              BEGIN
000124  2122                                P:=IDPTR ;
000125  2123                                LEVEL1:=LEVCNT ;
000127  2124                                NAME:=TRUE ;
000131  2125                                SCANNER ;
000132  2126                              END
000132  2127                              ELSE
000133  2128                                IF IDPTR^.KLASS = KONS
000137  2129                                  THEN
000141  2130                                  BEGIN
000141  2131                                    NAME:=FALSE ;
000143  2132                                    CT:=IDPTR^.VALU ;
000150  2133                                    SCANNER ;
000151  2134                                  END
000151  2135                                  ELSE
000152  2136                                  BEGIN
000152  2137                                    ERROR(20) ;
000154  2138                                    SKIP(LSYS) ;
000156  2139                                    INERROR:=TRUE ;
000160  2140                                  END ;
000160  2141                      END
000160  2142                      ELSE
000161  2143                        IF TOKEN.CLASS IN [NUMBER,STRING]
000161  2144                          THEN
000164  2145                          BEGIN
000164  2146                            NAME:=FALSE ;
000165  2147                            CT:=TOKEN.CSTADDR ;
000167  2148                            SCANNER ;
000170  2149                          END
000170  2150                          ELSE
000171  2151                          BEGIN
000171  2152                            ERROR(20) ;
000173  2153                            SKIP(LSYS) ;
000175  2154                            INERROR:=TRUE ;
000177  2155                          END ;
000177  2156                  END ;
000177  2157                IF NOT INERROR
000177  2158                  THEN
000201  2159                  BEGIN
000201  2160                    MUSTBE(ASGNOP) ;
```

```
000205  2161                        EXPR(LSYS) ;
000207  2162                        IF LEVEL=0
000207  2163                          THEN GENCODE(-1,' ',510,2,3,0,P^.VADDR+3)
000225  2164                          ELSE
000226  2165                          BEGIN
000226  2166                            COMPUTELEVEL(LEVEL) ;
000230  2167                            GENCODE(-1,' ',510,2,6,0,P^.VADDR+3) ;
000245  2168                          END ;
000245  2169                        IF V
000245  2170                          THEN
000247  2171                          BEGIN
000247  2172                            GETREG ;
000250  2173                            IF NAME
000250  2174                              THEN
000252  2175                                IF LEVEL1 = 0
000252  2176                                  THEN GENCODE(-1,' ',510,3,3,0,Q^.VADDR+3)
000270  2177                                  ELSE
000271  2178                                  BEGIN
000271  2179                                    COMPUTELEVEL(LEVEL1) ;
000273  2180                                    GENCODE(-1,' ',510,3,6,0,Q^.VADDR+3) ;
000310  2181                                  END
000310  2182                              ELSE GENCODE(-1,' ',800,3,0,0,CT^.ADDR) ;
000325  2183                            FREEREG ;
000326  2184                            GENCODE(-1,' ',360,2,2,3,0) ;
000336  2185                          END ;
000336  2186                        GENCODE(-1,' ',100,6,1,0,0) ;
000346  2187                        XTOP:=0 ;
000350  2188                        IF ABOOL
000350  2189                          THEN GENCODE(-1,' ',530,6,2,0,0)
000360  2190                          ELSE GENCODE(-1,' ',540,6,2,0,0) ;
000372  2191                      END ;
000372  2192                  END ;
000372  2193          END ;
000372  2194    END ; (* ADDROP *)
000373  2195
000373  2196  HALTSY : (* HALT SYMBOL *)
000373  2197    BEGIN
000373  2198      GENCODE(-1,' ',827,0,0,0,0) ;
000402  2199      SCANNER ;
000403  2200    END ; (* HALT SYMBOL *)
000404  2201
000404  2202  FINISHSY  : (* EMPTY COMMAND BETWEEN A SEMICOLON AND A FINISHSY *)
000404  2203    IF CLEVPTR ^. KLASS <> PROG
000411  2204      THEN BEGIN
000413  2205             ERROR(15) ;
000415  2206             SCANNER ;
000416  2207           END ; (* FINISHSY *)
000417  2208
000417  2209  FORSY : (* FOR COMMAND *)
000417  2210    BEGIN
000417  2211      SCANNER ;
000420  2212      FORPROC(LSYS) ;
000422  2213    END ; (* FORSY *)
000423  2214
000423  2215  GOTOSY : (* GOTO SYMBOL *)
000423  2216    BEGIN
000423  2217      SCANNER ;
000424  2218      IF TOKEN.CLASS <> ID
000424  2219        THEN BEGIN
000430  2220               ERROR(30) ;
```

```
000431  2221                IF TOKEN.CLASS <> SEMICOLON THEN SCANNER ;
000435  2222              END
000435  2223          ELSE BEGIN
000436  2224                SEARCHID(SCLEV) ;
000441  2225                IF IDPTR = NIL
000441  2226                  THEN BEGIN
000444  2227                         JUMPCNT := JUMPCNT + 1 ;
000446  2228                         ENTERID(ECLEV) ;
000450  2229                         WITH IDPTR ^ DO
000455  2230                           BEGIN
000455  2231                             ADDRESS := 0 ;
000456  2232                             KLASS   := LABL ;
000461  2233                             LADDR   := JUMPCNT ;
000466  2234                             DECLARED:= FALSE   ;
000470  2235                           END ;
000470  2236                         SCANNER ;
000471  2237                         GENCODE(-1,' ',023,0,0,0,IDPTR^.LADDR) ;
000505  2238                       END
000505  2239                  ELSE BEGIN
000506  2240                         IF IDPTR ^. KLASS <> LABL
000512  2241                           THEN ERROR(31)
000516  2242                           ELSE GENCODE(-1,' ',023,0,0,0,IDPTR^.LADDR) ;
000532  2243                         SCANNER ;
000533  2244                       END
000533  2245              END
000533  2246      END ; (* GOTOSY *)
000534  2247
000534  2248    ID : (* IDENTIFIER *)
000534  2249      BEGIN
000534  2250        SEARCHID(SALEV) ;
000537  2251        P := IDPTR ;
000541  2252        LEVEL:=LEVCNT ;
000543  2253        TOP:=TOPNODE ;
000547  2254        SEARCHID(SCLEV) ;
000551  2255        Q := IDPTR ;
000553  2256        SCANNER ;
000554  2257        CASE TOKEN.CLASS OF
000557  2258
000557  2259          LPARENT : (* CALL TO A PROCEDURE *)
000557  2260            BEGIN
000557  2261              IF P <> NIL
000557  2262                THEN
000562  2263                  IF P ^. KLASS = PROCFUNC
000566  2264                    THEN
000570  2265                      IF P ^. P
000574  2266                        THEN
000576  2267                        BEGIN
000576  2268                          TYPEOFCALL:=3 ;
000577  2269                          IF P=CLEVPTR
000577  2270                            THEN TYPEOFCALL:=0
000601  2271                            ELSE IF LEVEL=1
000603  2272                                   THEN IF TOP
000605  2273                                          THEN TYPEOFCALL:=0
000607  2274                                          ELSE TYPEOFCALL:=0
000610  2275                                   ELSE IF LEVEL=0
000612  2276                                          THEN TYPEOFCALL:=1
000614  2277                                          ELSE IF LEVEL>1
000615  2278                                                 THEN ERROR(42)
000620  2279                                                 ELSE ERROR(49) ;
000623  2280                          IF TYPEOFCALL=3
```

```
000623  2281                          THEN (* ERROR ALREADY REPORTED *)
000626  2282                          ELSE
000627  2283                          BEGIN
000627  2284                            SCANNER ;
000630  2285                            ELIST(LSYS+[RPARENT],P^.PARNO) ;
000640  2286                            MUSTBE(RPARENT) ;
000643  2287                            GENCODE(-1,' ',610,7,0,0,TYPEOFCALL) ;
000653  2288                            GENCODE(-1,' ',807,6,0,0,3) ;
000663  2289                            GENCODE(-2,' ',510,6,4,0,2) ;
000673  2290                            GENCODE(-2,' ',024,0,0,0,P^.PFADDR) ;
000707  2291                            GENCODE(-2,' ',460,0,0,0,0) ;
000716  2292                          END ;
000716  2293                        END
000716  2294                        ELSE BEGIN
000717  2295                               ERROR(35) ;
000721  2296                               SKIP(LSYS) ;
000723  2297                             END
000723  2298                    ELSE BEGIN
000724  2299                           ERROR(4) ;
000726  2300                           SKIP(LSYS) ;
000730  2301                         END
000730  2302                ELSE BEGIN
000731  2303                       ERROR(13) ;
000733  2304                       SKIP(LSYS) ;
000735  2305                     END ;
000735  2306          END ;
000736  2307
000736  2308        ASGNOP : (* SIMPLE ASSIGNMENT COMMAND *)
000736  2309          BEGIN
000736  2310            IF P = NIL
000736  2311              THEN
000741  2312              BEGIN
000741  2313                ERROR(13) ;
000742  2314                SKIP(LSYS) ;
000744  2315              END
000744  2316              ELSE
000745  2317                IF P^.KLASS = PROCFUNC
000751  2318                  THEN
000754  2319                    IF P^.P
000760  2320                      THEN
000761  2321                      BEGIN
000761  2322                        ERROR(47) ;
000763  2323                        SKIP(LSYS) ;
000765  2324                      END
000765  2325                      ELSE
000766  2326                        IF P <> CLEVPTR
000766  2327                          THEN
000770  2328                          BEGIN
000770  2329                            ERROR(48) ;
000772  2330                          END
000772  2331                          ELSE
000773  2332                          BEGIN
000773  2333                            SCANNER ;
000774  2334                            EXPR(LSYS) ;
000776  2335                            GENCODE(-1,' ',100,6,1,0,0) ;
001006  2336                            GENCODE(-1,' ',510,6,3,0,3) ;
001016  2337                            FREEREG ;
001017  2338                          END
001017  2339                    ELSE
001020  2340                      IF P^.KLASS <> VARS
```

```
001024  2341                        THEN
001026  2342                        BEGIN
001026  2343                          ERROR(5) ;
001030  2344                          SKIP(LSYS) ;
001032  2345                        END
001032  2346                        ELSE
001033  2347                        BEGIN
001033  2348                          SCANNER ;
001034  2349                          EXPR(LSYS) ;
001036  2350                          GENCODE(-1,' ',100,6,1,0,0) ;
001046  2351                          IF LEVEL = 0
001046  2352                            THEN GENCODE(-1,' ',510,6,3,0,P^.VADDR+3)
001065  2353                            ELSE
001066  2354                            BEGIN
001066  2355                              COMPUTELEVEL(LEVEL) ;
001070  2356                              GENCODE(-1,' ',510,6,6,0,P^.VADDR+3) ;
001105  2357                            END ;
001105  2358                          XTOP:=0 ;
001107  2359                        END ;
001107  2360          END ;
001110  2361
001110  2362        VECAP : (* SIMPLE ASSIGNMENT COMMAND WITH VECTOR OPERATION
001110  2363                   ON THE LEFT HAND SIDE                          *)
001110  2364          BEGIN
001110  2365            NAME:=FALSE ;
001112  2366            INERROR:=FALSE ;
001113  2367            IF P = NIL
001113  2368              THEN
001115  2369              BEGIN
001115  2370                ERROR(13) ;
001117  2371                SKIP(LSYS) ;
001121  2372              END
001121  2373              ELSE
001122  2374                IF P ^. KLASS <> VARS
001126  2375                  THEN
001131  2376                  BEGIN
001131  2377                    ERROR(32) ;
001132  2378                    SKIP(LSYS);
001134  2379                  END
001134  2380                  ELSE
001135  2381                  BEGIN
001135  2382                    SCANNER ;
001136  2383                    IF TOKEN.CLASS IN [NUMBER,STRING]
001136  2384                      THEN CT:=TOKEN.CSTADDR
001142  2385                      ELSE
001145  2386                        IF TOKEN.CLASS = ID
001145  2387                          THEN
001150  2388                          BEGIN
001150  2389                            SEARCHID(SALEV) ;
001153  2390                            IF IDPTR = NIL
001153  2391                              THEN
001156  2392                              BEGIN
001156  2393                                ERROR(13) ;
001157  2394                                SKIP(LSYS);
001161  2395                                INERROR:=TRUE ;
001163  2396                              END
001163  2397                              ELSE
001164  2398                                IF IDPTR^.KLASS = KONS
001170  2399                                THEN CT:=IDPTR^.VALU
001177  2400                                ELSE
```

```
001201  2401                                  IF IDPTR^.KLASS = VARS
001205  2402                                    THEN
001207  2403                                    BEGIN
001207  2404                                      NAME:=TRUE ;
001211  2405                                      Q:=IDPTR;
001212  2406                                      LEVEL1:=LEVCNT ;
001214  2407                                    END
001214  2408                                    ELSE
001215  2409                                    BEGIN
001215  2410                                      ERROR(20) ;
001217  2411                                      SKIP(LSYS);
001221  2412                                      INERROR:=TRUE ;
001223  2413                                    END
001223  2414                              END
001223  2415                              ELSE
001224  2416                              BEGIN
001224  2417                                ERROR(20) ;
001226  2418                                SKIP(LSYS);
001230  2419                                INERROR:=TRUE ;
001232  2420                              END ;
001232  2421                            IF NOT INERROR
001232  2422                              THEN
001234  2423                              BEGIN
001234  2424                                SCANNER ;
001235  2425                                MUSTBE(ASGNOP) ;
001240  2426                                EXPR(LSYS) ;
001242  2427                                IF LEVEL=0
001242  2428                                  THEN GENCODE(-1,' ',510,2,3,0,P^.VADDR+3)
001260  2429                                  ELSE
001261  2430                                  BEGIN
001261  2431                                    COMPUTELEVEL(LEVEL) ;
001263  2432                                    GENCODE(-1,' ',510,2,6,0,P^.VADDR+3) ;
001300  2433                                  END ;
001300  2434                                IF NAME
001300  2435                                  THEN
001302  2436                                  BEGIN
001302  2437                                    GETREG ;
001303  2438                                    IF LEVEL1 = 0
001303  2439                                      THEN GENCODE(-1,' ',510,3,3,0,Q^.VADDR+3)
001322  2440                                      ELSE
001323  2441                                      BEGIN
001323  2442                                        COMPUTELEVEL(LEVEL1) ;
001325  2443                                        GENCODE(-1,' ',510,3,6,0,Q^.VADDR+3);
001342  2444                                      END ;
001342  2445                                    FREEREG ;
001343  2446                                    GENCODE(-1,' ',360,2,2,3,0);
001353  2447                                  END
001353  2448                                  ELSE
001354  2449                                  BEGIN
001354  2450                                    GENCODE(-1,' ',800,3,0,0,CT^.ADDR) ;
001370  2451                                    GENCODE(-1,' ',360,2,2,3,0) ;
001400  2452                                  END ;
001400  2453                                GENCODE(-1,' ',100,6,1,0,0) ;
001410  2454                                GENCODE(-1,' ',530,6,2,0,0) ;
001420  2455                                XTOP:=0 ;
001422  2456                              END ;
001422  2457                        END ;
001422  2458                  END ; (* VECAP *)
001423  2459
001423  2460                COLON : (* LABEL *)
```

```
001423  2461          BEGIN
001423  2462            IF Q = NIL
001423  2463              THEN BEGIN
001426  2464                     ENTERID(ECLEV) ;
001431  2465                     JUMPCNT := JUMPCNT + 1 ;
001433  2466                     WITH IDPTR ^ DO
001440  2467                       BEGIN
001440  2468                         ADDRESS := 0 ;
001441  2469                         KLASS := LABL ;
001444  2470                         LADDR := JUMPCNT ;
001450  2471                         DECLARED := TRUE ;
001452  2472                       END ;
001452  2473                     SCANNER ;
001453  2474                     GENCODE(IDPTR^.LADDR,'J',460,0,0,0,0) ;
001466  2475                     COMMAND(LSYS) ;
001470  2476                   END
001470  2477              ELSE BEGIN
001471  2478                     IF Q ^. KLASS = LABL
001475  2479                       THEN IF Q ^. DECLARED
001504  2480                              THEN ERROR(21)
001506  2481                              ELSE BEGIN
001510  2482                                     Q ^. DECLARED := TRUE ;
001517  2483                                     GENCODE(Q^.LADDR,'J',460,0,0,0,0);
001531  2484                                   END
001531  2485                       ELSE ERROR(16) ;
001534  2486                     SCANNER ;
001535  2487                     COMMAND(LSYS) ;
001537  2488                   END ;
001537  2489          END ;
001540  2490
001540  2491        OTHERWISE BEGIN
001565  2492                    ERROR(4) ;
001567  2493                    SKIP(LSYS) ;
001571  2494                  END ;
001571  2495      END ; (* CASE *)
001571  2496    END ; (* ID *)
001572  2497
001572  2498  IFSY : (* IF COMMAND *)
001572  2499    BEGIN
001572  2500      SCANNER ;
001573  2501      IFPROC(LSYS) ;
001575  2502    END ; (* IFSY *)
001576  2503
001576  2504  LBRACK : (* BLOCK *)
001576  2505    BEGIN
001576  2506      SCANNER ;
001577  2507      IF TOKEN.CLASS <> RBRACK THEN COMMAND(LSYS+[RBRACK]) ;
001605  2508      WHILE (TOKEN.CLASS=SEMICOLON) OR (NOT(TOKEN.CLASS=RBRACK)) DO
001613  2509        BEGIN
001613  2510          IF NOT HAVE(SEMICOLON)
001616  2511            THEN BEGIN
001620  2512                   ERROR(24) ;
001621  2513                   SKIP(LSYS+[RBRACK]) ;
001624  2514                 END
001624  2515            ELSE IF TOKEN.CLASS = FINISHSY
001625  2516                   THEN BEGIN
001630  2517                          ERROR(17) ;
001632  2518                          SCANNER ;
001633  2519                          SKIP(LSYS+[RBRACK]) ;
001636  2520                        END
```

```
001636  2521                    ELSE IF TOKEN.CLASS = RBRACK
001637  2522                           THEN
001642  2523                           ELSE COMMAND(LSYS+[RBRACK]) ;
001646  2524          END ;
001647  2525        MUSTBE(RBRACK) ;
001652  2526      END ; (* LBRACK / BLOCK *)
001653  2527
001653  2528    READSY : (* A READ COMMAND *)
001653  2529      BEGIN
001653  2530        OPE:=TOKEN.OP ;
001657  2531        LIB[2]:=TRUE ;
001662  2532        IF OPE = L
001662  2533          THEN BEGIN           (* READLN *)
001663  2534                 SCANNER ;
001664  2535                 GENCODE(-1,' ',819,0,0,0,0) ;
001673  2536               END
001673  2537          ELSE IF OPE = C
001674  2538                 THEN BEGIN     (* READCH *)
001676  2539                        SCANNER ;
001677  2540                        READCHPROC ;
001700  2541                      END
001700  2542                 ELSE WRITELN('ERROR IN READ PORTION') ;
001707  2543      END ;
001710  2544
001710  2545    REPEATSY : (* REPEAT COMMAND *)
001710  2546      BEGIN
001710  2547        SCANNER ;
001711  2548        REPEATPROC(LSYS) ;
001713  2549      END ;
001714  2550
001714  2551    RESULTSY : (* RESULT COMMAND *)
001714  2552      BEGIN
001714  2553        SCANNER ;
001715  2554        RESULTPROC(LSYS) ;
001717  2555      END ;
001720  2556
001720  2557    RETURNSY : (* RETURN COMMAND *)
001720  2558      BEGIN
001720  2559        SCANNER ;
001721  2560        RETURNPROC(LSYS) ;
001723  2561      END ;
001724  2562
001724  2563    SEMICOLON : (* EMPTY COMMAND BETWEEN TWO SEMICOLONS *) ;
001725  2564
001725  2565    SWITCHSY : (* SWITCHON COMMAND *)
001725  2566      BEGIN
001725  2567        SCANNER ;
001726  2568        SWITCHPROC(LSYS) ;
001730  2569      END ;
001731  2570
001731  2571    TESTSY : (* TEST COMMAND *)
001731  2572      BEGIN
001731  2573        SCANNER ;
001732  2574        TESTPROC(LSYS) ;
001734  2575      END ;
001735  2576
001735  2577    UNLESSY : (* UNLESS COMMAND *)
001735  2578      BEGIN
001735  2579        SCANNER ;
001736  2580        UNLESSPROC(LSYS) ;
```

```
001740  2581     END ;
001741  2582
001741  2583   WHILESY : (* WHILE COMMAND *)
001741  2584     BEGIN
001741  2585       SCANNER ;
001742  2586       WHILEPROC(LSYS) ;
001744  2587     END ;
001745  2588
001745  2589   WRITESY : (* A WRITE COMMAND *)
001745  2590     BEGIN
001745  2591       OPE:=TOKEN.OP ;
001751  2592       LIB[1]:=TRUE ;
001754  2593       IF OPE = L
001754  2594         THEN BEGIN              (* WRITELN  *)
001755  2595                SCANNER ;
001756  2596                GENCODE(-1,' ',818,0,0,0,0) ;
001765  2597              END
001765  2598         ELSE IF OPE = N
001766  2599                THEN BEGIN       (* WRITEN   *)
001770  2600                       SCANNER ;
001771  2601                       WRITENUMPROC(LSYS) ;
001773  2602                     END
001773  2603                ELSE IF OPE = S
001774  2604                       THEN BEGIN              (* WRITESTR *)
001776  2605                              SCANNER ;
001777  2606                              WRITESTRPROC(LSYS) ;
002001  2607                            END
002001  2608                       ELSE BEGIN                    (* WRITEO *)
002002  2609                              SCANNER ;
002003  2610                              WRITEOPROC(LSYS) ;
002005  2611                            END ;
002005  2612
002005  2613     END ;
002006  2614
002006  2615   OTHERWISE BEGIN  (* ILLEGAL COMMAND *)
002042  2616               ERROR(27) ;
002044  2617               SKIP(LSYS) ;
002046  2618             END ;
002046  2619
002046  2620   END ; (* CASE *)
002046  2621 END ; (* COMMAND *)
002106  2622
002106  2623 (*******************************************
002106  2624  *                                         *
002106  2625  *  TRANSLATES :  IF <EXPR> DO <COMMAND>   *
002106  2626  *                                         *
002106  2627  *******************************************)
002106  2628
002106  2629 PROCEDURE IFPROC ;
000003  2630
000003  2631 VAR LAB1,L1,L2 : INTEGER ;
000006  2632
000006  2633 BEGIN (* IFPROC *)
000006  2634   LAB1 := LABCNT+1 ;
000006  2635   L1:=LAB1+1 ;
000007  2636   L2:=L1+1 ;
000010  2637   LABCNT := L2 ;
000011  2638   EXPR(LSYS+[DOSY]) ;
000013  2639   MUSTBE(DOSY) ;
000016  2640   GENCODE(-1,' ',033,0,1,0,L1) ;
```

```
000026  2641    GENCODE(-1,' ',020,0,0,0,LAB1) ;
000036  2642    GENCODE(L1,'L',030,0,1,0,L2) ;
000046  2643    GENCODE(-1,' ',020,0,0,0,LAB1) ;
000056  2644    GENCODE(L2,'L',460,0,0,0,0) ;
000065  2645    FREEREG  ;
000066  2646    COMMAND(LSYS) ;
000070  2647    GENCODE(LAB1,'L',460,0,0,0,0) ;
000077  2648  END ; (* IFPROC *)
000107  2649
000107  2650  (*********************************************
000107  2651   *                                           *
000107  2652   *  TRANSLATES : UNLESS <EXPR> DO <COMMAND>  *
000107  2653   *                                           *
000107  2654   *********************************************)
000107  2655
000107  2656  PROCEDURE UNLESSPROC ;
000003  2657
000003  2658  VAR LAB1,LAB2 : INTEGER ;
000005  2659
000005  2660  BEGIN (* UNLESSPROC *)
000005  2661    LAB1 := LABCNT+1 ;
000006  2662    LAB2 := LAB1+1 ;
000007  2663    LABCNT := LAB2 ;
000010  2664    EXPR(LSYS+[DOSY]) ;
000012  2665    MUSTBE(DOSY) ;
000015  2666    GENCODE(-1,' ',033,0,1,0,LAB1) ;
000025  2667    GENCODE(-1,' ',030,0,1,0,LAB2) ;
000035  2668    GENCODE(-1,' ',020,0,0,0,LAB1) ;
000045  2669    GENCODE(LAB2,'L',460,0,0,0,0) ;
000054  2670    FREEREG ;
000055  2671    COMMAND(LSYS) ;
000057  2672    GENCODE(LAB1,'L',460,0,0,0,0) ;
000066  2673  END ; (* UNLESSPROC *)
000074  2674
000074  2675  (*****************************************
000074  2676   *                                       *
000074  2677   *  TRANSLATES :  TEST <EXPR>            *
000074  2678   *                    THEN <COMMAND>     *
000074  2679   *                    OR   <COMMAND>     *
000074  2680   *                                       *
000074  2681   *****************************************)
000074  2682
000074  2683  PROCEDURE TESTPROC ;
000003  2684
000003  2685  VAR LAB1,LAB2,LAB3,LAB4 : INTEGER ;
000007  2686
000007  2687  BEGIN (* TESTPROC *)
000007  2688    LAB1 := LABCNT+1 ;
000006  2689    LAB2 := LAB1+1 ;
000007  2690    LAB3 := LAB2+1 ;
000010  2691    LAB4 := LAB3+1 ;
000011  2692    LABCNT := LAB4 ;
000012  2693    EXPR(LSYS+[THENSY]) ;
000014  2694    MUSTBE(THENSY) ;
000017  2695    GENCODE(-1,' ',033,0,1,0,LAB1) ;
000027  2696    GENCODE(-1,' ',030,0,1,0,LAB2) ;
000037  2697    GENCODE(-1,' ',020,0,0,0,LAB3) ;
000047  2698    GENCODE(LAB1,'L',030,0,1,0,LAB4) ;
000057  2699    GENCODE(-1,' ',020,0,0,0,LAB3) ;
000067  2700    GENCODE(LAB4,'L',460,0,0,0,0) ;
```

```
000076  2701    FREEREG ;
000077  2702    COMMAND(LSYS+[ORSY]) ;
000102  2703    GENCODE(-1,' ',020,0,0,0,LAB3) ;
000112  2704    MUSTBE(ORSY) ;
000115  2705    GENCODE(LAB2,'L',460,0,0,0,0) ;
000124  2706    COMMAND(LSYS) ;
000126  2707    GENCODE(LAB3,'L',460,0,0,0,0) ;
000135  2708  END ; (* TESTPROC *)
000147  2709
000147  2710  (**********************************
000147  2711   *                                *
000147  2712   *  TRANSLATES : RESULTIS <EXPR>  *
000147  2713   *  ONLY WITHIN FUCTIONS .        *
000147  2714   *                                *
000147  2715   **********************************)
000147  2716
000147  2717  PROCEDURE RESULTPROC ;
000003  2718
000003  2719  BEGIN (* RESULTPROC *)
000003  2720    IF CLEVPTR ^. P
000011  2721      THEN BEGIN
000012  2722             ERROR(26) ;
000014  2723             SKIP(LSYS) ;
000016  2724           END
000016  2725      ELSE BEGIN
000017  2726             EXPR(LSYS) ;
000020  2727             FREEREG ;
000021  2728             GENCODE(-1,' ',100,6,1,0,0) ;
000031  2729             GENCODE(-1,' ',510,6,3,0,3) ;
000041  2730             GENCODE(-1,' ',805,0,0,0,0) ;
000050  2731             GENCODE(-1,' ',510,1,4,0,2) ;
000060  2732             GENCODE(-1,' ',630,6,1,0,0) ;
000070  2733             GENCODE(-1,' ',021,6,0,0,0) ;
000100  2734           END ;
000100  2735  END ; (* RESULTPROC *)
000102  2736
000102  2737  (**********************************************
000102  2738   *                                            *
000102  2739   *  TRANSLATES : WHILE <EXPR> DO <COMMAND>    *
000102  2740   *                                            *
000102  2741   **********************************************)
000102  2742
000102  2743  PROCEDURE WHILEPROC ;
000003  2744
000003  2745  VAR LAB1,L1,L2,L3 : INTEGER ;
000007  2746
000007  2747  BEGIN (* WHILEPROC *)
000007  2748    LAB1 := LABCNT+1 ;
000006  2749    L1 := LAB1+1 ;
000007  2750    L2 := L1+1 ;
000010  2751    L3 := L2+1 ;
000011  2752    LABCNT := L3 ;
000012  2753    GENCODE(LAB1,'L',460,0,0,0,0) ;
000021  2754    EXPR(LSYS+[DOSY]) ;
000024  2755    MUSTBE(DOSY) ;
000027  2756    GENCODE(-1,' ',033,0,1,0,L1) ;
000037  2757    GENCODE(-1,' ',020,0,0,0,L2) ;
000047  2758    GENCODE(L1,'L',030,0,1,0,L3) ;
000057  2759    GENCODE(-1,' ',020,0,0,0,L2) ;
000067  2760    GENCODE(L3,'L',460,0,0,0,0 ) ;
```

```
000076  2761    FREEREG ;
000077  2762    COMMAND(LSYS) ;
000101  2763    GENCODE(-1,' ',020,0,0,0,LAB1) ;
000111  2764    GENCODE(L2,'L',460,0,0,0,0) ;
000120  2765  END ; (* WHILEPROC *)
000132  2766
000132  2767  (************************************************
000132  2768   *                                              *
000132  2769   *  TRANSLATES : REPEAT <COMMAND> UNTIL <EXPR>  *
000132  2770   *                                              *
000132  2771   ************************************************)
000132  2772
000132  2773  PROCEDURE REPEATPROC ;
000003  2774
000003  2775  VAR LAB1,LAB2,LAB3 :INTEGER ;
000006  2776
000006  2777  BEGIN (* REPEATPROC *)
000006  2778    LAB1 := LABCNT+1 ;
000006  2779    LAB2 := LAB1+1 ;
000007  2780    LAB3 := LAB2+1 ;
000010  2781    LABCNT := LAB3 ;
000011  2782    GENCODE(LAB1,'L',460,0,0,0,0) ;
000017  2783    COMMAND(LSYS+[UNTILSY]) ;
000022  2784    WHILE (TOKEN.CLASS = SEMICOLON) OR (NOT(TOKEN.CLASS = UNTILSY)) DO
000030  2785      BEGIN
000030  2786        IF NOT HAVE(SEMICOLON)
000033  2787          THEN BEGIN
000035  2788                 ERROR(24) ;
000036  2789                 COMMAND(LSYS+[UNTILSY]) ;
000041  2790               END
000041  2791          ELSE IF TOKEN.CLASS <> UNTILSY
000042  2792                 THEN COMMAND(LSYS+[UNTILSY]) ;
000050  2793      END ;
000051  2794    MUSTBE(UNTILSY) ;
000054  2795    EXPR(LSYS) ;
000056  2796    GENCODE(-1,' ',033,0,1,0,LAB2) ;
000066  2797    GENCODE(-1,' ',030,0,1,0,LAB1) ;
000076  2798    GENCODE(-1,' ',020,0,0,0,LAB1) ;
000106  2799    GENCODE(LAB2,'L',030,0,1,0,LAB3) ;
000116  2800    GENCODE(-1,' ',020,0,0,0,LAB1) ;
000126  2801    GENCODE(LAB3,'L',460,0,0,0,0) ;
000135  2802    FREEREG ;
000136  2803  END ; (* REPEATPROC *)
000146  2804
000146  2805  (*****************************
000146  2806   *                           *
000146  2807   *  TRANSLATES : RETURN      *
000146  2808   *  ONLY WITHIN PROCEDURES   *
000146  2809   *                           *
000146  2810   *****************************)
000146  2811
000146  2812  PROCEDURE RETURNPROC ;
000003  2813
000003  2814  BEGIN (* RETURNPROC *)
000003  2815    IF NOT ( CLEVPTR <> P )
000011  2816      THEN ERROR(25)
000013  2817      ELSE BEGIN
000015  2818             GENCODE(-1,' ',805,0,0,0,0) ;
000024  2819             GENCODE(-1,' ',021,4,0,0,2) ;
000034  2820           END ;
```

```
000034  2821 END ; (* RETURNPROC *)
000036  2822
000036  2823 (*****************************************************
000036  2824  *                                                   *
000036  2825  *  TRANSLATES : SWITCHON <EXPR> INTO                *
000036  2826  *               [ CASE <CONST> : <COMMAND> ; ]*     *
000036  2827  *               [ DEFAULT      : <COMMAND>   ]      *
000036  2828  *               ENDCASE                             *
000036  2829  *                                                   *
000036  2830  *****************************************************)
000036  2831
000036  2832 PROCEDURE SWITCHPROC ;
000003  2833
000003  2834 VAR LABN , CLAB : INTEGER ;
000005  2835
000005  2836 BEGIN (* SWITCHPROC *)
000005  2837   LABN:=LABCNT+1 ;
000006  2838   LABCNT := LABN ;
000007  2839   EXPR(LSYS+[INTOSY,CASESY,DEFAULTSY,ENDCASESY]) ;
000011  2840   MUSTBE(INTOSY) ;
000014  2841   WHILE HAVE(CASESY) DO
000020  2842     BEGIN
000020  2843       IF (TOKEN.CLASS IN [NUMBER,STRING])
000024  2844         THEN
000024  2845         BEGIN
000024  2846           GENCODE(-1,' ',800,2,0,0,TOKEN.CSTADDR ^. ADDR) ;
000040  2847           GENCODE(-1,' ',370,3,1,2,0) ;
000050  2848           CLAB := CASECNT+1 ;
000052  2849           CASECNT := CLAB ;
000053  2850           GENCODE(-1,' ',816,0,3,0,CLAB) ;
000062  2851           XTOP:=0 ;
000064  2852           SCANNER ;
000065  2853           MUSTBE(COLON) ;
000070  2854           COMMAND(LSYS+[CASESY,ENDCASESY,DEFAULTSY]) ;
000073  2855           GENCODE(-1,' ',020,0,0,0,LABN) ;
000103  2856           GENCODE(CLAB,'C',460,0,0,0,0) ;
000112  2857         END
000112  2858         ELSE
000113  2859           IF TOKEN.CLASS = ID
000113  2860             THEN
000116  2861             BEGIN
000116  2862               SEARCHID(SALEV) ;
000121  2863               IF IDPTR <> NIL
000121  2864                 THEN
000124  2865                   IF IDPTR ^. KLASS = KONS
000130  2866                     THEN
000132  2867                     BEGIN
000132  2868                       GENCODE(-1,' ',800,2,0,0,IDPTR^.VALU^.ADDR) ;
000152  2869                       GENCODE(-1,' ',370,3,1,2,0) ;
000162  2870                       CLAB := CASECNT+1 ;
000164  2871                       CASECNT := CLAB ;
000165  2872                       GENCODE(-1,' ',816,0,3,0,CLAB) ;
000174  2873                       XTOP:=0 ;
000176  2874                       SCANNER ;
000177  2875                       MUSTBE(COLON) ;
000202  2876                       COMMAND(LSYS+[CASESY,ENDCASESY,DEFAULTSY]);
000205  2877                       GENCODE(-1,' ',020,0,0,0,LABN) ;
000215  2878                       GENCODE(CLAB,'C',460,0,0,0,0) ;
000224  2879                     END
000224  2880                     ELSE
```

```
000225  2881                    BEGIN
000225  2882                      ERROR(12) ;
000227  2883                      SKIP([CASESY,ENDCASESY,DEFAULTSY]);
000231  2884                    END
000231  2885                ELSE
000232  2886                BEGIN
000232  2887                  ERROR(13) ;
000234  2888                  SKIP([CASESY,ENDCASESY,DEFAULTSY]) ;
000236  2889                END
000236  2890            END
000236  2891            ELSE
000237  2892            BEGIN
000237  2893              ERROR(12) ;
000241  2894              SKIP([CASESY,DEFAULTSY,ENDCASESY]) ;
000243  2895            END ;
000243  2896        IF TOKEN.CLASS = SEMICOLON THEN SCANNER ;
000247  2897      END ; (* WHILE *)
000250  2898    IF TOKEN.CLASS = DEFAULTSY
000250  2899      THEN BEGIN
000254  2900             SCANNER ;
000255  2901             MUSTBE(COLON) ;
000260  2902             COMMAND(LSYS+[ENDCASESY]) ;
000263  2903             IF TOKEN.CLASS = SEMICOLON THEN SCANNER ;
000267  2904             MUSTBE(ENDCASESY) ;
000272  2905             GENCODE(LABN,'L',460,0,0,0,0) ;
000301  2906           END
000301  2907      ELSE BEGIN
000302  2908             MUSTBE(ENDCASESY) ;
000306  2909             LIB[5]:=TRUE ;
000311  2910             GENCODE(-1,' ',817,0,0,0,0) ;
000317  2911             GENCODE(LABN,'L',460,0,0,0,0) ;
000326  2912           END ;
000326  2913 END ; (* SWITCHPROC *)
000337  2914
000337  2915 (*****************************************************************
000337  2916  *                                                               *
000337  2917  *  TRANSLATES : FOR <NAME> := <EXPR> TO <EXPR> [ BY <EXPR> ]    *
000337  2918  *                  DO <COMMAND>                                 *
000337  2919  *                                                               *
000337  2920  *****************************************************************)
000337  2921
000337  2922 PROCEDURE FORPROC ;
000003  2923
000003  2924 VAR INDEX : ICP     ;
000004  2925     LEVEL : INTEGER ; FROM : INTEGER ;
000006  2926     LAB3,LAB5,LAB6 : INTEGER ;
000011  2927
000011  2928 BEGIN (* FORPROC *)
000011  2929   LIB[3]:=TRUE ;
000007  2930   LAB3 := LABCNT+1 ;
000011  2931   LAB5 := LAB3+1 ;
000012  2932   LAB6 := LAB5+1 ;
000013  2933   LABCNT := LAB6 ;
000014  2934   IF TOKEN.CLASS <> ID
000014  2935     THEN
000020  2936     BEGIN
000020  2937       ERROR(5) ;
000021  2938       LABCNT:=LABCNT-3 ;
000023  2939       SKIP(LSYS) ;
000025  2940     END
```

```
000025  2941      ELSE
000026  2942      BEGIN
000026  2943        SEARCHID(SALEV) ;
000031  2944        IF IDPTR = NIL
000031  2945          THEN
000034  2946          BEGIN
000034  2947            ERROR(13) ;
000035  2948            LABCNT := LABCNT-3 ;
000037  2949            SKIP(LSYS) ;
000041  2950          END
000041  2951          ELSE
000042  2952            IF IDPTR ^. KLASS <> VARS
000046  2953              THEN
000051  2954              BEGIN
000051  2955                ERROR(5) ;
000052  2956                LABCNT := LABCNT-3 ;
000054  2957                SKIP(LSYS) ;
000056  2958              END
000056  2959              ELSE
000057  2960              BEGIN
000057  2961                INDEX := IDPTR ;
000061  2962                LEVEL := LEVCNT ;
000062  2963                GENCODE(-1,' ',815,0,0,0,0) ;
000071  2964                SCANNER ;
000072  2965                MUSTBE(ASGNOP) ;
000075  2966                EXPR(LSYS+[TOSY]) ;
000100  2967                GENCODE(-1,' ',100,6,1,0,0) ;
000110  2968                FREEREG ;
000111  2969                GENCODE(-1,' ',809,1,0,0,0) ;
000120  2970                GENCODE(-1,' ',814,6,1,0,-3) ;
000130  2971                MUSTBE(TOSY) ;
000133  2972                EXPR(LSYS+[BYSY,DOSY]) ;
000136  2973                GENCODE(-1,' ',100,6,1,0,0) ;
000146  2974                FREEREG ;
000147  2975                GENCODE(-1,' ',809,1,0,0,0) ;
000156  2976                GENCODE(-1,' ',814,6,1,0,-2) ;
000166  2977                IF HAVE(BYSY)
000171  2978                  THEN
000172  2979                  BEGIN
000172  2980                    EXPR(LSYS+[DOSY]) ;
000175  2981                    GENCODE(-1,' ',660,7,0,0,0) ;
000205  2982                    FREEREG ;
000206  2983                    GENCODE(-1,' ',811,0,0,0,0) ;
000215  2984                  END
000215  2985                  ELSE
000216  2986                  BEGIN
000216  2987                    GENCODE(-1,' ',660,7,1,0,0) ;
000226  2988                    GENCODE(-1,' ',811,0,0,0,0) ;
000235  2989                  END ;
000235  2990                MUSTBE(DOSY) ;
000240  2991                GENCODE(LAB3,'L',812,0,0,0,0) ;
000247  2992                GENCODE(-1,' ',040,7,0,0,LAB5) ;
000257  2993                GENCODE(-1,' ',020,0,0,0,LAB6) ;
000267  2994                GENCODE(LAB5,'L',809,1,0,0,0) ;
000276  2995                GENCODE(-1,' ',814,1,1,0,-3) ;
000306  2996                GENCODE(-1,' ',100,6,1,0,0) ;
000316  2997                IF LEVEL = 0
000316  2998                  THEN FROM:=3
000320  2999                  ELSE BEGIN
000322  3000                         COMPUTELEVEL(LEVEL) ;
```

```
000324  3001                        FROM:=6 ;
000326  3002                      END ;
000326  3003                GENCODE(-1,' ',510,6,FROM,0,INDEX ^. VADDR+3) ;
000344  3004                COMMAND(LSYS) ;
000346  3005                GENCODE(-1,' ',813,0,0,0,0) ;
000355  3006                GENCODE(-1,' ',020,0,0,0,LAB3) ;
000365  3007                GENCODE(LAB6,'L',809,1,0,0,0) ;
000374  3008                GENCODE(-1,' ',710,2,0,0,3) ;
000404  3009                GENCODE(-1,' ',370,6,1,2,0) ;
000414  3010                GENCODE(-1,' ',540,6,1,0,0) ;
000424  3011              END
000424  3012    END
000424  3013 END ; (* FORPROC *)
000443  3014
000443  3015 (*********************************
000443  3016  *                               *
000443  3017  *  TRANSLATES : READCH(<NAME>)   *
000443  3018  *                               *
000443  3019  *********************************)
000443  3020
000443  3021 PROCEDURE READCHPROC ;
000002  3022
000002  3023 VAR P,Q : IDP ; C : CTP ; V,NAME,INERROR : BOOLEAN ; L,L1 : INTEGER ;
000012  3024
000012  3025 BEGIN (* READCHPROC *)
000012  3026   INERROR:=FALSE ;
000006  3027   MUSTBE(LPARENT) ;
000011  3028   IF TOKEN.CLASS <> ID
000011  3029     THEN
000015  3030     BEGIN
000015  3031       ERROR(5) ;
000016  3032       SKIP([RPARENT]) ;
000020  3033     END
000020  3034     ELSE
000021  3035     BEGIN
000021  3036       SEARCHID(SALEV) ;
000024  3037       IF IDPTR = NIL
000024  3038         THEN
000027  3039         BEGIN
000027  3040           ERROR(13) ;
000030  3041           SKIP([RPARENT]) ;
000032  3042         END
000032  3043         ELSE
000033  3044           IF IDPTR^.KLASS <> VARS
000037  3045             THEN
000042  3046             BEGIN
000042  3047               ERROR(5) ;
000043  3048               SKIP([RPARENT]) ;
000045  3049             END
000045  3050             ELSE
000046  3051             BEGIN
000046  3052               P:=IDPTR ;
000050  3053               L:=LEVCNT;
000052  3054               SCANNER ;
000053  3055               IF HAVE(VECAP)
000056  3056                 THEN
000057  3057                 BEGIN
000057  3058                   V:=TRUE ;
000061  3059                   IF TOKEN.CLASS = ID
000061  3060                     THEN
```

```
000064  3061                    BEGIN
000064  3062                      SEARCHID(SALEV) ;
000067  3063                      IF IDPTR = NIL
000067  3064                        THEN
000072  3065                        BEGIN
000072  3066                          ERROR(13) ;
000073  3067                          SKIP([RPARENT]) ;
000075  3068                          INERROR:=TRUE ;
000077  3069                        END
000077  3070                        ELSE
000100  3071                          IF IDPTR^.KLASS = VARS
000104  3072                            THEN
000107  3073                            BEGIN
000107  3074                              C:=IDPTR ;
000110  3075                              L1:=LEVCNT ;
000112  3076                              NAME:=TRUE ;
000114  3077                              SCANNER ;
000115  3078                            END
000115  3079                            ELSE
000116  3080                              IF IDPTR^.KLASS = KONS
000122  3081                                THEN
000124  3082                                BEGIN
000124  3083                                  NAME:=FALSE ;
000126  3084                                  C:=IDPTR^.VALU ;
000133  3085                                  SCANNER ;
000134  3086                                END
000134  3087                                ELSE
000135  3088                                BEGIN
000135  3089                                  ERROR(20) ;
000137  3090                                  SKIP([RPARENT]) ;
000141  3091                                  INERROR:=TRUE ;
000143  3092                                END ;
000143  3093                    END
000143  3094                    ELSE
000144  3095                      IF TOKEN.CLASS IN [NUMBER,STRING]
000144  3096                        THEN
000147  3097                        BEGIN
000147  3098                          NAME:=FALSE ;
000150  3099                          C:=TOKEN.CSTADDR ;
000152  3100                          SCANNER ;
000153  3101                        END
000153  3102                        ELSE
000154  3103                        BEGIN
000154  3104                          ERROR(20) ;
000156  3105                          SKIP([RPARENT]) ;
000160  3106                          INERROR:=TRUE ;
000162  3107                        END ;
000162  3108                END
000162  3109              END ;
000162  3110            IF NOT INERROR
000162  3111              THEN
000164  3112              BEGIN
000164  3113                GENCODE(-1,' ',820,0,0,0,0) ;
000173  3114                GENCODE(-1,' ',100,6,1,0,0) ;
000203  3115                IF NOT V
000203  3116                  THEN GENCODE(-1,' ',510,6,3,0,P^.VADDR+3)
000222  3117                  ELSE
000223  3118                  BEGIN
000223  3119                    IF L=0
000223  3120                      THEN GENCODE(-1,' ',510,1,3,0,P^.VADDR+3)
```

```
000241  3121                      ELSE
000242  3122                      BEGIN
000242  3123                        COMPUTELEVEL(L) ;
000244  3124                        GENCODE(-1,' ',510,1,6,0,P^.VADDR+3) ;
000261  3125                      END ;
000261  3126                    IF NAME
000261  3127                      THEN
000263  3128                        IF L1=0
000263  3129                          THEN GENCODE(-1,' ',510,2,3,0,Q^.VADDR+3)
000301  3130                          ELSE
000302  3131                          BEGIN
000302  3132                            GETREG ;
000303  3133                            COMPUTELEVEL(L1) ;
000305  3134                            FREEREG ;
000306  3135                            GENCODE(-1,' ',510,2,6,0,Q^.VADDR+3) ;
000323  3136                          END
000323  3137                      ELSE GENCODE(-1,' ',800,2,0,0,C^.ADDR) ;
000340  3138                    GENCODE(-1,' ',360,1,1,2,0) ;
000347  3139                    GENCODE(-1,' ',530,6,2,0,0) ;
000357  3140                    XTOP:=0 ;
000361  3141                  END ;
000361  3142                END ;
000361  3143      END ;
000361  3144    MUSTBE(RPARENT) ;
000364  3145 END ; (* READCHPROC *)
000407  3146
000407  3147 (*********************************************
000407  3148  *                                           *
000407  3149  *  TRANSLATES : WRITEN( <EXPR> [,<EXPR>]* ) *
000407  3150  *  FOR DECIMAL OUTPUT                       *
000407  3151  *                                           *
000407  3152  *********************************************)
000407  3153
000407  3154 PROCEDURE WRITENUMPROC ;
000003  3155
000003  3156 LABEL 1 ;
000003  3157
000003  3158 BEGIN (* WRITENUMPROC *)
000003  3159   MUSTBE(LPARENT) ;
000010  3160 1 :
000010  3161   EXPR(LSYS+[RPARENT]) ;
000013  3162   GENCODE(-1,' ',821,0,0,0,0);
000022  3163   XTOP:=0 ;
000024  3164   IF HAVE(COMMA) THEN GOTO 1 ;
000031  3165   MUSTBE(RPARENT) ;
000034  3166 END ; (* WRITENUMPROC *)
000036  3167
000036  3168 (*********************************************
000036  3169  *                                           *
000036  3170  *  TRANSLATES : WRITEO( <EXPR> [,<EXPR>]* ) *
000036  3171  *  FOR OCTAL OUTPUT                         *
000036  3172  *                                           *
000036  3173  *********************************************)
000036  3174
000036  3175 PROCEDURE WRITEOPROC ;
000003  3176
000003  3177 LABEL 1 ;
000003  3178
000003  3179 BEGIN (* WRITEOPROC *)
000003  3180   MUSTBE(LPARENT) ;
```

```
000010  3181 1 :
000010  3182   EXPR(LSYS+[RPARENT]) ;
000013  3183   GENCODE(-1,' ',826,0,0,0,0);
000022  3184   XTOP:=0 ;
000024  3185   IF HAVE(COMMA) THEN GOTO 1 ;
000031  3186   MUSTBE(RPARENT) ;
000034  3187 END ; (* WRITEOPROC *)
000036  3188
000036  3189 (*****************************************************************
000036  3190  *                                                               *
000036  3191  *  TRANSLATES : WRITESTR( <EXPR>:<FORM> [,<EXPR>:<FORM>]* ) *
000036  3192  *  FOR STRING OUTPUT                                            *
000036  3193  *                                                               *
000036  3194  *****************************************************************)
000036  3195
000036  3196 PROCEDURE WRITESTRPROC ;
000003  3197
000003  3198 LABEL  1 ;
000003  3199
000003  3200 BEGIN (* WRITESTRPROC *)
000003  3201   MUSTBE(LPARENT) ;
000010  3202 1 :
000010  3203   EXPR(LSYS+[COLON,RPARENT]) ;
000013  3204   IF HAVE(COLON)
000016  3205     THEN BEGIN
000017  3206            EXPR(LSYS+[RPARENT]) ;
000022  3207            GENCODE(-1,' ',822,0,0,0,0) ;
000031  3208            XTOP:=0 ;
000033  3209          END
000033  3210     ELSE BEGIN
000034  3211            GENCODE(-1,' ',760,2,1,0,0) ;
000043  3212            GENCODE(-1,' ',822,0,0,0,0) ;
000052  3213            XTOP:=0 ;
000054  3214          END ;
000054  3215   IF HAVE(COMMA) THEN GOTO 1 ;
000061  3216   MUSTBE(RPARENT) ;
000064  3217 END ; (* WRITESTRPROC *)
000067  3218
000067  3219     (*  T R A N S L A T I O N     O F     E X P R E S S I O N S  *)
000067  3220
000067  3221
000067  3222 (*  <EXPR> ::= <EXPRO> [ -> <EXPR> , <EXPR> ]   *)
000067  3223
000067  3224 PROCEDURE EXPR ;
000003  3225
000003  3226 VAR LAB1,LAB2,LAB3,LAB4 : INTEGER ;
000007  3227
000007  3228 BEGIN (* EXPR *)
000007  3229   EXPRO(LSYS) ;
000005  3230   IF TOKEN.CLASS = CONDSY
000005  3231     THEN BEGIN
000011  3232            LAB1:=LABCNT+1;
000013  3233            LAB2:=LAB1+1 ;
000014  3234            LAB3:=LAB2+1 ;
000015  3235            LAB4:=LAB3+1 ;
000016  3236            LABCNT:=LAB4 ;
000017  3237            GENCODE(-1,' ',033,0,1,0,LAB1) ;
000026  3238            GENCODE(-1,' ',030,0,1,0,LAB2) ;
000036  3239            GENCODE(-1,' ',020,0,0,0,LAB3) ;
000046  3240            GENCODE(LAB1,'L',030,0,1,0,LAB4) ;
```

```
000056  3241              GENCODE(-1,' ',020,0,0,0,LAB3) ;
000066  3242              GENCODE(LAB4,'L',460,0,0,0,0) ;
000075  3243              FREEREG ;
000076  3244              SCANNER ;
000077  3245              EXPR(LSYS) ;
000101  3246              MUSTBE(COMMA) ;
000104  3247              GENCODE(-1,' ',020,0,0,0,LAB3) ;
000114  3248              GENCODE(LAB2,'L',460,0,0,0,0) ;
000123  3249              EXPR(LSYS) ;
000125  3250              GENCODE(LAB3,'L',460,0,0,0,0) ;
000134  3251            END ;
000134  3252 END ; (* EXPR *)
000146  3253
000146  3254 (*   <EXPRO> ::= <EXP1> [ NEQV <EXP1> ]*   *)
000146  3255
000146  3256 PROCEDURE EXPRO ;
000003  3257
000003  3258 BEGIN (* EXPRO *)
000003  3259   EXP1(LSYS) ;
000005  3260   WHILE TOKEN.CLASS = NEQVSY DO
000010  3261     BEGIN
000010  3262       SCANNER ;
000011  3263       EXP1(LSYS) ;
000013  3264       GENCODE(-1,' ',130,XTOP-1,XTOP-1,XTOP,0) ;
000023  3265       FREEREG ;
000024  3266     END ;
000025  3267 END ; (* EXPRO *)
000027  3268
000027  3269 (*   <EXP1> ::= <EXP2> [ EQV <EXP2> ]*   *)
000027  3270
000027  3271 PROCEDURE EXP1 ;
000003  3272
000003  3273 BEGIN (* EXP1 *)
000003  3274   EXP2(LSYS) ;
000005  3275   WHILE TOKEN.CLASS = EQVSY DO
000011  3276     BEGIN
000011  3277       SCANNER ;
000012  3278       EXP2(LSYS) ;
000014  3279       GENCODE(-1,' ',130,XTOP-1,XTOP-1,XTOP,0) ;
000024  3280       FREEREG ;
000025  3281       GENCODE(-1,' ',140,XTOP,XTOP,0,0) ;
000035  3282     END ;
000036  3283 END ; (* EXP1 *)
000040  3284
000040  3285 (*   <EXP2> ::= <EXP3> [ LOR <EXP3> ]*   *)
000040  3286
000040  3287 PROCEDURE EXP2 ;
000003  3288
000003  3289 BEGIN (* EXP2 *)
000003  3290   EXP3(LSYS) ;
000005  3291   WHILE TOKEN.CLASS = LOR DO
000011  3292     BEGIN
000011  3293       SCANNER ;
000012  3294       EXP3(LSYS) ;
000014  3295       GENCODE(-1,' ',120,XTOP-1,XTOP-1,XTOP,0) ;
000024  3296       FREEREG ;
000025  3297     END ;
000026  3298 END ; (* EXP2 *)
000030  3299
000030  3300 (*   <EXP3> ::= <EXP4> [ LAND <EXP4> ]*   *)
```

```
000030  3301
000030  3302 PROCEDURE EXP3 ;
000003  3303
000003  3304 BEGIN (* EXP3 *)
000003  3305   EXP4(LSYS) ;
000005  3306   WHILE TOKEN.CLASS = LAND DO
000011  3307     BEGIN
000011  3308       SCANNER ;
000012  3309       EXP4(LSYS) ;
000014  3310       GENCODE(-1,' ',110,XTOP-1,XTOP-1,XTOP,0) ;
000024  3311       FREEREG ;
000025  3312     END ;
000026  3313 END ; (* EXP3 *)
000030  3314
000030  3315 (*   <EXP4> ::= [ NOT ] <EXP5> [ <RELOPL> <EXP5> ]*   *)
000030  3316
000030  3317 PROCEDURE EXP4 ;
000003  3318
000003  3319 VAR N:BOOLEAN ; O:OPERATOR ; LAB1,LAB2:INTEGER ;
000007  3320
000007  3321 BEGIN (* EXP4 *)
000007  3322   N:=FALSE ;
000006  3323   IF TOKEN.CLASS = NOTSY
000006  3324     THEN BEGIN
000011  3325            SCANNER ;
000012  3326            N:=TRUE ;
000014  3327          END ;
000014  3328   EXP5(LSYS) ;
000016  3329   IF N THEN GENCODE(-1,' ',140,XTOP,0,XTOP,0) ;
000027  3330   WHILE TOKEN.CLASS = RELOP DO
000033  3331     BEGIN
000033  3332       O:=TOKEN.OP ;
000037  3333       SCANNER ;
000040  3334       EXP5(LSYS) ;
000042  3335       LAB1:=LABCNT+1 ;
000044  3336       LAB2:=LAB1+1 ;
000045  3337       LABCNT:=LAB2 ;
000046  3338       CASE O OF
000050  3339       EQOP : BEGIN
000050  3340                GENCODE(-1,' ',370,XTOP-1,XTOP-1,XTOP,0) ;
000060  3341                FREEREG ;
000061  3342                GENCODE(-1,' ',030,0,XTOP,0,LAB1) ;
000072  3343                GENCODE(-1,' ',760,XTOP,0,0,0) ;
000102  3344                GENCODE(-1,' ',020,0,0,0,LAB2) ;
000112  3345                GENCODE(LAB1,'L',800,XTOP,0,0,0) ;
000122  3346                GENCODE(LAB2,'L',460,0,0,0,0) ;
000131  3347              END ; (* EQOP *)
000132  3348       GTOP : BEGIN
000132  3349                GENCODE(-1,' ',370,XTOP-1,XTOP-1,XTOP,0) ;
000142  3350                FREEREG ;
000143  3351                GENCODE(-1,' ',033,0,XTOP,0,LAB1) ;
000154  3352                GENCODE(-1,' ',030,0,XTOP,0,LAB1) ;
000165  3353                GENCODE(-1,' ',800,XTOP,0,0,0) ;
000175  3354                GENCODE(-1,' ',020,0,0,0,LAB2) ;
000205  3355                GENCODE(LAB1,'L',760,XTOP,0,0,0) ;
000215  3356                GENCODE(LAB2,'L',460,0,0,0,0) ;
000224  3357              END ; (* GTOP *)
000225  3358       LTOP : BEGIN
000225  3359                GENCODE(-1,' ',370,XTOP-1,XTOP-1,XTOP,0) ;
000235  3360                FREEREG ;
```

```
000236  3361                GENCODE(-1,' ',030,0,XTOP,0,LAB1) ;
000247  3362                GENCODE(-1,' ',032,0,XTOP,0,LAB1) ;
000260  3363                GENCODE(-1,' ',800,XTOP,0,0,0) ;
000270  3364                GENCODE(-1,' ',020,0,0,0,LAB2) ;
000300  3365                GENCODE(LAB1,'L',760,XTOP,0,0,0) ;
000310  3366                GENCODE(LAB2,'L',460,0,0,0,0) ;
000317  3367              END ; (* LTOP *)
000320  3368        GEOP : BEGIN
000320  3369                GENCODE(-1,' ',370,XTOP-1,XTOP-1,XTOP,0) ;
000330  3370                FREEREG ;
000331  3371                GENCODE(-1,' ',033,0,XTOP,0,LAB1) ;
000342  3372                GENCODE(-1,' ',800,XTOP,0,0,0) ;
000352  3373                GENCODE(-1,' ',020,0,0,0,LAB2) ;
000362  3374                GENCODE(LAB1,'L',760,XTOP,0,0,0) ;
000372  3375                GENCODE(LAB2,'L',460,0,0,0,0) ;
000401  3376              END ; (* GEOP *)
000402  3377        LEOP : BEGIN
000402  3378                GENCODE(-1,' ',370,XTOP-1,XTOP-1,XTOP,0) ;
000412  3379                FREEREG ;
000413  3380                GENCODE(-1,' ',030,0,XTOP,0,LAB1) ;
000424  3381                GENCODE(-1,' ',033,0,XTOP,0,LAB1) ;
000435  3382                GENCODE(-1,' ',760,XTOP,0,0,0) ;
000445  3383                GENCODE(-1,' ',020,0,0,0,LAB2) ;
000455  3384                GENCODE(LAB1,'L',800,XTOP,0,0,0) ;
000465  3385                GENCODE(LAB2,'L',460,0,0,0,0) ;
000474  3386              END ; (* LEOP *)
000475  3387        NEOP : BEGIN
000475  3388                GENCODE(-1,' ',370,XTOP-1,XTOP-1,XTOP,0) ;
000505  3389                FREEREG ;
000506  3390                GENCODE(-1,' ',031,0,XTOP,0,LAB1) ;
000517  3391                GENCODE(-1,' ',760,XTOP,0,0,0) ;
000527  3392                GENCODE(-1,' ',020,0,0,0,LAB2) ;
000537  3393                GENCODE(LAB1,'L',800,XTOP,0,0,0) ;
000547  3394                GENCODE(LAB2,'L',460,0,0,0,0) ;
000556  3395              END ; (* NEOP *)
000557  3396        OTHERWISE ERROR(49) ;
000573  3397        END ; (* CASE *)
000573  3398      END ; (* WHILE *)
000574  3399 END ; (* EXP4 *)
000606  3400
000606  3401 (*   <EXP5> ::= <EXP6> [ <ADDOP> <EXP6> ]*    *)
000606  3402
000606  3403 PROCEDURE EXP5 ;
000003  3404
000003  3405 VAR O : OPERATOR ;
000004  3406
000004  3407 BEGIN (* EXP5 *)
000004  3408   EXP6(LSYS) ;
000005  3409   WHILE TOKEN.CLASS = ADDOP DO
000010  3410     BEGIN
000010  3411       O:=TOKEN.OP ;
000014  3412       SCANNER ;
000015  3413       EXP6(LSYS) ;
000017  3414       CASE O OF
000021  3415         PLUS  : GENCODE(-1,' ',360,XTOP-1,XTOP-1,XTOP,0) ;
000032  3416         MINUS : GENCODE(-1,' ',370,XTOP-1,XTOP-1,XTOP,0) ;
000043  3417         OTHERWISE ERROR(49) ;
000062  3418       END ; (* CASE *)
000062  3419       FREEREG ;
000063  3420     END ; (* WHILE *)
```

```
000064  3421 END ; (* EXP5 *)
000070  3422
000070  3423 (*  <EXP6> ::= [ <ADDOP> ] <EXP7> [ <MULOP> <EXP7> ]*    *)
000070  3424
000070  3425 PROCEDURE EXP6 ;
000003  3426
000003  3427 VAR O:OPERATOR ; A:BOOLEAN ;
000005  3428
000005  3429 BEGIN (* EXP6 *)
000005  3430   A:=FALSE ;
000006  3431   IF TOKEN.CLASS = ADDOP
000006  3432     THEN BEGIN
000011  3433            O:=TOKEN.OP ;
000014  3434            A:=TRUE ;
000015  3435            SCANNER ;
000016  3436          END ;
000016  3437   EXP7(LSYS) ;
000020  3438   IF A
000020  3439     THEN BEGIN
000022  3440            CASE O OF
000023  3441              PLUS : ;
000024  3442              MINUS : BEGIN
000024  3443                        GENCODE(-1,' ',760,7,0,0,0) ;
000034  3444                        GENCODE(-1,' ',370,XTOP,7,XTOP,0) ;
000044  3445                      END ; (* MINUS *)
000045  3446              OTHERWISE ERROR(49) ;
000064  3447            END ; (* CASE *)
000064  3448          END ;
000064  3449   WHILE TOKEN.CLASS=MULOP DO
000070  3450     BEGIN
000070  3451       O:=TOKEN.OP ;
000074  3452       SCANNER ;
000075  3453       EXP7(LSYS) ;
000077  3454       CASE O OF
000101  3455         ASTR : BEGIN
000101  3456                  GENCODE(-1,' ',420,XTOP-1,XTOP-1,XTOP,0) ;
000111  3457                  FREEREG ;
000112  3458                END ; (* ASTR *)
000113  3459         RDIV : BEGIN
000113  3460                  GENCODE(-1,' ',810,XTOP-1,XTOP-1,XTOP,0) ;
000123  3461                  FREEREG ;
000124  3462                END ; (* RDIV *)
000125  3463         REM  : BEGIN
000125  3464                  GENCODE(-1,' ',100,6,XTOP-1,0,0) ;
000135  3465                  GENCODE(-1,' ',100,7,XTOP,0,0) ;
000145  3466                  GENCODE(-1,' ',810,XTOP-1,XTOP-1,XTOP,0) ;
000155  3467                  GENCODE(-1,' ',420,XTOP,XTOP-1,7,0) ;
000166  3468                  GENCODE(-1,' ',370,XTOP-1,6,XTOP,0) ;
000176  3469                  FREEREG ;
000177  3470                END ; (* REM *)
000200  3471         OTHERWISE ERROR(49) ;
000211  3472       END ; (* CASE *)
000211  3473     END ; (* WHILE *)
000212  3474 END ; (* EXP6 *)
000220  3475
000220  3476 (*   <EXP7> ::= <EXP8> [ <SHFOP> <EXP8> ]    *)
000220  3477
000220  3478 PROCEDURE EXP7 ;
000003  3479
000003  3480 VAR S:OPERATOR ;
```

```
000004  3481
000004  3482 BEGIN (* EXP7 *)
000004  3483   EXP8(LSYS) ;
000005  3484   IF TOKEN.CLASS = SHFOP
000005  3485     THEN BEGIN
000011  3486            S:=TOKEN.OP ;
000015  3487            SCANNER ;
000016  3488            EXP8(LSYS) ;
000020  3489            GENCODE(-1,' ',630,6,XTOP,0,0) ;
000030  3490            GENCODE(-1,' ',824,6,0,0,0) ;
000040  3491            GENCODE(-1,' ',610,7,0,0,60)   ;
000050  3492            GENCODE(-1,' ',825,7,6,0,0) ;
000060  3493            CASE S OF
000062  3494            LSHF : ;
000063  3495            RSHF : BEGIN
000063  3496                     GENCODE(-1,' ',670,6,0,6,0) ;
000073  3497                     GENCODE(-1,' ',660,6,6,7,0) ;
000103  3498                   END ;
000104  3499            OTHERWISE ERROR(49) ;
000114  3500            END ; (* CASE *)
000114  3501            FREEREG ;
000115  3502            GENCODE(-1,' ',220,XTOP,6,0,0) ;
000125  3503          END ;
000125  3504 END ; (* EXP7 *)
000131  3505
000131  3506 (*  <EXP8> ::= [ <ADDROP> ] <EXP9>   *)
000131  3507
000131  3508 PROCEDURE EXP8 ;
000003  3509
000003  3510 VAR A:OPERATOR ; LEVEL,FROM : INTEGER ;
000006  3511
000006  3512 BEGIN (* EXP8 *)
000006  3513   IF TOKEN.CLASS = ADDROP
000004  3514     THEN BEGIN
000007  3515            A:=TOKEN.OP ;
000013  3516            SCANNER ;
000014  3517            IF TOKEN.CLASS <> ID
000014  3518              THEN BEGIN
000020  3519                     ERROR(5) ;
000021  3520                     SKIP(LSYS) ;
000023  3521                   END
000023  3522              ELSE BEGIN
000024  3523                     SEARCHID(SALEV) ;
000027  3524                     LEVEL:=LEVCNT ;
000031  3525                     IF IDPTR = NIL
000031  3526                       THEN BEGIN
000034  3527                              ERROR(13) ;
000035  3528                              SKIP(LSYS);
000037  3529                            END
000037  3530                       ELSE BEGIN
000040  3531                              IF LEVEL = 0
000040  3532                                THEN FROM:=3
000042  3533                                ELSE BEGIN
000044  3534                                       COMPUTELEVEL(LEVEL) ;
000046  3535                                       FROM:=6 ;
000050  3536                                     END ;
000050  3537                              GETREG ;
000051  3538                              CASE A OF
000053  3539                              LVOP : GENCODE(-1,' ',710,XTOP,FROM,0,
000062  3540                                             IDPTR^.VADDR+3) ;
```

```
000072  3541                                    RVOP : BEGIN
000072  3542                                             GENCODE(-1,' ',510,XTOP,FROM,0,
000101  3543                                                     IDPTR^.VADDR+3) ;
000110  3544                                             GENCODE(-1,' ',530,XTOP,XTOP,0,0) ;
000120  3545                                           END ;
000121  3546                                    OTHERWISE ERROR(49) ;
000131  3547                                    END ; (* CASE *)
000131  3548                                    SCANNER ;
000132  3549                              END ; (* ELSE *)
000132  3550                         END (* ELSE *)
000132  3551               END (* THEN *)
000132  3552          ELSE EXP9(LSYS) ;
000134  3553 END ; (* EXP8 *)
000144  3554
000144  3555 (*    <EXP9> ::= ( <EXPR> )
000144  3556                  <NAME> ! <CONST>
000144  3557                  <NAME> ! <NAME>
000144  3558                  <NAME>
000144  3559                  <NAME>(<ELIST>)          FUNCTION CALL
000144  3560                  <CONST>                                  *)
000144  3561
000144  3562 PROCEDURE EXP9 ;
000003  3563
000003  3564 VAR ID1 : IDP ; LEVEL,LEV1,TYPEOFCALL,FROM,SAVEREG : INTEGER ;
000011  3565
000011  3566 BEGIN (* EXP9 *)
000011  3567   IF TOKEN.CLASS = LPARENT
000004  3568     THEN
000010  3569     BEGIN
000010  3570       SCANNER ;
000011  3571       EXPR(LSYS+[RPARENT]) ;
000014  3572       MUSTBE(RPARENT) ;
000017  3573     END
000017  3574     ELSE
000020  3575       IF TOKEN.CLASS = ID
000020  3576         THEN
000023  3577         BEGIN
000023  3578           SEARCHID(SALEV) ;
000026  3579           LEVEL:=LEVCNT ;
000030  3580           IF IDPTR = NIL
000030  3581             THEN
000033  3582             BEGIN
000033  3583               ERROR(13) ;
000034  3584               SKIP(LSYS) ;
000036  3585             END
000036  3586             ELSE
000037  3587               IF IDPTR ^. KLASS = KONS
000043  3588                 THEN
000046  3589                 BEGIN
000046  3590                   GETREG ;
000047  3591                   GENCODE(-1,' ',800,XTOP,0,0,IDPTR^.VALU^.ADDR);
000070  3592                   SCANNER ;
000071  3593                 END
000071  3594                 ELSE
000072  3595                   IF IDPTR ^. KLASS = VARS
000076  3596                     THEN
000101  3597                     BEGIN
000101  3598                       ID1:=IDPTR ;
000102  3599                       LEV1:=LEVCNT ;
000103  3600                       SCANNER ;
```

```
000104  3601                    IF TOKEN.CLASS = VECAP
000104  3602                    THEN
000110  3603                    BEGIN
000110  3604                    SCANNER ;
000111  3605                    IF (TOKEN.CLASS = NUMBER) OR
000114  3606                       (TOKEN.CLASS = STRING)
000115  3607                      THEN
000117  3608                      BEGIN
000117  3609                        IF LEV1 = 0
000117  3610                          THEN FROM:=3
000121  3611                          ELSE BEGIN
000123  3612                                 COMPUTELEVEL(LEV1) ;
000125  3613                                 FROM:=6 ;
000127  3614                               END ;
000127  3615                        GETREG ;
000130  3616                        GENCODE(-1,' ',510,XTOP,FROM,0,ID1^.VADDR+3);
000146  3617                        GENCODE(-1,' ',630,7,XTOP,0,0) ;
000156  3618                        GENCODE(-1,' ',800,XTOP,0,0,
000165  3619                                TOKEN.CSTADDR^.ADDR) ;
000174  3620                        GENCODE(-1,' ',530,XTOP,XTOP,7,0) ;
000204  3621                        SCANNER ;
000205  3622                      END
000205  3623                      ELSE
000206  3624                        IF TOKEN.CLASS = ID
000206  3625                          THEN
000211  3626                          BEGIN
000211  3627                            SEARCHID(SALEV) ;
000214  3628                            LEVEL:=LEVCNT ;
000216  3629                            IF LEV1 = 0
000216  3630                              THEN FROM:=3
000220  3631                              ELSE
000222  3632                              BEGIN
000222  3633                                COMPUTELEVEL(LEV1) ;
000224  3634                                FROM:=6 ;
000226  3635                              END ;
000226  3636                            GETREG ;
000227  3637                            GENCODE(-1,' ',510,XTOP,FROM,0,
000236  3638                                    ID1^.VADDR+3) ;
000245  3639                            GENCODE(-1,' ',630,7,XTOP,0,0) ;
000255  3640                            IF LEVEL = 0
000255  3641                              THEN FROM:=3
000257  3642                              ELSE
000261  3643                              BEGIN
000261  3644                                COMPUTELEVEL(LEVEL) ;
000263  3645                                FROM:=6 ;
000265  3646                              END ;
000265  3647                            GENCODE(-1,' ',510,XTOP,FROM,0,
000274  3648                                    IDPTR^.VADDR+3) ;
000303  3649                            GENCODE(-1,' ',530,XTOP,XTOP,7,0) ;
000313  3650                            SCANNER ;
000314  3651                          END
000314  3652                          ELSE
000315  3653                          BEGIN
000315  3654                            ERROR(20) ;
000317  3655                            SKIP(LSYS) ;
000321  3656                          END
000321  3657                  END (* VECAP *)
000321  3658                  ELSE
000322  3659                  BEGIN
000322  3660                    IF LEV1 = 0
```

```
000322  3661                        THEN FROM:=3
000324  3662                        ELSE BEGIN
000326  3663                               COMPUTELEVEL(LEV1) ;
000330  3664                               FROM:=6 ;
000332  3665                             END ;
000332  3666                     GETREG ;
000333  3667                     GENCODE(-1,' ',510,XTOP,FROM,0,ID1^.VADDR+3) ;
000351  3668                   END
000351  3669                END (* VARS *)
000351  3670                ELSE
000352  3671                  IF IDPTR ^. KLASS = PROCFUNC
000356  3672                    THEN
000361  3673                      IF NOT IDPTR^.P
000365  3674                        THEN
000366  3675                        BEGIN
000366  3676                          TYPEOFCALL:=3 ;
000370  3677                          IF IDPTR = CLEVPTR
000370  3678                            THEN TYPEOFCALL:=0
000372  3679                            ELSE IF LEVEL = 1
000373  3680                                   THEN IF TOPNODE
000375  3681                                          THEN TYPEOFCALL:=0
000377  3682                                          ELSE TYPEOFCALL:=0
000400  3683                                   ELSE IF LEVEL = 0
000402  3684                                          THEN TYPEOFCALL:=1
000404  3685                                          ELSE IF LEVEL > 1
000405  3686                                                 THEN ERROR(46)
000410  3687                                                 ELSE ERROR(49) ;
000413  3688                          IF TYPEOFCALL = 3
000413  3689                            THEN (* ERROR REPORTED NO CODE *)
000416  3690                            ELSE
000417  3691                            BEGIN
000417  3692                              ID1:=IDPTR ;
000421  3693                              SAVEREG:=XTOP ; XTOP:=0 ;
000423  3694                              GENCODE(-1,' ',806,0,0,0,0);
000432  3695                              SCANNER ;
000433  3696                              MUSTBE(LPARENT) ;
000436  3697                              ELIST(LSYS+[RPARENT],ID1^.PARNO) ;
000445  3698                              MUSTBE(RPARENT) ;
000450  3699                              GENCODE(-1,' ',610,7,0,0,TYPEOFCALL) ;
000460  3700                              GENCODE(-1,' ',807,6,0,0,3) ;
000470  3701                              GENCODE(-2,' ',510,6,4,0,2) ;
000500  3702                              GENCODE(-2,' ',022,0,0,0,ID1^.PFADDR) ;
000514  3703                              GENCODE(-2,' ',808,0,0,0,0) ;
000523  3704                              XTOP:=SAVEREG ;
000525  3705                              GETREG ;
000526  3706                              GENCODE(-1,' ',510,XTOP,4,0,3) ;
000537  3707                            END
000537  3708                          END (* FUNCTION CALL *)
000537  3709                          ELSE
000540  3710                          BEGIN
000540  3711                            ERROR(34) ;
000542  3712                            SKIP(LSYS) ;
000544  3713                          END (* ERROR IF PROCEDURE *)
000544  3714      END (* ID ETC *)
000544  3715      ELSE
000545  3716        IF (TOKEN.CLASS = NUMBER) OR
000547  3717           (TOKEN.CLASS = STRING)
000550  3718          THEN
000552  3719          BEGIN
000552  3720            GETREG ;
```

```
000553  3721            GENCODE(-1,' ',800,XTOP,0,0;TOKEN.CSTADDR^.ADDR) ;
000570  3722            SCANNER ;
000571  3723          END (* CONST *)
000571  3724          ELSE
000572  3725          BEGIN
000572  3726            ERRCR(20) ;
000574  3727            SKIP(LSYS) ;
000576  3728          END ;
000576  3729 END ; (* EXP9 *)
000614  3730
000614  3731 (*************************************************
000614  3732  *                                               *
000614  3733  *  THIS ROUTINE HANDLES BCPL MAIN PROGRAM BLOCK  *
000614  3734  *                                               *
000614  3735  *************************************************)
000614  3736
000614  3737 PROCEDURE BLOCK ;
000003  3738
000003  3739 VAR SUCDECLARATION : BOOLEAN ; I : INTEGER ;
000005  3740
000005  3741 BEGIN (* BLOCK *)
000005  3742   SUCDECLARATION:=FALSE ;
000006  3743   DECLARATION(SUCDECLARATION) ;
000007  3744   WHILE SUCDECLARATION DO
000011  3745     BEGIN
000011  3746       IF NOT HAVE(SEMICOLON) THEN ERROR(24) ;
000017  3747       SUCDECLARATION:=FALSE ;
000021  3748       DECLARATION(SUCDECLARATION) ;
000022  3749     END ;
000023  3750   FOR I:=1 TO 12 DO DAYMESS[I+12]:=MAIN[I] ;
000057  3751   MESSAGE(DAYMESS) ;
000061  3752   WRITELN(CODE,'BCPL$    SX1  STACK$') ; IBANK:=((IBANK+3) DIV 4)*4+2 ;
000073  3753   GENCODE(-1,' ',610,1,0,0,1) ;
000101  3754   GENCODE(-1,' ',630,2,1,0,0) ;
000110  3755   GENCODE(-1,' ',630,3,1,0,0) ;
000120  3756   GENCODE(-1,' ',760,6,0,0,0) ;
000130  3757   GENCODE(-1,' ',560,6,3,0,0) ;
000140  3758   GENCODE(-1,' ',540,6,6,1,0) ;
000150  3759   GENCODE(-1,' ',540,6,6,1,0) ;
000160  3760   GENCODE(-1,' ',540,6,6,1,0) ;
000170  3761   INITVALU ;
000171  3762   GENCODE(-1,' ',760,1,2,0,0) ;
000200  3763   GENCODE(-1,' ',710,2,0,0,CLEVPTR^.ADDRESS+4) ;
000214  3764   GENCODE(-1,' ',360,1,1,2,0) ;
000223  3765   GENCODE(-1,' ',630,4,1,0,0) ;
000233  3766   COMMAND(LSYS+[FINISHSY]) ;
000235  3767   WHILE (TOKEN.CLASS = SEMICOLON) OR
000240  3768         (NOT(TOKEN.CLASS=FINISHSY))  DO
000244  3769     BEGIN
000244  3770       IF NOT HAVE(SEMICOLON)
000246  3771         THEN ERROR(24) ;
000252  3772       IF (TOKEN.CLASS=FINISHSY)
000255  3773         THEN
000255  3774         ELSE COMMAND(LSYS+[FINISHSY]) ;
000260  3775     END ;
000261  3776   FREELEVEL ;
000262  3777 END ; (* BLOCK *)
000272  3778
000272  3779 (*************************************************
000272  3780  *                                               *
```

```
000272  3781  *  THIS ROUTINE HANDLES BCPL PROCEDURE BLOCKS  *
000272  3782  *                                              *
000272  3783  ************************************************)
000272  3784
000272  3785 PROCEDURE PROCBODY ;
000004  3786
000004  3787 VAR SUCDECLARATION : BOOLEAN ; I : INTEGER ;
000006  3788
000006  3789 BEGIN (* PROCBODY *)
000006  3790   LIB[4]:=TRUE ;
000007  3791   IF TOKEN.CLASS = LBRACK
000007  3792     THEN BEGIN
000013  3793            SCANNER ;
000014  3794            SUCDECLARATION:=FALSE ;
000016  3795            DECLARATION(SUCDECLARATION) ;
000017  3796            WHILE SUCDECLARATION DO
000021  3797              BEGIN
000021  3798                IF NOT HAVE(SEMICOLON) THEN ERROR(24) ;
000027  3799                SUCDECLARATION:=FALSE ;
000031  3800                DECLARATION(SUCDECLARATION) ;
000032  3801              END ;
000033  3802            GENCODE(CLEVPTR ^. PFADDR,'P',460,0,0,0,0) ;
000045  3803            GENCODE(-1,' ',710,7,0,0,CLEVPTR^.ADDRESS) ;
000061  3804            GENCODE(-1,' ',800,1,0,0,C^.ADDR) ;
000075  3805            FOR I:=1 TO 10 DO DAYMESS[I+12]:=C^.STRVALU[I] ;
000131  3806            DAYMESS[23]:=' ' ;
000134  3807            DAYMESS[24]:=' ' ;
000136  3808            MESSAGE(DAYMESS) ;
000140  3809            GENCODE(-1,' ',804,0,0,0,0) ;
000147  3810            INITVALU ;
000150  3811            COMMAND(LSYS+[RBRACK]) ;
000153  3812            WHILE (TOKEN.CLASS=SEMICOLON) OR
000156  3813                  (TOKEN.CLASS <> RBRACK) DO
000162  3814              BEGIN
000162  3815                IF NOT HAVE(SEMICOLON)
000165  3816                  THEN ERROR(24) ;
000170  3817                IF TOKEN.CLASS = RBRACK
000170  3818                  THEN
000174  3819                  ELSE COMMAND(LSYS+[RBRACK]) ;
000200  3820              END ;
000201  3821            MUSTBE(RBRACK) ;
000204  3822          END
000204  3823     ELSE BEGIN
000205  3824            GENCODE(CLEVPTR^.PFADDR,'P',460,0,0,0,0) ;
000217  3825            GENCODE(-1,' ',710,7,0,0,CLEVPTR^.ADDRESS) ;
000233  3826            GENCODE(-1,' ',800,1,0,0,C^.ADDR) ;
000247  3827            GENCODE(-1,' ',804,0,0,0,0) ;
000256  3828            COMMAND(LSYS) ;
000260  3829          END ;
000260  3830   FREELEVEL ;
000261  3831   GENCODE(-1,' ',805,0,0,0,0) ;
000270  3832   GENCODE(-1,' ',510,1,4,0,2) ;
000300  3833   GENCODE(-1,' ',630,6,1,0,0) ;
000310  3834   GENCODE(-1,' ',021,6,0,0,0) ;
000320  3835 END ; (* PROCBODY *)
000330  3836
000330  3837 (************************************************
000330  3838  *                                              *
000330  3839  *  THIS ROUTINE HANDLES BCPL FUNCTION  BLOCKS  *
000330  3840  *                                              *
```

```
000330  3841  ************************************************)
000330  3842
000330  3843 PROCEDURE FUNCBODY ;
000004  3844
000004  3845 VAR SUCDECLARATION : BOOLEAN ; I : INTEGER ;
000006  3846
000006  3847 BEGIN (* FUNCBODY *)
000006  3848   LIB[4]:=TRUE ;
000007  3849   LIB[6]:=TRUE ;
000011  3850   IF HAVE(VALCFSY)
000014  3851     THEN BEGIN
000015  3852            IF TOKEN.CLASS = LBRACK
000015  3853              THEN BEGIN
000021  3854                     SCANNER ;
000022  3855                     SUCDECLARATION:=FALSE ;
000024  3856                     DECLARATION(SUCDECLARATION) ;
000025  3857                     WHILE SUCDECLARATION DO
000027  3858                       BEGIN
000027  3859                         IF NOT HAVE(SEMICOLON) THEN ERROR(24) ;
000035  3860                         SUCDECLARATION:=FALSE;
000037  3861                         DECLARATION(SUCDECLARATION) ;
000040  3862                       END ;
000041  3863                     GENCODE(CLEVPTR ^. PFADDR,'F',460,0,0,0,0) ;
000053  3864                     GENCODE(-1,' ',710,7,0,0,CLEVPTR^.ADDRESS) ;
000067  3865                     GENCODE(-1,' ',800,1,0,0,C^.ADDR) ;
000103  3866                     FOR I:=1 TO 10 DO DAYMESS[I+12]:=C^.STRVALU[I] ;
000137  3867                     DAYMESS[23] := ' ';
000142  3868                     DAYMESS[24] := ' ' ;
000144  3869                     MESSAGE(DAYMESS) ;
000146  3870                     GENCODE(-1,' ',804,0,0,0,0) ;
000155  3871                     INITVALU ;
000156  3872                     COMMAND(LSYS+[RBRACK]) ;
000161  3873                     WHILE (TOKEN.CLASS=SEMICOLON) OR
000164  3874                           (TOKEN.CLASS <> RBRACK) DO
000170  3875                       BEGIN
000170  3876                         IF NOT HAVE(SEMICOLON)
000173  3877                           THEN ERROR(24) ;
000176  3878                         IF TOKEN.CLASS = RBRACK
000176  3879                           THEN
000202  3880                           ELSE COMMAND(LSYS+[RBRACK]) ;
000206  3881                       END ;
000207  3882                     MUSTBE(RBRACK) ;
000212  3883                   END
000212  3884              ELSE BEGIN
000213  3885                     GENCODE(CLEVPTR^.PFADDR,'F',460,0,0,0,0) ;
000225  3886                     GENCODE(-1,' ',710,7,0,0,CLEVPTR^.ADDRESS) ;
000241  3887                     GENCODE(-1,' ',800,1,0,0,C^.ADDR) ;
000255  3888                     GENCODE(-1,' ',804,0,0,0,0) ;
000264  3889                     COMMAND(LSYS) ;
000266  3890                   END
000266  3891          END
000266  3892     ELSE BEGIN
000267  3893            GENCODE(CLEVPTR^.PFADDR,'F',460,0,0,0,0) ;
000301  3894            GENCODE(-1,' ',710,7,0,0,CLEVPTR^.ADDRESS) ;
000315  3895            GENCODE(-1,' ',800,1,0,0,C^.ADDR) ;
000331  3896            GENCODE(-1,' ',804,0,0,0,0) ;
000340  3897            EXPR(LSYS) ;
000342  3898            GENCODE(-1,' ',100,6,1,0,0) ;
000352  3899            GENCODE(-1,' ',510,6,3,0,3) ;
000362  3900            XTCP:=0 ;
```

```
000364  3901          END ;
000364  3902    FREELEVEL ;
000365  3903    GENCODE(-1,' ',805,0,0,0,0) ;
000374  3904    GENCODE(-1,' ',510,1,4,0,2) ;
000404  3905    GENCODE(-1,' ',630,6,1,0,0) ;
000414  3906    GENCODE(-1,' ',021,6,0,0,0) ;
000424  3907  END ; (* FUNCBODY *)
000434  3908
000434  3909  (*************************************************************
000434  3910   *                                                           *
000434  3911   *  THIS ROUTINE RECOGNIZES EXPRESSION LISTS TO BE PASSED    *
000434  3912   *  AS ARGUMENTS TO A BCPL ROUTINE . IT ALSO PROVIDES        *
000434  3913   *  NECESSARY CODE TO DO IT AT RUN-TIME                      *
000434  3914   *                                                           *
000434  3915   *************************************************************)
000434  3916
000434  3917  PROCEDURE ELIST ;
000004  3918
000004  3919  VAR ELISTCNT : INTEGER ;
000005  3920
000005  3921  BEGIN (* ELIST *)
000005  3922    ELISTCNT:=3 ;
000006  3923    IF TOKEN.CLASS <> RPARENT
000006  3924      THEN BEGIN
000011  3925             EXPR(LSYS+[COMMA,RPARENT]) ;
000013  3926             ELISTCNT:=ELISTCNT+1 ;
000015  3927             IF PNO = 0
000015  3928               THEN BEGIN
000017  3929                      ERROR(43) ;
000021  3930                      SKIP([RPARENT]) ;
000023  3931                    END
000023  3932               ELSE BEGIN
000024  3933                      GENCODE(-1,' ',100,6,1,0,0) ;
000033  3934                      GENCODE(-1,' ',510,6,4,0,ELISTCNT) ;
000043  3935                      XTOP:=0 ;
000045  3936                      WHILE HAVE(COMMA) DO
000051  3937                        BEGIN
000051  3938                          EXPR(LSYS+[COMMA,RPARENT]) ;
000054  3939                          ELISTCNT:=ELISTCNT+1 ;
000056  3940                          IF ELISTCNT-3 > PNO
000060  3941                            THEN BEGIN
000061  3942                                   ERROR(44) ;
000062  3943                                   SKIP([RPARENT]) ;
000064  3944                                 END
000064  3945                            ELSE BEGIN
000065  3946                                   GENCODE(-1,' ',100,6,1,0,0) ;
000074  3947                                   GENCODE(-1,' ',510,6,4,0,ELISTCNT) ;
000104  3948                                   XTOP:=0 ;
000106  3949                                 END ;
000106  3950                        END ;
000107  3951                    END ;
000107  3952             END
000107  3953      ELSE IF PNO <> 0
000110  3954             THEN ERROR(45) ;
000113  3955  END ; (* ELIST *)
000122  3956
000122  3957  (*************************************************************
000122  3958   *                                                           *
000122  3959   *  THIS ROUTINE PRINTS COMPILE-TIME ERROR MESSAGES          *
000122  3960   *                                                           *
```

```
000122  3961  ********************************************************)
000122  3962
000122  3963 PROCEDURE ERROR ;
000003  3964
000003  3965 BEGIN (* ERROR *)
000003  3966   IF ERRBUF[SYMSTART] = ' '
000014  3967     THEN BEGIN
000017  3968            ERRCNT:=ERRCNT+1 ;
000020  3969            IF ERRPOS='9'
000020  3970              THEN ERRPOS:='*'
000023  3971              ELSE IF ERRPOS='*'
000025  3972                     THEN
000027  3973                     ELSE ERRPOS:=SUCC(ERRPOS)
000031  3974          END
000033  3975     ELSE TOTERR:=TOTERR+1 ;
000036  3976   ERRBUF [SYMSTART] := ERRPOS ;
000053  3977   WRITE('    ** POSITION',ERRCNT:3,' ** ERROR ',I:2,' ** ') ;
000101  3978   CASE I OF
000103  3979   1 : WRITELN('OCTAL DIGIT EXPECTED'                          )          ;
000112  3980   2 : WRITELN('STRING CONSTANT CANNOT EXCEED SOURCE LINE')               ;
000121  3981   3 : WRITELN('NULL STRING NOT ALLOWED'                        )          ;
000130  3982   4 : WRITELN('ILLEGAL SYMBOL'                                 )          ;
000137  3983   5 : WRITELN('IDENTIFIER EXPECTED'                            )          ;
000146  3984   6 : WRITELN('SKIPPED UNTIL HERE'                              )          ;
000155  3985   7 : WRITELN('"BE" OR "=" EXPECTED'                           )          ;
000164  3986   8 : WRITELN('"=" EXPECTED'                                    )          ;
000173  3987   9 : WRITELN('"VEC" OR "TABLE" EXPECTED'                       )          ;
000202  3988  10 : WRITELN('"(" OR "=" EXPECTED'                            )          ;
000211  3989  11 : WRITELN('DECIMAL/OCTAL CONSTANT EXPECTED'                 )          ;
000220  3990  12 : WRITELN('CONSTANT EXPECTED'                               )          ;
000227  3991  13 : WRITELN('NAME NOT DECLARED'                               )          ;
000236  3992  14 : WRITELN('NAME PREVIOUSLY DECLARED'                        )          ;
000245  3993  15 : WRITELN('"FINISH" ILLEGAL WITHIN PROCEDURE'               )          ;
000254  3994  16 : WRITELN('ILLEGAL LABEL'                                   )          ;
000263  3995  17 : WRITELN('ILLEGAL PLACEMENT OF "FINISH" SYMBOL'           )          ;
000272  3996  18 : WRITELN('IDENTIFIER CANNOT BE USED AS FUNC OR PROC NAME')           ;
000301  3997  19 : WRITELN('EMPTY TABLE NOT ALLOWED'                         )          ;
000310  3998  20 : WRITELN('IDENTIFIER OR CONSTANT EXPECTED'                 )          ;
000317  3999  21 : WRITELN('LABEL PREVIOUSLY DECLARED'                       )          ;
000326  4000  22 : WRITELN('ONLY INTEGER CONSTANTS ALLOWED'                  )          ;
000335  4001  23 : WRITELN('DECIMAL/OCTAL CONSTANT EXCEEDS LIMIT'           )          ;
000344  4002  24 : WRITELN(' ; EXPECTED'                                    )          ;
000353  4003  25 : WRITELN('ILLEGAL PLACEMENT OF "RETURN" COMMAND'          )          ;
000362  4004  26 : WRITELN('ILLEGAL PLACEMENT OF "RESULTIS" COMMAND'       )          ;
000371  4005  27 : WRITELN('COMMAND EXPECTED'                               )          ;
000400  4006  28 : WRITELN('FORMAT FACTOR IN DECIMAL EXPECTED'              )          ;
000407  4007  29 : WRITELN('FORMAT FACTOR MUSTBE <= 10 '                    )          ;
000416  4008  30 : WRITELN('LABEL EXPECTED'                                  )          ;
000425  4009  31 : WRITELN('NAME NOT DECLARED AS LABEL'                      )          ;
000434  4010  32 : WRITELN('VECAP OPERATOR CAN BE APPLIED ONLY TO VARIABLES'  )          ;
000443  4011  33 : WRITELN('LABEL ',IDP1^.NAME,' REFERANCED BUT NOT DECLARED' )          ;
000467  4012  34 : WRITELN('ONLY FUNCTION REFERENCE ALLOWED WITHIN EXPRESSION')          ;
000476  4013  35 : WRITELN('FUNCTION REFERENCE ALLOWED ONLY WITHIN EXPRESSION')          ;
000505  4014  36 : WRITELN('STRING CONSTANT CANNOT EXCEED 10 CHARACTER'       )          ;
000514  4015  38 : WRITELN('COMPILER ERROR . REGISTER UNDERFLOW'              )          ;
000523  4016  39 : BEGIN
000523  4017         WRITELN('EXPRESSION TOO COMPLEX . SIMPLIFY IT . EXECUTION ',
000530  4018                 'TERMINATED SORRY ') ;
000536  4019         HALT ;
000540  4020       END ;
```

```
000541  4021   40 : WRITELN('ILLEGAL OR OUT OF RANGE SHIFT PARAMETER')          ;
000550  4022   41 : WRITELN('LV OPERATOR MEANINGLESS AT LEFT HAND SIDE')        ;
000557  4023   42 : WRITELN('CANNOT CALL THAT PROCEDURE')                       ;
000566  4024   43 : WRITELN('NO PARAMETER EXPECTED ')                          ;
000575  4025   44 : WRITELN('TOO MUCH PARAMETERS')                             ;
000604  4026   45 : WRITELN('PARAMETER EXPECTED')                              ;
000613  4027   46 : WRITELN('CAN NOT CALL THAT FUNCTION')                       ;
000622  4028   47 : WRITELN('PROCEDURE NAME CANNOT APPEAR ON LEFT OF :=')       ;
000631  4029   48 : WRITELN('FUNCTION NAME NOT CURRENT ONE')                    ;
000640  4030   49 : WRITELN('COMPILER ERROR OF SIMPLE TYPE')                    ;
000647  4031   50 : WRITELN('COMPILER ERROR IN GENCODE ROUTINE')                ;
000656  4032   51 : WRITELN('COMPILER ERROR IN INITVALU ROUTINE')               ;
000665  4033    END ; (* CASE *)
000723  4034    LINECNT:=LINECNT+1 ;
000725  4035    IF LINECNT=60 THEN PRINTTITLE ;
000730  4036 END ; (* ERROR *)
001210  4037
001210  4038       (*   S Y M B O L   T A B L E   M A N I P U L A T I O N   *)
001210  4039
001210  4040       (*  THE SYMBOL TABLE IS ORGANIZED AS A STACK-TYPE LIST   *)
001210  4041       (*  OF UNBALANCED BINARY TREES REPRESENTING THE NESTING  *)
001210  4042       (*  LEVEL OF PROCEDURES                                  *)
001210  4043
001210  4044
001210  4045 (*********************************************************************
001210  4046  *                                                                   *
001210  4047  *   THIS ROUTINE SEARCHES A NAME AT THE CURRENT LEVEL OR AT ALL     *
001210  4048  *   LEVELS KEEPING TRACK THE NUMBER OF LEVELS AND RETURNS A         *
001210  4049  *   POINTER TO THAT ENTRY TOGETHER WITH THE LEVEL COUNTER ,         *
001210  4050  *   OTHERWISE IT RETURNS THE VALUE NIL .                            *
001210  4051  *                                                                   *
001210  4052  *********************************************************************)
001210  4053
001210  4054 PROCEDURE SEARCHID ;
000003  4055
000003  4056 VAR P,Q : IDP     ;
000005  4057     FOUND : BOOLEAN ;
000006  4058
000006  4059 BEGIN (* SEARCHID *)
000006  4060   TOPNODE:=FALSE ;
000006  4061   LEVCNT := 0 ;
000007  4062   IF LEVEL = SCLEV
000007  4063     THEN BEGIN
000011  4064            FOUND := FALSE ;
000012  4065            Q := CLEVPTR ;
000014  4066            IF Q ^. NAME = IDSTR
000020  4067              THEN BEGIN
000022  4068                     FOUND:=TRUE ;
000023  4069                     TOPNODE:=TRUE ;
000025  4070                   END ;
000025  4071            WHILE (Q <> NIL) AND (NOT FOUND) DO
000032  4072              BEGIN
000032  4073                IF Q ^. NAME = IDSTR
000036  4074                  THEN FOUND := TRUE
000037  4075                  ELSE IF IDSTR < Q ^. NAME
000045  4076                         THEN Q := Q ^. LLINK
000053  4077                         ELSE Q := Q ^. RLINK
000061  4078              END ;
000063  4079            IF NOT FOUND
000063  4080              THEN IDPTR := NIL
```

```
000065  4081              ELSE IDPTR := Q ;
000071  4082          END
000071  4083      ELSE BEGIN
000072  4084             P := CLEVPTR ;
000074  4085             FOUND := FALSE ;
000075  4086             WHILE (P <> NIL) AND ( NOT FOUND) DO
000101  4087               BEGIN
000101  4088                 Q := P ;
000102  4089                 IF Q ^. NAME = IDSTR
000106  4090                   THEN BEGIN
000110  4091                          FOUND:=TRUE ;
000111  4092                          TOPNODE:=TRUE ;
000112  4093                        END ;
000112  4094                 WHILE (Q <> NIL) AND (NOT FOUND) DO
000117  4095                   BEGIN
000117  4096                     IF Q ^. NAME = IDSTR
000123  4097                       THEN FOUND := TRUE
000124  4098                       ELSE IF IDSTR < Q ^. NAME
000132  4099                              THEN Q := Q ^. LLINK
000140  4100                              ELSE Q := Q ^. RLINK
000146  4101                   END ;
000150  4102                 IF NOT FOUND
000150  4103                   THEN P := PREVLEVPTR(P) ;
000155  4104               END ;
000156  4105             IF NOT FOUND
000156  4106               THEN IDPTR := NIL
000160  4107               ELSE IDPTR := Q ;
000164  4108          END ;
000164  4109 END ; (* SEARCHID *)
000176  4110
000176  4111 (*****************************************************************
000176  4112  *                                                               *
000176  4113  *  THIS ROUTINE ENTERS A NAME EITHER AT THE CURRENT LEVEL       *
000176  4114  *  OR AT A NEW LEVEL BY PUSHING A NEW TREE HEADER INTO THE      *
000176  4115  *  SYMBOL TABLE . IT ALSO RETURNS A POINTER TO THE NEW          *
000176  4116  *  ENTRY MADE .                                                 *
000176  4117  *                                                               *
000176  4118  *****************************************************************)
000176  4119
000176  4120 PROCEDURE ENTERID ;
000003  4121
000003  4122 VAR P,Q,R : ICP     ;
000006  4123
000006  4124 BEGIN (* ENTERID *)
000006  4125   IF LEVEL = ECLEV
000004  4126     THEN BEGIN
000006  4127            P := CLEVPTR ;
000010  4128            WHILE P <> NIL DO
000013  4129              BEGIN
000013  4130                Q := P ;
000014  4131                IF IDSTR < P ^. NAME
000020  4132                  THEN P := P ^. LLINK
000026  4133                  ELSE P := P ^. RLINK ;
000035  4134              END ;
000036  4135            NEW(R) ;
000040  4136            WITH R ^ DO
000045  4137              BEGIN
000045  4138                NAME := IDSTR ;
000047  4139                NEXT := NIL   ;
000052  4140                LLINK := NIL  ;
```

```
000053  4141                RLINK := NIL  ;
000054  4142              END ;
000054  4143            IF IDSTR < Q ^. NAME
000061  4144              THEN Q ^. LLINK := R
000067  4145              ELSE Q ^. RLINK := R ;
000075  4146            CLEVPTR ^. ADDRESS := CLEVPTR ^. ADDRESS + 1 ;
000110  4147          END
000110  4148      ELSE BEGIN
000111  4149             NEW(R) ;
000113  4150             WITH R ^ DO
000120  4151               BEGIN
000120  4152                 ADDRESS := 0 ;
000121  4153                 NAME := IDSTR ;
000123  4154                 NEXT := NIL   ;
000126  4155                 LLINK := NIL ;
000127  4156                 RLINK := NIL ;
000130  4157                 KLASS := PROCFUNC ;
000133  4158                 PARNO := 0 ;
000135  4159                 PPTR := NIL ;
000140  4160               END ;
000140  4161             CLEVPTR ^. NEXT := R ;
000147  4162             CLEVPTR := R ;
000150  4163           END ;
000150  4164    IDPTR := R;
000152  4165 END ; (* ENTERID *)
000164  4166
000164  4167 (*************************************************************
000164  4168  *                                                           *
000164  4169  *  THIS ROUTINE IS USED TO FIND THE POINTER OF THE HEADER   *
000164  4170  *  OF THE PREVIOUS LEVEL TREE .                             *
000164  4171  *                                                           *
000164  4172  *************************************************************)
000164  4173
000164  4174 FUNCTION PREVLEVPTR ;
000006  4175
000006  4176 VAR P , Q : ICP      ;
000010  4177
000010  4178 BEGIN
000010  4179   LEVCNT:=LEVCNT+1 ;
000010  4180   IF R = STSTART
000010  4181     THEN PREVLEVPTR := NIL
000012  4182     ELSE BEGIN
000014  4183            P := STSTART ;
000016  4184            WHILE P <> R DO
000021  4185              BEGIN
000021  4186                Q := P ;
000022  4187                P := P ^. NEXT ;
000027  4188              END ;
000030  4189            PREVLEVPTR := Q ;
000032  4190          END ;
000032  4191 END ; (* PREVLEVPTR *)
000043  4192
000043  4193 (*************************************************************
000043  4194  *                                                           *
000043  4195  *  THIS ROUTINE RELEASES A SPECIFIED NODE FROM THE          *
000043  4196  *  SYMBOL TABLE ADJUSTING ANY LINK REQUIRED .               *
000043  4197  *                                                           *
000043  4198  *************************************************************)
000043  4199
000043  4200 PROCEDURE FREENODE ;
```

```
000003  4201
000003  4202 BEGIN (* FREENODE *)
000003  4203   IF R ^. KLASS = LABL
000010  4204     THEN IF NOT (R ^. DECLARED)
000017  4205            THEN BEGIN
000021  4206                   IDP1 := R ;
000022  4207                   ERROR(33) ;
000024  4208                   GENCODE(R^.LADDR,'J',460,0,0,0,0) ;
000037  4209                 END ;
000037  4210   IF (R ^. LLINK = NIL) AND (R ^. RLINK = NIL)
000051  4211     THEN DISPOSE(R)
000055  4212     ELSE BEGIN
000056  4213            IF R ^. LLINK <> NIL
000062  4214              THEN FREENODE( R ^. LLINK ) ;
000071  4215            IF R ^. RLINK <> NIL
000076  4216              THEN FREENODE( R ^. RLINK ) ;
000105  4217            DISPOSE(R) ;
000107  4218          END ;
000107  4219 END ; (* FREENODE *)
000113  4220
000113  4221 (*************************************************************
000113  4222  *                                                           *
000113  4223  *  RELEASES THE LAST LEVEL IE. THE LAST TREE FROM           *
000113  4224  *  SYMBOL TABLE AND INSERTS ITS HEADER NODE AT THE          *
000113  4225  *  APPROPRIATE PLACE IN THE PREVIOUS LEVEL , IT ALSO        *
000113  4226  *  MODIFIES THE CURRENT LEVEL POINTER TO POINT TO THE       *
000113  4227  *  PREVIOUS LEVEL .                                         *
000113  4228  *                                                           *
000113  4229  *************************************************************)
000113  4230
000113  4231 PROCEDURE FREELEVEL ;
000002  4232
000002  4233 VAR P,Q,R : IDP     ;
000005  4234
000005  4235 BEGIN (* FREELEVEL *)
000005  4236   P := CLEVPTR ;
000006  4237   IF P ^. LLINK <> NIL
000012  4238     THEN FREENODE( P ^. LLINK ) ;
000022  4239   IF P ^. RLINK <> NIL
000027  4240     THEN FREENODE( P ^. RLINK ) ;
000036  4241   IF CLEVPTR=STSTART
000036  4242     THEN DISPOSE(CLEVPTR)
000042  4243     ELSE BEGIN
000043  4244            CLEVPTR := PREVLEVPTR ( CLEVPTR ) ;
000046  4245            CLEVPTR ^. NEXT := NIL ;
000055  4246            P ^. LLINK := NIL ;
000062  4247            P ^. RLINK := NIL ;
000067  4248            P ^. PPTR  := NIL ;
000075  4249            Q := CLEVPTR ;
000076  4250            WHILE Q <> NIL DO
000101  4251              BEGIN
000101  4252                R := Q ;
000102  4253                IF P ^. NAME < Q ^. NAME
000112  4254                  THEN Q := Q ^. LLINK
000120  4255                  ELSE Q := Q ^. RLINK ;
000126  4256              END ;
000127  4257            IF P ^. NAME < R ^. NAME
000137  4258              THEN R ^. LLINK := P
000145  4259              ELSE R ^. RLINK := P ;
000153  4260          END ;
```

```
000153  4261 END ; (* FREELEVEL *)
000163  4262
000163  4263
000163  4264 (*************************************************************
000163  4265  *                                                           *
000163  4266  *  THIS ROUTINE DUMPS THE CONSTANTS TABLE AT THE END        *
000163  4267  *  OF THE ASSEMBLER CODE GENERATED .                        *
000163  4268  *                                                           *
000163  4269  *************************************************************)
000163  4270
000163  4271 PROCEDURE DUMPCONST ;
000002  4272
000002  4273 VAR Q : CTP ; I : INTEGER ;
000004  4274
000004  4275 BEGIN (* DUMPCONST *)
000004  4276 (*  WRITELN(' DUMP OF CONSTANTS TABLE ') ;
000004  4277   WRITELN(' ',CTSTART^.VALU) ; *)
000004  4278   WRITELN(CODE,'C$TABLE   VFD  60/7777777777777777777B') ;
000012  4279   Q := CTSTART ^. NEXT ;
000020  4280   WHILE Q <> NIL DO
000023  4281     BEGIN
000023  4282       CASE Q ^. KLASS OF
000030  4283         INT : BEGIN
000030  4284                 IF Q^.VALU[1]<>' '
000034  4285                   THEN WRITELN(CODE,' ':10,'VFD  60/',Q^.VALU,'B')
000064  4286                   ELSE BEGIN
000066  4287                          I:=1 ;
000067  4288                          WHILE Q^.VALU[I]=' ' DO I:=I+1 ;
000107  4289                          WRITE(CODE,' ':10,'VFD  60/') ;
000120  4290                          FOR I:=I TO 20 DO WRITE(CODE,Q^.VALU[I]) ;
000152  4291                          WRITELN(CODE) ;
000153  4292                        END ;
000153  4293               (* WRITELN(' ',Q ^. VALU) ;  *)
000153  4294                END ;
000154  4295         STR : BEGIN
000154  4296               (* WRITE(' "') ; *)
000154  4297                  WRITE(CODE,' ':10,'VFD  60/',Q^.STRLEN:ZER,'L');
000200  4298                  FOR I:=1 TO Q ^. STRLEN DO
000207  4299                    BEGIN
000211  4300                   (* WRITE(Q ^. STRVALU[I] ) ;*)
000211  4301                      WRITE(CODE,Q ^. STRVALU[I] ) ;
000227  4302                    END ;
000234  4303              (*  WRITELN('"     ',Q ^. STRLEN:ZER) ; *)
000234  4304                  WRITELN(CODE) ;
000235  4305                END ;
000236  4306       END ; (* CASE *)
000242  4307       Q := Q ^. NEXT ;
000247  4308     END ; (* WHILE *)
000250  4309 END ; (* DUMPCONST *)
000263  4310
000263  4311 (*************************************************************
000263  4312  *                                                           *
000263  4313  *  THE FOLLOWING TWO ROUTINES ENABLES THE TRANSLATOR TO     *
000263  4314  *  GENERATE CODE THAT MANIPULATES X-REGISTERS AT RUN-TIME    *
000263  4315  *  X-REGISTERS ARE ORGANIZED AS A STACK (X1,X2,X3,X4,X5)    *
000263  4316  *                                                           *
000263  4317  *  GETREG  : PROVIDES THE TRANSLATOR WITH THE NEXT          *
000263  4318  *            X-REGISTER AVAILABLE .                         *
000263  4319  *  FREEREG : RELEASES THE LAST REGISTER SO THAT IT BECOMES   *
000263  4320  *            AVAILABLE FOR FUTURE USE .                     *
```

```
000263  4321  *                                                             *
000263  4322  ***************************************************************)
000263  4323
000263  4324 PROCEDURE GETREG ;
000002  4325
000002  4326 BEGIN (* GETREG *)
000002  4327   XTOP:=XTOP+1 ;
000006  4328   IF XTOP > 5 THEN ERROR(39) ;
000011  4329 END ; (* GETREG *)
000013  4330
000013  4331 PROCEDURE FREEREG ;
000002  4332
000002  4333 BEGIN (* FREEREG *)
000002  4334   IF XTOP > 0 THEN XTOP:=XTOP-1
000006  4335               ELSE ERROR(38) ;
000012  4336 END ; (* FREEREG *)
000014  4337
000014  4338 (*****************************************************************
000014  4339  *                                                               *
000014  4340  *  THE FOLLOWING ROUTINE IS GENERATING THE ACTUAL ASSEMBLER     *
000014  4341  *  CODES INTO FILE 'CODE' , AND IT IS INVOKED BY THE VARIOUS    *
000014  4342  *  TRANSLATING ROUTINES . IT ALSO CALCULATES THE NUMBER OF      *
000014  4343  *  INSTRUCTION WORDS GENERATED                                  *
000014  4344  *                                                               *
000014  4345  *****************************************************************)
000014  4346
000014  4347 PROCEDURE GENCODE ;
000011  4348
000011  4349 VAR  TEMP : INTEGER ;
000012  4350
000012  4351 BEGIN (* GENCODE *)
000012  4352   IF LAB >= 0
000004  4353     THEN IF LABTYPE IN ['L','P','F','J','C']
000006  4354            THEN BEGIN
000010  4355                   WRITE(CODE,LABTYPE,'$',LAB:ZER) ;
000025  4356                   IF LAB < 10
000025  4357                     THEN WRITE(CODE,' ':6)
000034  4358                     ELSE IF LAB < 100
000035  4359                            THEN WRITE(CODE,' ':5)
000043  4360                            ELSE IF LAB < 1000
000044  4361                                   THEN WRITE(CODE,' ':4)
000052  4362                                   ELSE WRITE(CODE,' ':3)
000057  4363                 END
000057  4364            ELSE ERROR(50)
000061  4365     ELSE IF LAB = -2
000063  4366            THEN WRITE(CODE,'+         ')
000072  4367            ELSE WRITE(CODE,' ':9) ;
000077  4368     IF (LAB>=0) OR (LAB=-2)
000102  4369       THEN BEGIN
000103  4370              TEMP:=(IBANK+3) DIV 4 ;
000106  4371              IBANK:=TEMP*4 ;
000107  4372            END ;
000107  4373
000107  4374   IF (LAB >= 0) OR (LAB = -2)
000112  4375     THEN IBANK:=((IBANK+3) DIV 4)*4 ;
000116  4376
000116  4377   CASE INSTR OF
000120  4378
000120  4379   020 : WRITELN(CODE,'JP    B',I:ZER,'+L$',KK:ZER) ;
000142  4380   021 : WRITELN(CODE,'JP    B',I:ZER,'+0',KK:ZER) ;
```

```
000163  4381    022 : WRITELN(CODE,'JP    B',I:ZER,'+F$',KK:ZER) ;
000205  4382    023 : WRITELN(CODE,'JP    B',I:ZER,'+J$',KK:ZER) ;
000227  4383    024 : WRITELN(CODE,'JP    B',I:ZER,'+P$',KK:ZER) ;
000251  4384    030 : WRITELN(CODE,'ZR    X',J:ZER,',L$',KK:ZER) ;
000273  4385    031 : WRITELN(CODE,'NZ    X',J:ZER,',L$',KK:ZER) ;
000315  4386    032 : WRITELN(CODE,'PL    X',J:ZER,',L$',KK:ZER) ;
000337  4387    033 : WRITELN(CODE,'NG    X',J:ZER,',L$',KK:ZER) ;
000361  4388    034 : ;
000362  4389    035 : ;
000363  4390    036 : ;
000364  4391    037 : ;
000365  4392    040 : WRITELN(CODE,'EQ    B',I:ZER,',B',J:ZER,',L$',KK:ZER) ;
000416  4393    050 : ;
000417  4394    060 : ;
000420  4395    070 : ;
000421  4396    100 : WRITELN(CODE,'BX',I:ZER,'  X',J:ZER) ;
000443  4397    110 : WRITELN(CODE,'BX',I:ZER,'  X',J:ZER,'*X',K:ZER) ;
000474  4398    120 : WRITELN(CODE,'BX',I:ZER,'  X',J:ZER,'+X',K:ZER) ;
000525  4399    130 : WRITELN(CODE,'BX',I:ZER,'  X',J:ZER,'-X',K:ZER) ;
000556  4400    140 : WRITELN(CODE,'BX',I:ZER,'  -X',K:ZER) ;
000600  4401    150 : ;
000601  4402    160 : ;
000602  4403    170 : ;
000603  4404    200 : WRITELN(CODE,'LX',I:ZER,'  ',KK:ZER) ;
000624  4405    210 : ;
000625  4406    220 : WRITELN(CODE,'LX',I:ZER,'  B',J:ZER) ;
000647  4407    230 : ;
000650  4408    240 : ;
000651  4409    250 : ;
000652  4410    260 : ;
000653  4411    270 : ;
000654  4412    300 : ;
000655  4413    310 : ;
000656  4414    320 : ;
000657  4415    330 : ;
000660  4416    340 : ;
000661  4417    350 : ;
000662  4418    360 : WRITELN(CODE,'IX',I:ZER,'  X',J:ZER,'+X',K:ZER) ;
000713  4419    370 : WRITELN(CODE,'IX',I:ZER,'  X',J:ZER,'-X',K:ZER) ;
000744  4420    400 : ;
000745  4421    410 : ;
000746  4422    420 : WRITELN(CODE,'IX',I:ZER,'  X',J:ZER,'*X',K:ZER) ;
000777  4423          (* MACRO FOR DIVISION . REMEMBER IT DESTROYS REGS XJ,XK,B7 *)
000777  4424    810 : WRITELN(CODE,'IX',I:ZER,'  X',J:ZER,'/X',K:ZER) ;
001030  4425    430 : ;
001031  4426    440 : ;
001032  4427    450 : ;
001033  4428    460 : WRITELN(CODE,'NO') ;
001042  4429    470 : ;
001043  4430    500 : WRITELN(CODE,'SA',I:ZER,'  A',J:ZER,'+0',KK:ZER) ;
001074  4431    510 : WRITELN(CODE,'SA',I:ZER,'  B',J:ZER,'+0',KK:ZER) ;
001125  4432    520 : WRITELN(CODE,'SA',I:ZER,'  X',J:ZER,'+0',KK:ZER) ;
001156  4433    530 : WRITELN(CODE,'SA',I:ZER,'  X',J:ZER,'+B',K:ZER) ;
001207  4434    540 : WRITELN(CODE,'SA',I:ZER,'  A',J:ZER,'+B',K:ZER) ;
001240  4435    550 : WRITELN(CODE,'SA',I:ZER,'  A',J:ZER,'-B',K:ZER) ;
001271  4436    560 : WRITELN(CODE,'SA',I:ZER,'  B',J:ZER,'+B',K:ZER) ;
001322  4437    570 : WRITELN(CODE,'SA',I:ZER,'  B',J:ZER,'-B',K:ZER) ;
001353  4438    600 : WRITELN(CODE,'SB',I:ZER,'  A',J:ZER,'+0',KK:ZER) ;
001404  4439    610 : WRITELN(CODE,'SB',I:ZER,'  B',J:ZER,'+0',KK:ZER) ;
001435  4440    620 : WRITELN(CODE,'SB',I:ZER,'  X',J:ZER,'+0',KK:ZER) ;
```

```
001466  4441   630 : WRITELN(CODE,'SB',I:ZER,'  X',J:ZER,'+B',K:ZER) ;
001517  4442   640 : WRITELN(CODE,'SB',I:ZER,'  A',J:ZER,'+B',K:ZER) ;
001550  4443   650 : WRITELN(CODE,'SB',I:ZER,'  A',J:ZER,'-B',K:ZER) ;
001601  4444         (* CR INSTRUCTION HERE . IT HAS SAME CODE WITH NEXT ONE ? *)
001601  4445   660 : WRITELN(CODE,'SB',I:ZER,'  B',J:ZER,'+B',K:ZER) ;
001632  4446   670 : WRITELN(CODE,'SB',I:ZER,'  B',J:ZER,'-B',K:ZER) ;
001663  4447   700 : WRITELN(CODE,'SX',I:ZER,'  A',J:ZER,'+0',KK:ZER) ;
001714  4448   710 : WRITELN(CODE,'SX',I:ZER,'  B',J:ZER,'+0',KK:ZER) ;
001745  4449   720 : WRITELN(CODE,'SX',I:ZER,'  X',J:ZER,'+0',KK:ZER) ;
001776  4450   730 : WRITELN(CODE,'SX',I:ZER,'  X',J:ZER,'+B',K:ZER) ;
002027  4451   740 : WRITELN(CODE,'SX',I:ZER,'  A',J:ZER,'+B',K:ZER) ;
002060  4452   750 : WRITELN(CODE,'SX',I:ZER,'  A',J:ZER,'-B',K:ZER) ;
002111  4453   760 : WRITELN(CODE,'SX',I:ZER,'  B',J:ZER,'+B',K:ZER) ;
002142  4454   770 : WRITELN(CODE,'SX',I:ZER,'  B',J:ZER,'-B',K:ZER) ;
002173  4455         (* FURTHER SPECIAL PURPOSE INSTRUCTIONS *)
002173  4456   800 : WRITELN(CODE,'SA',I:ZER,'  C$TABLE+',KK:ZER) ;
002215  4457   801 : WRITELN(CODE,'SA',I:ZER,'  R$STACK+',KK:ZER) ;
002237  4458   804 : WRITELN(CODE,'RJ   R$ENTRY') ;
002246  4459   805 : WRITELN(CODE,'RJ   R$EXIT') ;
002255  4460   806 : WRITELN(CODE,'RJ   SAV$REG') ;
002264  4461   807 : WRITELN(CODE,'SX',I:ZER,'  *+0',KK:ZER) ;
002306  4462   808 : WRITELN(CODE,'RJ   RES$REG') ;
002315  4463   809 : WRITELN(CODE,'SA',I:ZER,'  F$TOP') ;
002334  4464   811 : WRITELN(CODE,'RJ   FOR$2') ;
002343  4465   812 : WRITELN(CODE,'RJ   FOR$3') ;
002352  4466   813 : WRITELN(CODE,'RJ   FOR$4') ;
002361  4467   814 : WRITELN(CODE,'SA',I:ZER,'  X',J:ZER,'+F$STACK+0',KK:ZER) ;
002413  4468   815 : WRITELN(CODE,'RJ   FOR$1') ;
002422  4469   816 : WRITELN(CODE,'NZ   X',J:ZER,',C$',KK:ZER) ;
002444  4470   817 : WRITELN(CODE,'RJ   CASE$ER');
002453  4471   818 : WRITELN(CODE,'RJ   WR$LN');
002462  4472   819 : WRITELN(CODE,'RJ   RD$LN');
002471  4473   820 : WRITELN(CODE,'RJ   RD$CH');
002500  4474   821 : WRITELN(CODE,'RJ   WR$NUM');
002507  4475   822 : WRITELN(CODE,'RJ   WR$STR');
002516  4476   823 : WRITELN(CODE,'SA',I:ZER,'  STACK$') ;
002535  4477   824 : WRITELN(CODE,'LT   B',I:ZER,',B',J:ZER,',SHF$ERR') ;
002563  4478   825 : WRITELN(CODE,'GT   B',J:ZER,',B',I:ZER,',SHF$ERR') ;
002611  4479   826 : WRITELN(CODE,'RJ   WR$O') ;
002620  4480   827 : WRITELN(CODE,'JP   HALT$') ;
002627  4481
002627  4482   OTHERWISE ERROR(50) ;
003461  4483
003461  4484   END ; (* CASE *)
003461  4485
003461  4486   CASE INSTR OF
003463  4487
003463  4488   020,021,022,023,024,030,031,032,033,040,
003463  4489   200,500,510,520,600,610,620,700,710,720,
003463  4490   800,801,804,805,806,807,808,809,811,812,
003463  4491   813,814,815,816,817,818,819,820,821,822,
003463  4492   823,824,825,826,827                      :
003463  4493
003463  4494     IF (((IBANK+3) DIV 4)*4 - IBANK) = 1
003467  4495       THEN IBANK:=(((IBANK+3) DIV 4)*4) + 2
003472  4496       ELSE IBANK:=IBANK+2 ;
003476  4497
003476  4498   OTHERWISE  IBANK:=IBANK+1 ;
004330  4499
004330  4500   CASE INSTR OF
```

```
004332  4501       804,805,806,808,811,812,813,815,817,
004332  4502       818,819,820,821,822,826,827 :
004332  4503         IBANK:=((IBANK+3) DIV 4)*4 ;
004336  4504
004336  4505       OTHERWISE ;
004356  4506     END ;
004356  4507
004356  4508     END ; (* CASE *)
004356  4509 END ; (* GENCODE *)
004501  4510
004501  4511 (*  THIS ROUTINE GENERATES CODE TO ACCESS A VARIABLE AT RUN-TIME  *)
004501  4512 (*  FROM A DATA SEGMENT THAT IS NOT THE CURRENT ONE , IE IT IS A  *)
004501  4513 (*  GLOBAL ONE TO THAT DATA SEGMENT .                             *)
004501  4514
004501  4515 PROCEDURE COMPUTELEVEL ;
000003  4516
000003  4517 VAR I : INTEGER ;
000004  4518
000004  4519 BEGIN (* COMPUTELEVEL *)
000004  4520   IF L = 0
000004  4521     THEN GENCODE(-1,' ',660,6,3,0,0)
000014  4522     ELSE BEGIN
000016  4523            GETREG ;
000017  4524            GENCODE(-1,' ',560,XTOP,3,0,0) ;
000027  4525            FOR I:=2 TO L DO GENCODE(-1,' ',530,XTOP,XTOP,0,0) ;
000050  4526            GENCODE(-1,' ',630,6,XTOP,0,0) ;
000060  4527            FREEREG ;
000061  4528          END ;
000061  4529 END ; (* COMPUTELEVEL *)
000067  4530
000067  4531 (*  THIS ROUTINE ENTERS RELATED ROUTINES FROM LIBRARY FILE NAMELY  *)
000067  4532 (*  'BCPLLIB' , AT THE END OF THE ASSEMBLER CODE GENERATED .       *)
000067  4533
000067  4534 PROCEDURE ENTERLIB ;
000002  4535
000002  4536 VAR LINE : PACKED ARRAY [1..70] OF CHAR ;
000011  4537     I,J,ROUTINESIZE : INTEGER ;
000014  4538     CH : CHAR ;
000015  4539
000015  4540   PROCEDURE COPYLIBROUTINE ;
000002  4541
000002  4542   VAR I : INTEGER ;
000003  4543
000003  4544   BEGIN (* COPYLIBROUTINE *)
000003  4545     READ(BCPLLIB,LINE[1]) ;
000014  4546     REPEAT
000014  4547       I := 2 ;
000016  4548       WHILE (NOT EOLN(BCPLLIB)) AND (I<=70) DO
000021  4549         BEGIN
000021  4550           READ(BCPLLIB,LINE[I]) ;
000041  4551           I:=I+1 ;
000043  4552         END ;
000044  4553       FOR I:=I TO 70 DO LINE[I]:=' ' ;
000066  4554       FOR I:=1 TO 70 DO WRITE(CODE,LINE[I]) ;
000114  4555       WRITELN (CODE) ;
000115  4556       READLN(BCPLLIB) ;
000117  4557       READ(BCPLLIB,LINE[1]) ;
000127  4558     UNTIL (LINE[1] = '/') OR EOF(BCPLLIB) ;
000134  4559   END ; (* COPYLIBROUTINE *)
000140  4560
```

```
000140  4561 BEGIN (* ENTERLIB *)
000140  4562   WRITELN(CODE,'***** BCPLLIB ROUTINES FOLLOW *****') ;
000012  4563   READLN(BCPLLIB) ;
000014  4564   READ(BCPLLIB,LINE[1]) ;
000024  4565   FOR J:=1 TO 6 DO
000026  4566     BEGIN
000030  4567       IF LIB[J]
000033  4568         THEN
000035  4569         BEGIN
000035  4570           I:=2 ;
000036  4571           WHILE (NOT EOLN(BCPLLIB)) AND (I<=70) DO
000041  4572             BEGIN
000041  4573               READ(BCPLLIB,LINE[I]) ;
000060  4574               I:=I+1 ;
000062  4575             END ;
000063  4576           FOR I:= I TO 70 DO LINE[I]:=' ' ;
000105  4577           I:=2 ;
000106  4578           WHILE LINE[I]=' ' DO I:=I+1 ;
000122  4579           ROUTINESIZE:=0 ;
000124  4580           WHILE ( LINE[I]>='0' ) AND ( LINE[I]<='9' ) DO
000150  4581             BEGIN
000150  4582               ROUTINESIZE:=ROUTINESIZE*10+ORD(LINE[I])-ORD('0') ;
000164  4583               I:=I+1 ;
000165  4584             END ;
000166  4585           LIBSIZE:=LIBSIZE+ROUTINESIZE ;
000170  4586           READLN(BCPLLIB) ;
000172  4587           COPYLIBROUTINE ;
000174  4588         END
000174  4589         ELSE
000175  4590         BEGIN
000175  4591           READLN(BCPLLIB) ;
000177  4592           READ(BCPLLIB,CH) ;
000205  4593           WHILE (CH <> '/')  DO
000210  4594             BEGIN
000210  4595               READLN(BCPLLIB) ;
000211  4596               IF EOF(BCPLLIB)
000211  4597                 THEN CH:='/'
000213  4598                 ELSE READ(BCPLLIB,CH) ;
000223  4599             END ;
000224  4600           LINE[1]:= CH ;
000230  4601         END ;
000230  4602     END ; (* FOR *)
000235  4603   I:=2 ;
000236  4604   WHILE (NOT EOLN(BCPLLIB)) AND (I<=70) DO
000241  4605     BEGIN
000241  4606       READ(BCPLLIB,LINE[I]) ;
000260  4607       I:=I+1 ;
000262  4608     END ;
000263  4609   FOR I:= I TO 70 DO LINE[I]:=' ' ;
000305  4610   I:=2 ;
000306  4611   WHILE LINE[I]=' ' DO I:=I+1 ;
000322  4612   ROUTINESIZE:=0 ;
000324  4613   WHILE ( LINE[I]>='0' ) AND ( LINE[I]<='9' ) DO
000350  4614     BEGIN
000350  4615       ROUTINESIZE:=ROUTINESIZE*10+ORD(LINE[I])-ORD('0') ;
000364  4616       I:=I+1 ;
000365  4617     END ;
000366  4618   LIBSIZE:=LIBSIZE+ROUTINESIZE ;
000370  4619   READLN(BCPLLIB) ;
000372  4620   COPYLIBROUTINE ;
```

```
000374  4621 END ; (* ENTERLIB *)
000412  4622
000412  4623 (*  THIS ROUTINE GENERATES CODE TO INITIALIZE VARIABLES AT  *)
000412  4624 (*  RUN-TIME UPON ENTERING A PROCEDURE .                    *)
000412  4625
000412  4626 PROCEDURE INITVALU ;
000002  4627
000002  4628   PROCEDURE FOLLOWIDP(P:IDP) ;
000003  4629
000003  4630   VAR Q : CTP ; I : INTEGER ;
000005  4631
000005  4632   BEGIN (* FOLLOWIDP *)
000005  4633     IF P^.KLASS = VARS
000010  4634       THEN
000013  4635       BEGIN
000013  4636         IF (NOT (P^.KLASS=PROCFUNC)) AND (NOT P^.TAB) AND (NOT P^.VEC)
000033  4637           THEN
000033  4638           BEGIN
000033  4639             IF P^.VALLINK <> NIL
000037  4640               THEN
000041  4641               BEGIN
000041  4642                 GENCODE(-1,' ',800,1,0,0,P^.VALLINK^.ADDR) ;
000060  4643                 GENCODE(-1,' ',100,6,1,0,0) ;
000070  4644                 GENCODE(-1,' ',510,6,3,0,P^.VADDR+3) ;
000105  4645               END
000105  4646           END
000105  4647           ELSE
000106  4648             IF P^.VEC
000112  4649               THEN
000114  4650               BEGIN
000114  4651                 GENCODE(-1,' ',710,6,3,0,P^.VADDR+4) ;
000130  4652                 GENCODE(-1,' ',510,6,3,0,P^.VADDR+3) ;
000145  4653               END
000145  4654               ELSE
000146  4655                 IF P^.TAB
000152  4656                   THEN
000154  4657                   BEGIN
000154  4658                     GENCODE(-1,' ',710,6,3,0,P^.VADDR+4) ;
000170  4659                     GENCODE(-1,' ',510,6,3,0,P^.VADDR+3) ;
000205  4660                     Q:=P^.VALLINK ;
000213  4661                     I:=1 ;
000214  4662                     WHILE (Q <> NIL) AND (I < P^.NOOFELEM) DO
000225  4663                       BEGIN
000225  4664                         GENCODE(-1,' ',800,1,0,0,Q^.ADDR) ;
000240  4665                         GENCODE(-1,' ',100,6,1,0,0) ;
000250  4666                         GENCODE(-1,' ',510,6,3,0,P^.VADDR+3+I) ;
000266  4667                         Q:=Q^.NEXT ;
000273  4668                         I:=I+1 ;
000275  4669                       END ;
000276  4670                     IF I = P^.NOOFELEM
000302  4671                       THEN (* CORRECT *)
000304  4672                       ELSE ERROR(51) ;
000307  4673                   END ;
000307  4674       END ;
000307  4675     IF P^.LLINK <> NIL
000314  4676       THEN FOLLOWIDP(P^.LLINK) ;
000323  4677     IF P^.RLINK <> NIL
000330  4678       THEN FOLLOWIDP(P^.RLINK) ;
000337  4679   END ; (* FOLLOWIDP *)
000347  4680
```

```
000347  4681 BEGIN (* INITVALU *)
000347  4682   IF CLEVPTR^.LLINK <> NIL
000011  4683     THEN FOLLOWIDP(CLEVPTR^.LLINK) ;
000020  4684   IF CLEVPTR^.RLINK <> NIL
000025  4685     THEN FOLLOWIDP(CLEVPTR^.RLINK) ;
000034  4686 END ; (* INITVALU *)
000036  4687
000036  4688
000036  4689
000036  4690
000036  4691
000036  4692
000036  4693
000036  4694
000036  4695
000036  4696
000036  4697
000036  4698
000036  4699
000036  4700
000036  4701
000036  4702
000036  4703     (***********************************
000036  4704      *                                 *
000036  4705      *    M A I N     P R O G R A M    *
000036  4706      *                                 *
000036  4707      ***********************************)
000036  4708
000036  4709
000036  4710
000036  4711 BEGIN (*  M A I N     P R O G R A M   *)
000036  4712   LSYS:=[SEMICOLON,FINISHSY] ;
000040  4713   INITIALIZE ;
000041  4714   PRINTTITLE ;
000042  4715   TIME1 := CLOCK ;
000045  4716   WRITELN(CODE,'          IDENT  BCPL') ;
000053  4717   WRITELN(CODE,'          ENTRY  BCPL$') ;
000061  4718   SCANNER ;
000062  4719   BLOCK(LSYS) ;
000064  4720   WRITE('      ',CARDCNT:5,'     ');
000102  4721   FOR I:= 1 TO 80 DO WRITE(INBUF[I]) ;
000130  4722   WRITELN ;
000131  4723   WRITELN ;
000133  4724   WRITELN('1PROGRAM CONTAINS ',CARDCNT:ZER,' LINES AND ',PROCADDR+FUNCADDR
000150  4725           :ZER,' PROCEDURES') ;
000162  4726   GOTO 3333 ;
000163  4727 1111 :
000163  4728   WRITELN ;
000165  4729   WRITELN ;
000167  4730   WRITELN(' NO PROGRAM FOUND *** COMPILATION STOPPED') ;
000175  4731   GOTO 3333 ;
000176  4732 2222 :
000176  4733   WRITELN ;
000200  4734   WRITELN ;
000202  4735   IF UNCLOSED
000202  4736     THEN WRITELN('1PROGRAM CONTAINS UNCLOSED STRING')
000211  4737     ELSE WRITE('1') ;
000220  4738   WRITELN(' ECF FOUND DURING COMPILATION *** COMPILATION ABORTED') ;
000226  4739   WRITELN(' PROGRAM CONTAINS ',CARDCNT:ZER,' LINES AND ',PROCADDR+FUNCADDR
000243  4740           :ZER,' PROCEDURES') ;
```

```
000255  4741 3333 :
000255  4742   TIME2 := CLOCK ;
000260  4743   TIME1 := ( TIME2 - TIME1 ) / 1000 ;
000266  4744   IBANK := ((IBANK+3) DIV 4) * 4 +8 ;
000272  4745   WRITELN('   PROGRAM STATISTICS :') ;
000300  4746   WRITELN('    ** INSTRUCTIONS     : ',IBANK DIV 4:6,' WORDS') ;
000320  4747   WRITELN('    ** RUN TIME  STACK  : ',DSTSIZE    :6,' WORDS') ;
000337  4748   WRITELN('    ** CONSTANTS TABLE  : ',CONSTADDR+1:6,' WORDS') ;
000356  4749   WRITELN('    ** FOR LOOP  STACK  : ',FSTSIZE    :6,' WORDS') ;
000375  4750   WRITELN('    ** REGISTER  STACK  : ',RSTSIZE    :6,' WORDS') ;
000414  4751   WRITELN('    ** PROC-NAME STACK  : ',NSTSIZE    :6,' WORDS') ;
000433  4752   WRITELN(CODE,'          ENDRUN') ;
000441  4753   WRITELN(CODE,'F$STACK  BSS   ',FSTSIZE:ZER) ;
000452  4754   WRITELN(CODE,'F$MAX    EQU   *-F$STACK') ;
000460  4755   WRITELN(CODE,'F$TOP    BSSZ  1') ;
000466  4756   WRITELN(CODE,'R$STACK  BSS   ',RSTSIZE:ZER) ;
000477  4757   WRITELN(CODE,'R$MAX    EQU   *-R$STACK') ;
000505  4758   WRITELN(CODE,'R$TOP    BSSZ  1') ;
000513  4759   WRITELN(CODE,'N$STACK  BSS   ',NSTSIZE:ZER) ;
000524  4760   WRITELN(CODE,'N$MAX    EQU   *-N$STACK') ;
000532  4761   WRITELN(CODE,'N$TOP    BSSZ  1') ;
000540  4762   WRITELN(CODE,'STACK$   BSS   ',DSTSIZE:ZER) ;
000551  4763   WRITELN(CODE,'LEN$     EQU   *-1') ;
000557  4764   WRITELN(CODE,'BLK$     DATA  H*          *') ;
000565  4765   WRITELN(CODE,'MASK$    VFD   60/77000000000000000000B') ;
000573  4766   WRITELN(CODE,'OBF$     BSS   301B') ;
000601  4767   WRITELN(CODE,'IBF$     BSS   301B') ;
000607  4768   WRITELN(CODE,'INPUT    FILEC IBF$,301B') ;
000615  4769   WRITELN(CODE,'OUTPUT   FILEC OBF$,301B') ;
000623  4770   DUMPCONST ;
000624  4771   ENTERLIB ;
000625  4772   WRITELN('    ** LIBRARY   SIZE   : ',LIBSIZE    :6,' WORDS') ;
000644  4773   WRITE  ('   TOTAL PROGRAM SIZE   : ') ;
000651  4774   WRITE  (IBANK DIV 4+DSTSIZE+CONSTADDR+FSTSIZE+RSTSIZE+NSTSIZE+1+
000657  4775           LIBSIZE:6) ;
000663  4776   WRITELN(' WORDS') ;
000671  4777   WRITELN(' END BCPL *** LEVEL 1 *** ',TOTERR:ZER,' ERRORS FOUND ',
000705  4778           'DURING COMPILATION . TIME USED : ',TIME1:7:3,' SECONDS') ;
000725  4779   WRITELN ;
000727  4780   WRITELN(CODE,'          END  BCPL$') ;
000735  4781   WRITELN ;
000737  4782 (*   IF TOTERR>0 THEN REWRITE(CODE) ;  *)
000737  4783 END   (*  M A I N     P R O G R A M   *) .

          COMPILER-ESTIMATED 'W' OPTION = 005426B.


17.12.21.UCLP, AA, P04     ,      5.062KLNS.
```

APPENDIX C. LISTING OF THE BCPL LIBRARY

```
WR$LN    BSSZ   1
*
*  THIS IS A ROUTINE TO OUTPUT A LINE FROM OBUF$ TO PRINTER
*
*  SAVE REGISTERS B1,B2,B3,B4,B5
*
         SX6    B1
         SA6    B$12345
         SX6    B2
         SA6    A6+B1
         SX6    B3
         SA6    A6+B1
         SX6    B4
         SA6    A6+B1
*
*  DUMP THE BUFFER OBUF$ TO FILE OUTPUT
*
         WRITEH OUTPUT,OBUF$,OBUF$L
         WRITER OUTPUT
*
*  RESTORE REGISTERS B1,B2,B3,B4,B5
*
         SA1    B$12345
         SB1    X1
         SA1    A1+B1
         SB2    X1
         SA1    A1+B1
         SB3    X1
         SA1    A1+B1
         SB4    X1
*
*  SET OWRD$ TO ZERO , BLANK FILL OUTPUT BUFFER
*
         SX6    B0
         SA6    OWRD$
         SA6    OBYTE$
         SA1    BLK$
         BX6    X1
         SA6    OBUF$
         SA6    A6+B1
         SA6    A6+B1
         SA6    A6+B1
         SA6    A6+B1
         SA6    A6+B1
         SA6    A6+B1
         SA6    A6+B1
         SA6    A6+B1
         SA6    A6+B1
         SA6    A6+B1
         SA6    A6+B1
*
*  RETURN
*
         EQ     WR$LN
B$12345  BSS    4
OBUF$L   EQU    13
OBUF$    DATA   H*          *
```

```
59.                DATA    H*              *
60.                DATA    H*              *
61.                DATA    H*              *
62.                DATA    H*              *
63.                DATA    H*              *
64.                DATA    H*              *
65.                DATA    H*              *
66.                DATA    H*              *
67.                DATA    H*              *
68.                DATA    H*              *
69.                DATA    H*              *
70.                DATA    H*              *
71.      OWRD$     BSSZ    1
72.      OBYTE$    BSSZ    1
73.      WR$NUM    BSSZ    1
74.      *
75.      *  ROUTINE TO CONVERT AN INTEGER TO A STRING AND PRINT IT
76.      *  X1  CONTAINS THE INTEGER TO BE CONVERTED
77.      *
78.      *  SAVE  X1,B2,B3,B4
79.      *
80.                BX6     X1
81.                SA6     X$1
82.                SX6     B2
83.                SA6     B$234
84.                SX6     B3
85.                SA6     A6+B1
86.                SX6     B4
87.                SA6     A6+B1
88.      *
89.      *  IS IT A NEGATIVE INTEGER ?
90.      *
91.                PL      X1,WR$11
92.                BX6    -X1              X1<0 , MAKE IT POSITIVE AND
93.                SA6     X$1
94.                SA1     MINUS$
95.                SX2     B1
96.                RJ      WR$STR          OUTPUT A MINUS SIGN
97.      *
98.      *  IS IT A LARGE INTEGER ( GT 10000000000 ) ?
99.      *
100.
101.     WR$11      SA1     X$1
102.               BX2     X1
103.               AX2     10
104.               MX3     56
105.               BX3    -X3
106.               BX6     X3*X2
107.               AX2     4
108.               BX3     X2
109.               SA4     DIV$
110.               IX3     X3/X4
111.               SA4     DIV$
112.               IX4     X4*X3
113.               IX7     X2-X4
114.               LX3     4
115.               LX7     4
116.               IX4     X6+X7
```

```
117.                SA5     DIV$
118.                IX5     X4/X5
119.                IX2     X3+X5
120.                ZR      X2,WR$5
121.                SA3     DIV$
122.                IX3     X3*X2
123.                LX3     10
124.                IX6     X1-X3
125.                SA6     X$1
126.                BX1     X2
127.                RJ      =XCDD
128.                BX1     X4
129.                SX3     B2
130.                SX6     6
131.                IX2     X3/X6
132.                RJ      WR$STR
133.                SA1     X$1
134.                RJ      =XCDD
135.                BX1     X6
136.                SX2     B2
137.                SX6     6
138.                IX2     X2/X6
139.                SX6     B1
140.                IX2     X2-X6
141.                SA3     MZER$+X2
142.                SA4     ZER$
143.                BX4     X4*X3
144.                BX3     -X3
145.                BX1     X1*X3
146.                BX1     X1+X4
147.                SX2     10
148.                RJ      WR$STR
149.                JP      RES$
150.    *
151.    *  A SHORT INTEGER ( LE 10000000000 )
152.    *
153.    WR$5        SA1     X$1
154.                RJ      =XCDD
155.                BX1     X4
156.                SX3     B2
157.                SX4     6
158.                IX2     X3/X4
159.                RJ      WR$STR          OUTPUT THE NUMBER
160.    RES$        SA1     B$234
161.                SB2     X1
162.                SA1     A1+B1
163.                SB3     X1
164.                SA1     A1+B1
165.                SB4     X1
166.                EQ      WR$NUM
167.    X$1         BSS     1
168.    B$234       BSS     3
169.    DIV$        VFD     60/9765625
170.    MINUS$      DATA    H*-             *
171.    ZER$        VFD     60/3333333333333333333B
172.    MZER$       VFD     60/777777777777777700B
173.                VFD     60/777777777777770000B
174.                VFD     60/777777777777000000B
```

```
175.                VFD     60/7777777777700000000B
176.                VFD     60/7777777770000000000B
177.                VFD     60/7777777000000000000B
178.                VFD     60/7777700000000000000B
179.                VFD     60/7770000000000000000B
180.                VFD     60/7700000000000000000B
181.                VFD     60/0000000000000000000B
182.    WR$0        BSSZ    1
183.    *
184.    *  ROUTEINE TO CONVERT REGISTER X1 TO OCTAL AND PRINT IT
185.    *  WITH A SUFFIX 'B'
186.    *
187.                BX6     X1
188.                SA6     X$1
189.                SX6     B2
190.                SA6     B$234
191.                SX6     B3
192.                SA6     A6+B1
193.                SX6     B4
194.                SA6     A6+B1
195.                MX2     30
196.                BX3     X2*X1
197.                LX3     30
198.                BX2     -X2
199.                BX6     X2*X1
200.                SA6     X$1
201.                BX1     X3
202.                RJ      =XCOD
203.                BX1     X6
204.                SX2     B2
205.                SX6     6
206.                IX2     X2/X6
207.                SX6     B1
208.                IX2     X2-X6
209.                SA3     MZER$+X2
210.                SA4     ZER$
211.                BX4     X4*X3
212.                BX3     -X3
213.                BX1     X1*X3
214.                BX1     X1+X4
215.                SX2     10
216.                RJ      WR$STR
217.                SA1     X$1
218.                RJ      =XCOD
219.                BX1     X6
220.                SX2     B2
221.                SX6     6
222.                IX2     X2/X6
223.                SX6     B1
224.                IX2     X2-X6
225.                SA3     MZER$+X2
226.                SA4     ZER$
227.                BX4     X4*X3
228.                BX3     -X3
229.                BX1     X1*X3
230.                BX1     X1+X4
231.                SX2     10
232.                RJ      WR$STR
```

```
233.                SA1     OCT$
234.                SX2     B1
235.                RJ      WR$STR
236.                SA1     B$234
237.                SB2     X1
238.                SA1     A1+B1
239.                SB3     X1
240.                SA1     A1+B1
241.                SB4     X1
242.                JP      WR$0
243.    OCT$        DATA    H*B             *
244.    WR$STR      BSSZ    1
245.    *
246.    *  THIS IS A ROUTINE TO PLACE A STRING INTO THE OUTPUT BUFFER
247.    *
248.    *  X1  CONTAINS THE STRING
249.    *  X2  CONTAINS THE FORMAT DESIGNATOR
250.    *
251.                SA3     OBYTE$
252.                SX4     10
253.                IX5     X3-X4
254.                MI      X5,WR$1
255.                SA3     OWRD$
256.                SX6     X3+B1
257.                SA6     OWRD$
258.                SX4     13
259.                IX5     X6-X4
260.                MI      X5,WR$2
261.                BX6     X1
262.                SA6     X$12
263.                BX6     X2
264.                SA6     A6+B1
265.                RJ      WR$LN
266.                SA1     X$12
267.                SA2     A1+B1
268.    WR$2        SX6     B0
269.                SA6     OBYTE$
270.    WR$1        SA3     MASK$
271.                BX6     X3
272.                BX3     X3*X1
273.                SA4     OBYTE$
274.                SX5     6
275.                IX4     X5*X4
276.                LX1     6
277.                ZR      X4,WR$3
278.                SB6     X4
279.                SB6     B0-B6
280.                SB6     B6+60
281.                LX3     B6
282.                LX6     B6
283.    WR$3        BX6     -X6
284.                SA4     OWRD$
285.                SB7     X4
286.                SA4     OBUF$+B7
287.                BX6     X4*X6
288.                BX6     X3+X6
289.                SA6     A4
290.                SX3     B1
```

```
291.                IX2     X2-X3
292.                SA4     OBYTE$
293.                IX6     X4+X3
294.                SA6     OBYTE$
295.                ZR      X2,WR$STR
296.                SX3     10
297.                IX4     X6-X3
298.                MI      X4,WR$1
299.                SA3     OWRD$
300.                SX4     B1
301.                IX6     X4+X3
302.                SA6     A3
303.                SX3     13
304.                IX4     X6-X3
305.                MI      X4,WR$4
306.                BX6     X1
307.                SA6     X$12
308.                BX6     X2
309.                SA6     A6+B1
310.                RJ      WR$LN
311.                SA1     X$12
312.                SA2     A1+B1
313.    WR$4        SX6     B0
314.                SA6     OBYTE$
315.                JP      WR$1
316.    X$12        BSS     2
317.                SST
318.    RELOPL      XTEXT   COMCCIO
319.    RELOPL      XTEXT   COMCCDO
320.    RELOPL      XTEXT   COMCWTH
321.    RELOPL      XTEXT   COMCCOD
```

```
 1.    RD$CH     BSSZ   1
 2.    *
 3.    *  THIS RCUTINE GETS THE NEXT CHARACTER FROM IBUF$ AND PLACES
 4.    *  IT INTO REGISTER X1 , IF NO NEXT CHARACTER AVAILABLE THEN
 5.    *  FILLS IBUF$ BY CALLING RD$LN AND THEN GETS THE FIRST CHARACTER
 6.    *  TO PLACE IN X1 .
 7.    *
 8.              SA2    IBYTE$
 9.              SX3    10
10.              IX4    X2-X3
11.              MI     X4,RD$1
12.              SX6    B0
13.              SA6    IBYTE$
14.              SA2    IWRD$
15.              SX6    X2+B1
16.              SA6    A2
17.              SX2    8
18.              IX3    X6-X2
19.              MI     X3,RD$1
20.              RJ     RD$LN
21.    RD$1      SA2    IWRD$
22.              SB6    X2
23.              SA2    B6+IBUF$
24.              SA3    MASK$
25.              SA4    IBYTE$
26.              ZR     X4,RD$2
27.              SX5    6
28.              IX4    X4*X5
29.              SB7    X4
30.              LX2    B7
31.    RD$2      BX1    X2*X3
32.              SA2    IBYTE$
33.              SX6    X2+B1
34.              SA6    A2
35.              EQ     RD$CH
36.    RD$LN     BSSZ   1
37.    *
38.    *  THIS RCUTINE READS A CODED LINE INTO IBUF$ BLANK FILLED
39.    *  IT ALSC RESETS IWRD$ AND IBYTE$ TO ZERO
40.    *
41.    *  RESET IWRD$ , IBYTE$ TO ZERO . BLANKFILL INPUT BUFFER
42.    *
43.              SA1    BLK$
44.              SX6    B0
45.              SA6    IWRD$
46.              SA6    IBYTE$
47.              BX6    X1
48.              SA0    IBUF$
49.              SB7    8
50.              SB6    B1
51.    RD$3      GT     B6,B7,RD$4
52.              SA6    A0
53.              SA0    A0+B1
54.              SB6    B6+B1
55.              JP     RD$3
56.    *
57.    *  SAVE REGISTERS B2,B3,B4,B5
58.    *
```

```
RD$4      SX6     B2
          SA6     B$2345
          SX6     B3
          SA6     A6+B1
          SX6     B4
          SA6     A6+B1
          SX6     B5
          SA6     A6+B1
*
*  READ A LINE INTO INPUT BUFFER
*
          READH   INPUT,IBUF$,IBUF$L
*
*  RESTORE REGISTERS B2,B3,B4,B5
*
          SA2     B$2345
          SB2     X2
          SA2     A2+B1
          SB3     X2
          SA2     A2+B1
          SB4     X2
          SA2     A2+B1
          SB5     X2
*
*  TEST WETHER EOF OCCURED
*
          NZ      X1,RD$ERR
          EQ      RD$LN
*
*  REPORT EOF CONDITTION AND ABORT
*
RD$ERR    SX6     B5
          SA6     B$2345
          WRITEH  OUTPUT,RD$MSG1,RD$LEN1
          WRITER  OUTPUT
          SA1     B$2345
          RJ      TRACE$B
          ABORT
          ENDRUN
RD$MSG1   DATA    H* EOR/EOF/EOI ENCOUNTERED ON INPUT FILE *
RD$LEN1   EQU     *-RD$MSG1
B$2345    BSSZ    4
IBUF$     BSS     8
IBUF$L    EQU     8
IWRD$     BSSZ    1
IBYTE$    BSSZ    1
```

```
 1.     FOR$1     BSSZ   1              LIBRARY OF FOR$1
 2.     *
 3.     *  ROUTINE TO ALLOCATE THREE WORDS IN FOR-LOOP STACK
 4.     *
 5.               SA1    F$TOP
 6.               SX2    3
 7.               IX3    X1+X2
 8.               SX4    F$MAX
 9.               IX5    X4-X3
10.               MI     X5,F$ERR
11.               BX6    X3
12.               SA6    A1
13.               EQ     FOR$1
14.     *
15.     *  REPORT ERROR IF STACK OVERFLOW OCCURED
16.     *
17.     F$ERR     SX6    B5
18.               SA6    B$5A
19.               WRITEH OUTPUT,MSG$4,LEN$4
20.               WRITER OUTPUT
21.               SA1    B$5A
22.               RJ     TRACE$B
23.               ABORT
24.               ENDRUN
25.     B$5A      BSSZ   1
26.     MSG$4     DATA   H* FOR-LOOP STACK OVERFLOW *
27.     LEN$4     EQU    *-MSG$4
28.     FOR$2     BSSZ   1              LIBRARY OF FOR$2
29.               EQ     B7,B0,IL$1
30.               SX1    B1
31.     IL$1      BX0    X1
32.               ZR     X0,STEP$ER
33.               SA1    F$TOP
34.               SA2    X1+F$STACK-2
35.               SA1    X1+F$STACK-3
36.               MI     X0,IL$2
37.               IX1    X2-X1
38.               MI     X1,STEP$ER
39.               JP     IL$3
40.     IL$2      IX1    X1-X2
41.               ZR     X1,IL$3
42.               PL     X1,IL$3
43.               JP     STEP$ER
44.     IL$3      BX6    X0
45.               SA1    F$TOP
46.               SA6    X1+F$STACK-1
47.               EQ     FOR$2
48.     STEP$ER   SX6    B5
49.               SA6    B$5B
50.               WRITEH OUTPUT,MSG$5,LEN$5
51.               WRITER OUTPUT
52.               SA1    B$5B
53.               RJ     TRACE$B
54.               ABORT
55.               ENDRUN
56.     B$5B      BSSZ   1
57.     MSG$5     DATA   H* ERROR IN STEP OF FOR LOOP *
58.     LEN$5     EQU    *-MSG$5
```

```
59.     FOR$3     BSSZ    1              LIBRARY OF FOR$3
60.               SA1     F$TOP
61.               SA2     X1+F$STACK-1
62.               SA3     X1+F$STACK-3
63.               SA4     X1+F$STACK-2
64.               MI      X2,IL$4        IF STEP CNT < 0
65.               IX3     X3-X4          HERE STEP CNT IS > 0
66.               ZR      X3,IL$5
67.               MI      X3,IL$5
68.               SB7     B1             LOOP ENDS
69.               EQ      FOR$3
70.     IL$4      IX3     X4-X3          HERE STEP CNT IS < 0
71.               ZR      X3,IL$5
72.               MI      X3,IL$5
73.               SB7     B1             LOOP ENDS
74.               EQ      FOR$3
75.     IL$5      SB7     B0             LOOP CONTINUES
76.               EQ      FOR$3
77.     FOR$4     BSSZ    1              LIBRARY OF FOR$4
78.               SA1     F$TOP
79.               SA2     X1+F$STACK-1
80.               SA3     X1+F$STACK-3
81.               IX6     X2+X3          INCREMENT LOOP COUNTER
82.               SA6     A3
83.               EQ      FOR$4
```

```
1.      *
2.      *  THIS IS A ROUTINE TO EXPAND THE RUN-TIME DATA STACK
3.      *  AND ALLOCATE PLACE FOR THE DATA REQUIREMENTS OF A NEWLY
4.      *  CALLED ROUTINE
5.      *
6.      R$ENTRY  BSSZ   1              RETURN ADDRESS
7.               SA2  N$TOP            INCREMENT AND TEST N$TOP
8.      *
9.      *  B7  CONTAINS THE CALL TYPE
10.     *  X7  CONTAINS THE NUMBER OF WORDS THE NEW ROUTINE REQUIRES
11.     *      ITS DATA AREA
12.     *  X1  CONTAINS THE NAME OF THE ROUTINE
13.     *
14.              SX3  X2+B1
15.              SX4  N$MAX
16.              IX4  X4-X3
17.              MI   X4,R$E$ERR
18.              BX6  X1               SAVE ROUTINE NAME
19.              SA6  X2+N$STACK
20.              BX6  X3               RESTORE N$TOP
21.              SA6  A2
22.              SX6  B3               CREATE DYNAMIC LINK
23.              SA6  B4+B1
24.              EQ   B7,B1,D$TYPE     CREATE STATIC LINK ACCORDING TO CALL TYPE
25.              NE   B7,B0,E$TYPE
26.              SA1  B3
27.              SX6  X1
28.              JP   CRE$SL
29.     D$TYPE   SX6  B3
30.              JP   CRE$SL
31.     E$TYPE   SA1  B3
32.     R$E$1    SB7  B7-B1
33.              LT   B7,R$E$2
34.              SA1  X1
35.              JP   R$E$1
36.     R$E$2    BX6  X1
37.     CRE$SL   SA6  B4
38.              SX1  B4+4             ALLOCATE L=ARGUMENTS+LOCAL VARS WORDS IN STACK
39.              IX1  X1+X7
40.              SB3  B4               SET B3 TO POINT START OF NEW DATA SEGMENT
41.              SB4  X1               SET B4 TO POINT NEXT AVAILABLE LOCATION IN STACK
42.              SB6  LEN$
43.              GE   B4,B6,R$E$ER1    REPORT STACK OVERFLOW
44.              EQ   R$ENTRY          RETURN
45.     R$E$ERR  SX6  B5
46.              SA6  B$5C
47.              WRITEH OUTPUT,MSG$3,LEN$3     REPORT N$STACK OVERFLOW
48.              WRITER OUTPUT
49.              SA1  B$5C
50.              RJ   TRACE$B
51.              ABORT
52.              ENDRUN
53.     B$5C     BSSZ   1
54.     MSG$3    DATA   H* PROCEDURE-NAME STACK OVERFLOW *
55.     LEN$3    EQU    *-MSG$3
56.     R$E$ER1  SX6    B5
57.              SA6    B$5C
58.              WRITEH OUTPUT,MSG$6,LEN$6     REPORT STACK$  OVERFLOW
```

```
59.                WRITER OUTPUT
60.                SA1    B$5C
61.                RJ     TRACE$B
62.                ABORT
63.                ENDRUN
64.     MSG$6      DATA   H* RUN-TIME DATA STACK OVERFLOW *
65.     LEN$6      EQU    *-MSG$6
66.     *
67.     *  THIS IS A ROUTINE TO SHRINK THE RUN-TIME DATA STACK
68.     *  UPON ROUTINE EXIT
69.     *
70.     R$EXIT     BSSZ 1
71.                SA1  N$TOP          EXTRACT ROUTINE NAME
72.                SX2  B1
73.                IX6  X1-X2
74.                SA6  A1
75.                SB4  B3             RESTORE B4
76.                SA1  B3+B1
77.                SB3  X1             RESTORE B3 (DL)
78.                EQ   R$EXIT
```

```
 1.    CASE$ER   BSSZ   1            REPROT ERROR IF CASE SELECTOR OUT OF RANGE
 2.              SX6    B5
 3.              SA6    B$5D
 4.              WRITEH OUTPUT,MSG$1,LEN$1
 5.              WRITER OUTPUT
 6.              SA1    B$5D
 7.              RJ     TRACE$B
 8.              ABORT
 9.              ENDRUN
10.    B$5D      BSSZ   1
11.    MSG$1     DATA   H* CASE EXPR OUT OF RANGE *
12.    LEN$1     EQU    *-MSG$1
```

```
 1.  SAV$REG  BSSZ   1              AND EXITING FUNCTIONS ( SAVES X1 - X5 )
 2.           SB6    R$STACK
 3.           BX6    X1
 4.           SA1    R$TOP
 5.           SX0    5
 6.           IX0    X0+X1
 7.           SX7    R$MAX
 8.           IX7    X7-X0           IS THERE ENOUGH SPACE IN R$STACK ?
 9.           MI     X7,SAV$ERR      IF X7 < 0 THEN NOT ENOUGH SPACE
10.           SA6    X1+B6           SAVE X1 AT R$STACK+R$TOP
11.           BX6    X2              SAVE X2
12.           SA6    A6+B1
13.           BX6    X3              SAVE X3
14.           SA6    A6+B1
15.           BX6    X4              SAVE X4
16.           SA6    A6+B1
17.           BX6    X5              SAVE X5
18.           SA6    A6+B1
19.           BX6    X0              SAVE R$TOP NEW VALUE
20.           SA6    A1
21.           EQ     SAV$REG
22.  SAV$ERR  SX6    B5
23.           SA6    B$5E
24.           WRITEH OUTPUT,MSG$2,LEN$2
25.           WRITER OUTPUT
26.           SA1    B$5E
27.           RJ     TRACE$B
28.           ABORT
29.           ENDRUN
30.  B$5E     BSSZ   1
31.  MSG$2    DATA   H* REGISTER STACK OVERFLOW *
32.  LEN$2    EQU    *-MSG$2
33.  RES$REG  BSSZ   1              RESTORE REGISTERS UPON RETURNING FROM A
34.           SA1    R$TOP           FUNCTION ( RESTORES X1 - X5 )
35.           SB6    R$STACK
36.           SX0    -5
37.           IX0    X0+X1
38.           SA1    X0+B6           RESTORE X1
39.           SA2    A1+B1           RESTORE X2
40.           SA3    A2+B1           RESTORE X3
41.           SA4    A3+B1           RESTORE X4
42.           SA5    A4+B1           RESTORE X5
43.           BX6    X0              RESTORE R$TOP NEW VALUE
44.           SA6    R$TOP
45.           EQ     RES$REG
```

```
1.    TRACE$B  BSSZ      1
2.    *
3.    *  THIS ROUTINE TARCES BACK THE RECURSIVE PROCEDURES
4.    *  FROM PROCEDURE-NAME STACK
5.    *
6.             SA2     N$TOP
7.             SX3     1
8.             IX2     X2-X3
9.             BX6     X2
10.            SA6     A2
11.            MI      X2,TR$1
12.            SB6     X2
13.            SA2     N$STACK+B6
14.            BX6     X2
15.            SA6     MSG$2B
16.            RJ     =XCDD
17.            SA6     MSG$2D
18.            WRITEH  OUTPUT,MSG$2A,LN$2
19.            WRITER  OUTPUT
20.            JP      TR$2
21.   TR$1     RJ     =XCDD
22.            SA6     MSG$1C
23.            WRITEH  OUTPUT,MSG$1A,LN$1
24.            WRITER  OUTPUT
25.            JP      TRACE$B
26.   TR$2     SA1     N$TOP
27.            MI      X1,TRACE$B
28.            SB6     X1
29.            SA2     N$STACK+B6
30.            BX6     X2
31.            SA6     MSG$3B
32.            SA6     MSG$4B
33.            SX7     1
34.            IX6     X1-X7
35.            MI      X6,TR$3
36.            SB6     B6-B1
37.            SA2     N$STACK+B6
38.            BX7     X2
39.            SA7     MSG$3D
40.            WRITEH  OUTPUT,MSG$3A,LN$3
41.            WRITER  OUTPUT
42.            SA1     N$TOP
43.            SX2     1
44.            IX6     X1-X2
45.            SA6     A1
46.            JP      TR$2
47.   TR$3     WRITEH  OUTPUT,MSG$4A,LN$4
48.            WRITER  OUTPUT
49.            JP      TRACE$B
50.   MSG$1A   DATA    H* ERROR OCCURED AT MAIN PROGRAM*
51.   MSG$1B   DATA    H* NEAR LINE*
52.   MSG$1C   BSSZ    1
53.   LN$1     EQU     *-MSG$1A
54.   MSG$2A   DATA    H* ERROR OCCURED AT PROCEDURE    *
55.   MSG$2B   BSSZ    1
56.   MSG$2C   DATA    H* NEAR LINE*
57.   MSG$2D   BSSZ    1
58.   LN$2     EQU     *-MSG$2A
```

```
59.     MSG$3A    DATA    H*    ... PROCEDURE     *
60.     MSG$3B    BSSZ    1
61.     MSG$3C    DATA    H* WAS CALLED FROM PROCEDURE     *
62.     MSG$3D    BSSZ    1
63.     LN$3      EQU     *-MSG$3A
64.     MSG$4A    DATA    H*    ... PROCEDURE     *
65.     MSG$4B    BSSZ    1
66.     MSG$4C    DATA    H* WAS CALLED FROM MAIN PROGRAM *
67.     LN$4      EQU     *-MSG$4A
68.     *
69.     *  THIS ROUTINE REPORTS AN ILLEGAL SHIFT COUNT
70.     *
71.     SHF$ERR   SX6     B5
72.               SA6     B$5F
73.               WRITEH  OUTPUT,MSG$SHF,LEN$SHF
74.               WRITER  OUTPUT
75.               SA1     B$5F
76.               RJ      TRACE$B
77.               ENDRUN
78.     B$5F      BSSZ    1
79.     MSG$SHF   DATA    H* SHIFT COUNT <0 OR >60 *
80.     LEN$SHF   EQU     *-MSG$SHF
81.     *
82.     *  THIS ROUTINE REPORTS THE 'HALT' LINE WHERE COMMAND HALT WAS EXECUTED
83.     *
84.     HALT$     SX6     B5
85.               SA6     B$5Z
86.               WRITEH  OUTPUT,MSG$HLT,LEN$HLT
87.               WRITER  OUTPUT
88.               SA1     B$5Z
89.               RJ      TRACE$B
90.               ENDRUN
91.     B$5Z      BSSZ    1
92.     MSG$HLT   DATA    H* PROGRAM HALTED VIA A HALT COMMAND *
93.     LEN$HLT   EQU     *-MSG$HLT
```

APPENDIX D. TOWERS OF HANOI PROBLEM

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
 1      //TOWERS
 2      LET TOWERS(N,FROMP,TOP,AUXP) BE
 3        [ TEST N=1
 4              THEN [ WRITESTR(" MOVE DISK":10) ;
 5                     WRITESTR(" 1 FROM PE":10) ;
 6                     WRITESTR("G "         :2 ) ;
 7                     WRITESTR(FROMP        :1 ) ;
 8                     WRITESTR(" TO PEG "   :8 ) ;
 9                     WRITESTR(TOP          :1 ) ;
10                     WRITELN                    ]
11              OR     [ TOWERS(N-1,FROMP,AUXP,TOP) ;
12                     WRITESTR(" MOVE DISK":10) ;
13                     WRITESTR(" "          :1 ) ;
14                     WRITEN(N) ;
15                     WRITESTR(" FROM PEG ":10) ;
16                     WRITESTR(FROMP        :1 ) ;
17                     WRITESTR(" TO PEG "   :8 ) ;
18                     WRITESTR(TOP          :1 ) ;
19                     WRITELN ;
20                     TOWERS(N-1,AUXP,TOP,FROMP)  ] ] ; // END TOWERS
21
22      WRITESTR("1":1) ; WRITELN ;
23      TOWERS(5,"A","C","B") ;
24      FINISH
```

```
PROGRAM CONTAINS 24 LINES AND 1 PROCEDURES
  PROGRAM STATISTICS :
   ** INSTRUCTIONS      :      85 WORDS
   ** RUN TIME  STACK   :    3072 WORDS
   ** CONSTANTS TABLE   :      17 WORDS
   ** FOR LOOP  STACK   :      48 WORDS
   ** REGISTER  STACK   :     128 WORDS
   ** PROC-NAME STACK   :     128 WORDS
   ** LIBRARY   SIZE    :     367 WORDS
  TOTAL PROGRAM SIZE    :    3845 WORDS
END BCPL *** LEVEL 1 *** 0 ERRORS FOUND DURING COMPILATION . TIME USED :    0.615 SECONDS
```

```
MOVE DISK 1 FROM PEG A TO PEG C
MOVE DISK 2 FROM PEG A TO PEG B
MOVE DISK 1 FROM PEG C TO PEG B
MOVE DISK 3 FROM PEG A TO PEG C
MOVE DISK 1 FROM PEG B TO PEG A
MOVE DISK 2 FROM PEG B TO PEG C
MOVE DISK 1 FROM PEG A TO PEG C
MOVE DISK 4 FROM PEG A TO PEG B
MOVE DISK 1 FROM PEG C TO PEG B
MOVE DISK 2 FROM PEG C TO PEG A
MOVE DISK 1 FROM PEG B TO PEG A
MOVE DISK 3 FROM PEG C TO PEG B
MOVE DISK 1 FROM PEG A TO PEG C
MOVE DISK 2 FROM PEG A TO PEG B
MOVE DISK 1 FROM PEG C TO PEG B
MOVE DISK 5 FROM PEG A TO PEG C
MOVE DISK 1 FROM PEG B TO PEG A
MOVE DISK 2 FROM PEG B TO PEG C
MOVE DISK 1 FROM PEG A TO PEG C
MOVE DISK 3 FROM PEG B TO PEG A
MOVE DISK 1 FROM PEG C TO PEG B
MOVE DISK 2 FROM PEG C TO PEG A
MOVE DISK 1 FROM PEG B TO PEG A
MOVE DISK 4 FROM PEG B TO PEG C
MOVE DISK 1 FROM PEG A TO PEG C
MOVE DISK 2 FROM PEG A TO PEG B
MOVE DISK 1 FROM PEG C TO PEG B
MOVE DISK 3 FROM PEG A TO PEG C
MOVE DISK 1 FROM PEG B TO PEG A
MOVE DISK 2 FROM PEG B TO PEG C
MOVE DISK 1 FROM PEG A TO PEG C
```

APPENDIX E. TREE DIFFERENTIATION PROGRAM

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
 1      //DIFF1
 2      #F 1024
 3      #R 1000
 4      #N 354
 5      #S 33000
 6
 7                              // GLOBAL VARIABLES
 8
 9      STATIC [ CHAR : 0  ;  //  TO STORE NEXT CHARACTER READ FROM INPUT
10               TY   : 0  ;  //  TYPE OF NODE READ IN
11               VAL  : 0  ;  //  INFO OF NODE READ IN
12               RL   : 0  ;  //  RIGHT LINK OF NODE READ IN
13               LL   : 0  ;  //  LEFT LINK OF NODE READ IN
14               RT   : 0  ;  //  RIGHT TAG OF NODE READ IN
15               G    : 0  ;  //  SAME AS V
16               A    : 0  ;  //  ADDRESS OF A CERTAIN NODE
17               I    : 0  ;  //  INDEX VARIABLE
18               J    : 0  ;  //  INDEX VARIABLE
19               Q    : 0  ;  //  ADDRESS OF SOME NODE
20               Q1   : 0  ;  //  ADDRESS OF SOME NODE
21               P    : 0  ;  //  ADDRESS OF SOME NODE
22               P1   : 0  ;  //  ADDRESS OF SOME NODE
23               P2   : 0  ;  //  ADDRESS OF SOME NODE
24               Y    : 0  ;  //  ADDRESS OF FIRST TREE'S HEADER NODE
25               DY   : 0  ;  //  ADDRESS OF SECOND TREE'S HEADER NODE
26             MASKTY : $7700000000000000000 ; // TYPE FIELD
27             MASKRT : $0077777700000000000 ; // RTAG FIELD
28             MASKLL : $0000000077777000000 ; // LLINK FIELD
29             MASKRL : $0000000000000777777   // RLINK FIELD
30                        ] ;
31
32      LET  V1 = VEC 499 ;     // VECTOR TO KEEP TREE
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
33      #P
34            // ROUTINES TO RETURN DIFFERENT FIELDS OF A NODE
35
36      LET INFO(H) = VALOF
37        [ INFO := H!0 ]  ;  //  END FUNCTION INFO(H)
38
39      LET RLINK(H) = VALOF
40        [ RLINK := H!1 LAND MASKRL ] ;  // END FUNCTION RLINK(H)
41
42      LET LLINK(H) = VALOF
43        [ LLINK := ( H!1 LAND MASKLL ) RSHIFT 18 ]  ;  //  END FUNCTION LLINK(H)
44
45      LET RTAG(H) = VALOF
46        [ RTAG := ( H!1 LAND MASKRT ) RSHIFT 36 ]  ;  // END FUNCTION RTAG(H)
47
48      LET TYPE(H) = VALOF
49        [ TYPE := ( H!1 LAND MASKTY ) RSHIFT 54 ]  ;  // END FUNCTION TYPE(H)
50
51              // ROUTINES TO SET DIFFERENT FIELDS OF A NODE
52
53      LET SINFO(H,P) BE
54        [ H!0 := P ]  ;                                          // END SINFO
55
56      LET STYPE(H,P) BE
57        [ H!1 := (H!1 LAND (NOT MASKTY)) LOR (P LSHIFT 54) ]  ;  // END STYPE
58
59      LET SRTAG(H,P) BE
60        [ H!1 := (H!1 LAND (NOT MASKRT)) LOR (P LSHIFT 36) ]  ;  // END SRTAG
61
62      LET SLLINK(H,P) BE
63        [ H!1 := (H!1 LAND (NOT MASKLL)) LOR (P LSHIFT 18) ]  ;  // END SLLINK
64
65      LET SRLINK(H,P) BE
66        [ H!1 := (H!1 LAND (NOT MASKRL)) LOR P ]  ;              // END SLLINK
67
68      LET CREATE() = VALOF
69        [ A:=A+2 ;
70          IF A > 499 DO
71            [ WRITESTR(" VECTOR OV":10 , "ERFLOW    ":10) ;
72              WRITELN ;
73              HALT     ] ;
74          CREATE:= V1+(A-1) ]  ;  // END CREATE
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
      #P
            // THIS ROUTINE LINKS RIGHT THREADS OF NODES AFTER
            // THEY HAVE BEEN COPIED BY ROUTINE COPY(H)

      LET THREAD(H) BE
        [ STATIC [ W:0 ] ;
          TEST LLINK(H)<>0
            THEN [ W:=LLINK(H) ;
                   TEST RTAG(W)=0
                     THEN SRLINK(W,H)
                     OR   [ W:=RLINK(W) ;
                            SRLINK(W,H)  ] ]
            OR   [ WRITESTR(" NO RIGHT ":10 , "THREAD CAN":10 ,
                            " POINT TO ":10 , "THAT NODE ":10 ) ;
                   WRITELN  ]  ]  ;       //  END THREAD

           // THIS ROUTINE COPIES NODES STARTING AT ADDRESS H
           // TO A NEW PLACE AND RETURNS THEIR DESTINATION ADDRESS

      LET COPY(H) = VALOF
        [ STATIC [ W:0 ; W1:0 ; I:0 ] ;
          W:=CREATE() ;
          I:=INFO(H) ;
          SINFO(W,I) ;
          I:=TYPE(H) ;
          STYPE(W,I) ;
          I:=RTAG(H) ;
          SRTAG(W,I) ;
          TEST RTAG(H)=1
            THEN [ W1:=COPY(RLINK(H)) ;
                   SRLINK(W,W1) ;
                   TEST LLINK(H)<>0
                     THEN [ W1:=COPY(LLINK(H)) ;
                            SLLINK(W,W1) ;
                            THREAD(W)  ]
                     OR   SLLINK(W,0)  ]
            OR   TEST LLINK(H)<>0
                   THEN [ W1:=COPY(LLINK(H)) ;
                          SLLINK(W,W1) ;
                          THREAD(W)  ]
                   OR   SLLINK(W,0) ;
          COPY:=W  ]  ;  //  END COPY
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
118     #P
119
120         //  THESE ROUTINES CREATE NEW TREES WITH INFO AND SUBTREES
121         //  THE ARGUMENTS SENT
122
123     LET  TREE0(A) = VALOF
124       [ STATIC [ W:0 ] ;
125         W:=CREATE() ;
126         SINFO(W,A) ;
127         SLLINK(W,0) ;
128         RESULTIS W          ] ;    // END FUNCTION TREE0
129
130     LET TREE1(A,U) = VALOF
131       [ STATIC [ W:0 ] ;
132         W:=CREATE() ;
133         SINFO(W,A) ;
134         SLLINK(W,U) ;
135         SRLINK(U,W) ;
136         SRTAG(U,0) ;
137         RESULTIS W          ] ;    // END FUNCTION TREE1
138
139     LET TREE2(A,U,V) = VALOF
140       [ STATIC [ W:0 ] ;
141         W:=CREATE() ;
142         SINFO(W,A) ;
143         SLLINK(W,U) ;
144         SRLINK(U,V) ;
145         SRTAG(U,1) ;
146         SRLINK(V,W) ;
147         SRTAG(V,0) ;
148         RESULTIS W          ]  ;    // END FUNCTION TREE2
149
150     LET MULT(U,V) = VALOF
151       [ STATIC [ P:0 ] ;
152         IF (INFO(U)=1) LAND (TYPE(U)=0) DO RESULTIS V ;
153         IF (INFO(V)=1) LAND (TYPE(V)=0) DO RESULTIS U ;
154         P:= TREE2("          *",U,V) ;
155         STYPE(P,6) ;
156         RESULTIS P                                          ] ; // END FUNCTION MU
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
157     #P
158
159         // THE FOLLOWING PROCEDURE MAKES THE ACTUAL DIFFERENTIATION
160
161     LET DIFF(T) BE
162       [ STATIC [ C :0 ; C2:0 ; M:0 ; T2 :0 ;
163                  T0:0 ; T1:0 ; R :0 ; I:0 ; T2A:0  ] ;
164         IF T=0 DO
165           [ Q:=TREE0(0) ;
166             STYPE(Q,0)    ] ;
167         IF T=1 DO
168           TEST INFO(P)="          X"
169               THEN [ Q:=TREE0(1) ;
170                      STYPE(Q,0)    ]
171               OR   [ Q:=TREE0(0) ;
172                      STYPE(Q,0)    ] ;
173         IF T=2 DO
174           IF INFO(Q)<>0 DO
175             [ C:=COPY(P1) ;
176               Q:=TREE2("          /",Q,C) ;
177               STYPE(Q,7)                    ] ;
178         IF T=3 DO
179           IF INFO(Q)<>0 DO
180             [ Q:=TREE1("          N",Q) ;
181               STYPE(Q,3)                  ] ;
182         IF T=4 DO
183           TEST INFO(Q1)=0
184               THEN [ ]
185               OR   TEST INFO(Q)=0
186                         THEN Q:=Q1
187                         OR   [ Q:=TREE2("          +",Q1,Q) ;
188                                STYPE(Q,4)                      ]  ;
189         IF T=5 DO
190           TEST INFO(Q)=0
191               THEN Q:=Q1
192               OR   TEST INFO(Q1)=0
193                         THEN [ Q:=TREE1("          N",Q) ;
194                                STYPE(Q,3)                    ]
195                         OR   [ Q:=TREE2("          -",Q1,Q) ;
196                                STYPE(Q,5)                    ] ;
197         IF T=6 DO
198           [ IF INFO(Q1)<>0 DO
199               [ C:=COPY(P2) ;
200                 Q1:=MULT(Q1,C) ] ;
201             IF INFO(Q)<>0 DO
202               [ C:=COPY(P1) ;
203                 Q:=MULT(C,Q)   ] ;
204             DIFF(4)                ] ;
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
205     #P
206         IF T=7 DO
207           [ IF INFO(Q1)<>0 DO
208                [ C:=COPY(P2) ;
209                  Q1:=TREE2("          /",Q1,C) ;
210                  STYPE(Q1,7)                    ] ;
211             IF INFO(Q)<>0 DO
212               [ C:= COPY(P1) ;
213                 C2:=COPY(P2) ;
214                 M:= MULT(C,Q) ;
215                 T0:=TREE0(2) ;
216                 STYPE(T0,0) ;
217                 T2:=TREE2("          ^",C2,T0) ;
218                 STYPE(T2,8) ;
219                 Q:=TREE2("          /",M,T2) ;
220                 STYPE(Q,7)                     ] ;
221             DIFF(5)                                ] ;
222         IF T=8 DO
223           [ IF INFO(Q1)<>0 DO
224                [ R:=COPY(P1) ;
225                  IF (TYPE(P2)=0) LAND (INFO(P2)<>2) DO
226                    [ I:=INFO(P2)-1 ;
227                      T0:=TREE0(I) ;
228                      STYPE(T0,0) ;
229                      R:=TREE2("          ^",R,T0) ;
230                      STYPE(R,8)                  ] ;
231                  IF TYPE(P2)<>0 DO
232                    [ C2:=COPY(P2) ;
233                      T0:=TREE0(1) ;
234                      STYPE(T0,0) ;
235                      T2:=TREE2("          -",C2,T0) ;
236                      STYPE(T2,5) ;
237                      R:=TREE2("          ^",R,T2) ;
238                      STYPE(R,8)                      ] ;
239                  C2:=COPY(P2) ;
240                  M:=MULT(C2,R) ;
241                  Q1:=MULT(Q1,M) ] ;
242             IF INFO(Q)<>0 DO
243               [ C:=COPY(P1) ;
244                 T1:=TREE1("          L",C) ;
245                 STYPE(T1,2) ;
246                 M:=MULT(T1,Q) ;
247                 C:=COPY(P1) ;
248                 C2:=COPY(P2) ;
249                 T2:=TREE2("          ^",C,C2) ;
250                 STYPE(T2,8) ;
251                 Q:=TREE2("          *",M,T2) ;
252                 STYPE(Q,6)                     ] ;
253             DIFF(4)  ]  ;                          ] ; // END PROCEDURE DIFF(T)
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
      #P

         // THIS ROUTINE FINDS THE POSTORDER SUCCESSOR OF NODE
         // H AND RETURNS THE ADRESS OF IT

      LET  SUCC(H,V) = VALOF
        [ TEST RTAG(H)=0
            THEN SUCC := RLINK(H)
            OR   [ IF (RLINK(H)<>0) LAND (H<>V) DO H := RLINK(H) ;
                   WHILE LLINK(H)<>0 DO H := LLINK(H) ;
                   SUCC := H   ]  ]  ;  // END FUNCTION SUCC

         // THIS ROUTINE STARTS DIFFERENTIATION AND
         // MANIPULATES IT BY CALLING DIFF(T)

      LET DOES() BE
        [ STATIC [ T:0 ] ;
          P:=SUCC(Y) ;
        ONE :
          P1:=LLINK(P) ;
          IF P1<>C DO Q1:=RLINK(P1) ;
          T:=TYPE(P) ;
          DIFF(T) ;
          IF T>=4 DO SRLINK(P1,P2) ;
          P2:=P ;
          P:=SUCC(P) ;
          IF RTAG(P2)=1 DO SRLINK(P2,Q) ;
          IF P<>Y DO GOTO ONE ;
          SLLINK(DY,Q) ;
          SRLINK(C,DY) ;
          SRTAG(Q,0)          ]  ;  // END PROCEDURE DOES

        // THIS RCUTINE OUTPUTS THE INFO FIELD OF A NODE TO FILE OUTPUT

      LET OUTPUT(H) BE
        [ STATIC [ IN:0 ; I:0 ; J:0 ] ;
          TY := TYPE(H) ;
          TEST TY=0
            THEN WRITEN(INFO(H))
            OR   [ IN:=INFO(H) ;
                   FOR I:=1 TO 10 DO
                     [ J:=IN LAND MASKTY ;
                       IF J<>" " DO WRITESTR(J:1) ;
                       IN:=IN LSHIFT 6          ]] ;
          WRITESTR(",":1) ]  ;                    //  END PROCEDURE OUTPUT
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
299     #P
300                         //  MAIN PROGRAM
301
302                         //  BUILD TREE
303     WRITESTR("1":1) ;                   WRITELN ; WRITELN ;
304     WRITESTR(" INPUT DAT":10,"A":1) ; WRITELN ; WRITELN ;
305     A := 0 ;
306     READLN ;
307     READCH(CHAR) ; WRITESTR(" ":1) ; WRITESTR(CHAR:1) ;
308
309     WHILE CHAR <> "." DO
310       [ TY := (CHAR-"0") RSHIFT 54 ;
311         TEST TY=0
312           THEN [ VAL := 0 ;
313                  FOR I:=1 TO 10 DO
314                    [ READCH(CHAR) ;
315                      TEST CHAR <> "$"
316                        THEN [ WRITESTR(CHAR:1) ;
317                               VAL := VAL*10 + (CHAR-"0") RSHIFT 54 ]
318                        OR   WRITESTR(" ":1) ]]
319           OR    [ VAL := 0 ;
320                  FOR I:=1 TO 10 DO
321                    [ READCH(CHAR) ;
322                      TEST CHAR="$"
323                         THEN [ WRITESTR("  ":1) ; CHAR:=" " ]
324                         OR   WRITESTR(CHAR:1) ;
325                      J := (I-1)*6 ;
326                      CHAR := CHAR RSHIFT J ;
327                      VAL := VAL LOR CHAR    ] ] ;
328
329         READCH(CHAR) ; WRITESTR(CHAR:1) ;
330         RT := (CHAR-"0") RSHIFT 54 ;
331
332         READCH(CHAR) ; WRITESTR(CHAR:1) ;
333         LL := 0 ;
334         TEST CHAR=" "
335           THEN [ READCH(CHAR) ; WRITESTR(CHAR:1) ;
336                  IF CHAR="V" DO LL := V1 ;
337                  IF CHAR<>"V" DO LL := (CHAR-"0") RSHIFT 54 + V1 ]
338           OR    [ LL := (CHAR-"0") RSHIFT 54 ;
339                  READCH(CHAR) ; WRITESTR(CHAR:1) ;
340                  LL := (CHAR-"0") RSHIFT 54 + LL*10 ;
341                  TEST LL=99
342                    THEN LL:=0
343                    OR   LL:=LL+V1 ] ;
344
345         READCH(CHAR) ; WRITESTR(CHAR:1) ;
346         RL := 0 ;
347         TEST CHAR=" "
348           THEN [ READCH(CHAR) ; WRITESTR(CHAR:1) ;
349                  IF CHAR="V" DO RL := V1 ;
350                  IF CHAR<>"V" DO RL := (CHAR-"0") RSHIFT 54 + V1 ]
351           OR    [ RL := (CHAR-"0") RSHIFT 54 ;
352                  READCH(CHAR) ; WRITESTR(CHAR:1) ;
353                  RL := (CHAR-"0") RSHIFT 54 + RL*10 ;
354                  TEST RL=99
355                    THEN RL:=0
356                    OR   RL:=RL +V1 ] ;
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
357     #P
358         //  FILL A NODE OF THE TREE THAT IS BEING BUILT
359
360         V1!A := VAL ;
361         A := A+1 ;
362         V1!A := (TY LSHIFT 54) LOR (RT LSHIFT 36) LOR
363                 (LL LSHIFT 18) LOR RL ;
364         A := A+1 ;
365         READLN ; WRITELN ; WRITESTR(" ":1) ;
366         READCH(CHAR) ; WRITESTR(CHAR:1) ;    ]  ;  // END WHILE
367     WRITELN ;
368
369     A:=A-1 ;    // NEXT AVAILABLE LOCATION  IN VECTOR V1
370
371     Y:=V1 ;     // Y HOLDS THE STARTING ADRESS OF THE ORIGINAL TREE
372
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
373     #P
374       //  FIND POSTORDER OF ORIGINAL EXPRESSION
375
376     WRITESTR("1":1) ;
377     WRITELN ;
378     WRITESTR(" POSTORDER":10,
379              " OF ORIGIN":10,
380              "AL EXPRESS":10,
381              "ION       ":10 ) ;
382     WRITELN ;
383     WRITELN ;
384     WRITESTR("  ":1) ;
385     G := SUCC(V1,V1) ; I:=1 ;
386     WHILE G <> V1 DO
387       [ OUTPUT(G) ; IF I=50 DO [ WRITELN ; WRITESTR(" ":1) ] ; I:=I+1 ;
388         G := SUCC(G,V1) ] ;
389     WRITELN ;
390     WRITELN ;
391     WRITESTR(" DUMP OF M":10,
392              "EMORY OF T":10,
393              "HE ORIGINA":10,
394              "L TREE    ":10 ) ;
395     WRITELN ;
396     WRITELN ;
397     WRITESTR(" ADRESS   ":10,
398              "          ":10,
399              "   CONTENT":10,
400              "S         ":10 ) ;
401     WRITELN ;
402     WRITELN ;
403     FOR I:=0 TO 50 DO
404       [ WRITESTR(" ":1) ; WRITEO(Y+I) ; WRITESTR(" ":1) ;
405         WRITEO(Y!I) ; WRITELN ] ;
406     WRITELN ;
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
407     #P
408       //  START DIFFERENTIATION
409
410     DY:=CREATE() ;
411     SRLINK(DY,DY) ;
412     SRTAG(DY,1) ;
413     SINFO(DY,"          Y") ;
414     STYPE(DY,1) ;
415     DOES() ;
416
417       //  OUTPUT DIFFERENTIATED TREE
418
419     WRITESTR("1":1) ;
420     WRITELN ;
421     WRITESTR(" POSTORDER":10,
422              " OF DIFFER":10,
423              "ENTIATED E":10,
424              "XPRESSION ":10 ) ;
425     WRITELN ;
426     WRITELN ;
427     WRITESTR("  ":1) ;
428     G := SUCC(DY,DY) ; I:=1 ;
429     WHILE G <> DY DO
430       [ OUTPUT(G) ; IF I=50 DO [ WRITELN ; WRITESTR(" ":1) ] ; I:=I+1 ;
431         G := SUCC(G,DY) ] ;
432     WRITELN ; WRITELN ;
433
434     WRITESTR(" DUMP OF M":10,
435              "EMORY OF T":10,
436              "HE DIFFERE":10,
437              "NTIATED TR":10,
438              "EE         ":10 ) ;
439     WRITELN ;
440     WRITELN ;
441     WRITESTR(" ADRESS   ":10,
442              "          ":10,
443              "   CONTENT":10,
444              "S         ":10 ) ;
445     WRITELN ;
446     WRITELN ;
447     FOR I:=0 TO 250 DO
448       [ IF (I=51) LOR (I=101) LOR (I=151) LOR (I=201) DO
449           [ WRITESTR("1":1) ; WRITELN ;
450             WRITESTR(" ADRESS   ":10,
451                      "          ":10,
452                      "   CONTENT":10,
453                      "S         ":10 ) ;
454             WRITELN ;
455             WRITELN ; ] ;
456         WRITESTR(" ":1) ; WRITEO(DY+I) ; WRITESTR(" ":1) ;
457         WRITEO(DY!I)    ; WRITELN                          ] ;
458     WRITELN ;
459
460     FINISH
```

```
PROGRAM CONTAINS 460 LINES AND 21 PROCEDURES
  PROGRAM STATISTICS :
   ** INSTRUCTIONS      :    2531 WORDS
   ** RUN TIME  STACK   :   33000 WORDS
   ** CONSTANTS TABLE   :      90 WORDS
   ** FOR LOOP  STACK   :    1024 WORDS
   ** REGISTER  STACK   :    1000 WORDS
   ** PROC-NAME STACK   :     354 WORDS
   ** LIBRARY   SIZE    :     790 WORDS
  TOTAL PROGRAM SIZE    :   38789 WORDS
END BCPL *** LEVEL 1 *** 0 ERRORS FOUND DURING COMPILATION . TIME USED :  14.684 SECONDS
```

INPUT DATA

```
1          Y1 2 V
5          -0 4 0
6          #1 616
0          3199 8
2          L010 4
4          +012 8
1          X19914
0          109910
7          /018 2
1          A19920
8          ^02216
1          X19924
0          209920
•
```

```
POSTORDER OF ORIGINAL EXPRESSION

3,X,1,+,L,*,A,X,2,^,/,-,

DUMP OF MEMORY OF THE ORIGINAL TREE

ADRESS                    CONTENTS

000000000000000011766B 5555555555555555531B
000000000000000011767B 0100C0010117700011766B
000000000000000011770B 5555555555555555546B
000000000000000011771B 0500C0000117720011766B
000000000000000011772B 5555555555555555547B
000000000000000011773B 0600C0010117740012006B
000000000000000011774B 0000C0000000000000003B
000000000000000011775B 0000C0010000000011776B
000000000000000011776B 5555555555555555514B
000000000000000011777B 0200C0000120000011772B
000000000000000012000B 5555555555555555545B
000000000000000012001B 0400C0000120020011776B
000000000000000012002B 5555555555555555530B
000000000000000012003B 0100C0010000000120048
000000000000000012004B 0000C0000000000000018
000000000000000012005B 0000C0000000000120008
000000000000000012006B 5555555555555555550B
000000000000000012007B 0700C0000120100011770B
000000000000000012010B 5555555555555555501B
000000000000000012011B 0100C0010000000120128
000000000000000012012B 5555555555555555576B
000000000000000012013B 1000C0000120140120068
000000000000000012014B 5555555555555555530B
000000000000000012015B 0100C0010000000120168
000000000000000012016B 0000C0000000000000028
000000000000000012017B 0000C0000000000120128
000000000000000012020B 0000C0000000000000008
000000000000000012021B 0000C0000000000000008
000000000000000012022B 0000C0000000000000008
000000000000000012023B 0000C0000000000000008
000000000000000012024B 0000C0000000000000008
000000000000000012025B 0000C0000000000000008
000000000000000012026B 0000C0000000000000008
000000000000000012027B 0000C0000000000000008
000000000000000012030B 0000C0000000000000008
000000000000000012031B 0000C0000000000000008
000000000000000012032B 0000C0000000000000008
000000000000000012033B 0000C0000000000000008
000000000000000012034B 0000C0000000000000008
000000000000000012035B 0000C0000000000000008
000000000000000012036B 0000C0000000000000008
000000000000000012037B 0000C0000000000000008
000000000000000012040B 0000C0000000000000008
000000000000000012041B 0000C0000000000000008
000000000000000012042B 0000C0000000000000008
000000000000000012043B 0000C0000000000000008
000000000000000012044B 0000C0000000000000008
000000000000000012045B 0000C0000000000000008
000000000000000012046B 0000C0000000000000008
000000000000000012047B 0000C0000000000000008
000000000000000012050B 0000C0000000000000008
```

POSTORDER OF DIFFERENTIATED EXPRESSION

3,1,X,1,+,/,*,A,2,X,*,*,X,2,^,2,^,/,N,-,

DUMP OF MEMORY OF THE DIFFERENTIATED TREE

ADRESS	CONTENTS
000000000C0000120208	55555555555555555318
000000000C0000120218	0100C00101211001202B
000000000C0000120228	0000C0000000000000008
000000000C0000120238	0000C0000000000000008
000000000C0000120248	0000C0000000000000018
000000000C0000120258	0000C0010000000120308
000000000C0000120268	0000C0000000000000008
000000000C0000120278	0000C0000000000000008
000000000C0000120308	55555555555555555458
000000000C0000120318	0400C0000120320120368
000000000C0000120328	55555555555555555308
000000000C0000120338	0100C0010000000120348
000000000C0000120348	0000C0000000000000018
000000000C0000120358	0000C0000000000120308
000000000C0000120368	55555555555555555508
000000000C0000120378	0700C0000120240120448
000000000C0000120408	0000C0000000000000038
000000000C0000120418	0000C0010000000120368
000000000C0000120428	0000C0000000000000008
000000000C0000120438	0000C0000000000000008
000000000C0000120448	55555555555555555478
000000000C0000120458	0600C0010120400121068
000000000C0000120468	0000C0000000000000008
000000000C0000120478	0000C0000000000000008
000000000C0000120508	0000C0000000000000018
000000000C0000120518	0000C0000000000000008
000000000C0000120528	0000C0000000000000008
000000000C0000120538	0000C0000000000000008
000000000C0000120548	55555555555555555308
000000000C0000120558	0100C0000000000120628
000000000C0000120568	0000C0000000000000018
000000000C0000120578	0000C0000000000000008
000000000C0000120608	0000C0000000000000028
000000000C0000120618	0000C0010000000120548
000000000C0000120628	55555555555555555478
000000000C0000120638	0600C0000120600120768
000000000C0000120648	55555555555555555018
000000000C0000120658	0100C0010000000120628
000000000C0000120668	0000C0000000000000008
000000000C0000120678	0000C0000000000000008
000000000C0000120708	55555555555555555768
000000000C0000120718	1000C0010120720121008
000000000C0000120728	55555555555555555308
000000000C0000120738	0100C0010000000120748
000000000C0000120748	0000C0000000000000028
000000000C0000120758	0000C0000000000120708
000000000C0000120768	55555555555555555478
000000000C0000120778	0600C0010120640121028
000000000C0000121008	0000C0000000000000028
000000000C0000121018	0000C0000000000121028
000000000C0000121028	55555555555555555768

ADRESS	CONTENTS
000000000C0000121038	1000C00001207001210 4B
000000000C0000121048	55555555555555555508
000000000C0000121058	0700C00001207601210 6B
000000000C0000121068	55555555555555555168
000000000C0000121078	0300C00001210401211 0B
000000000C0000121108	55555555555555555468
000000000C0000121118	0500C00001204401202 0B
000000000C0000121128	0000C00000C0000000 0B
000000000C0000121138	0000C0000000000000 0B
000000000C0000121148	0000C0000000000000 0B
000000000C0000121158	0000C0000000000000 0B
000000000C0000121168	0000C0000000000000 0B
000000000C0000121178	0000C0000000000000 0B
000000000C0000121208	0000C0000000000000 0B
000000000C0000121218	0000C0000000000000 0B
000000000C0000121228	0000C0000000000000 0B
000000000C0000121238	0000C0000000000000 0B
000000000C0000121248	0000C0000000000000 0B
000000000C0000121258	0000C0000000000000 0B
000000000C0000121268	0000C0000000000000 0B
000000000C0000121278	0000C0000000000000 0B
000000000C0000121308	0000C0000000000000 0B
000000000C0000121318	0000C0000000000000 0B
000000000C0000121328	0000C0000000000000 0B
000000000C0000121338	0000C0000000000000 0B
000000000C0000121348	0000C0000000000000 0B
000000000C0000121358	0000C0000000000000 0B
000000000C0000121368	0000C0000000000000 0B
000000000C0000121378	0000C0000000000000 0B
000000000C0000121408	0000C0000000000000 0B
000000000C0000121418	0000C0000000000000 0B
000000000C0000121428	0000C0000000000000 0B
000000000C0000121438	0000C0000000000000 0B
000000000C0000121448	0000C0000000000000 0B
000000000C0000121458	0000C0000000000000 0B
000000000C0000121468	0000C0000000000000 0B
000000000C0000121478	0000C0000000000000 0B
000000000C0000121508	0000C0000000000000 0B
000000000C0000121518	0000C0000000000000 0B
000000000C0000121528	0000C0000000000000 0B
000000000C0000121538	0000C0000000000000 0B
000000000C0000121548	0000C0000000000000 0B
000000000C0000121558	0000C0000000000000 0B
000000000C0000121568	0000C0000000000000 0B
000000000C0000121578	0000C0000000000000 0B
000000000C0000121608	0000C0000000000000 0B
000000000C0000121618	0000C0000000000000 0B
000000000C0000121628	0000C0000000000000 0B
000000000C0000121638	0000C0000000000000 0B
000000000C0000121648	0000C0000000000000 0B

APPENDIX F. THE EIGHT QUEENS PROGRAM

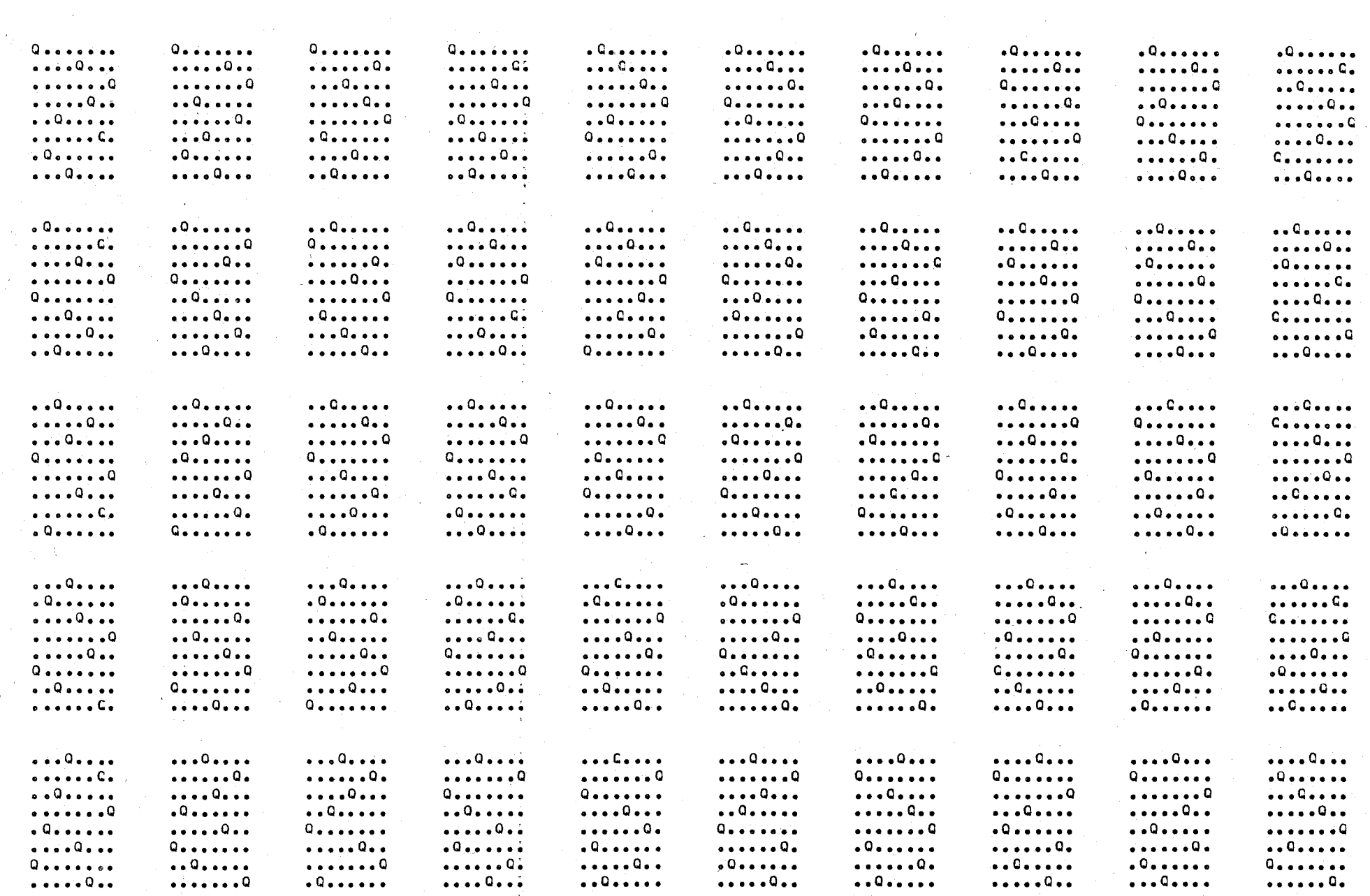
SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

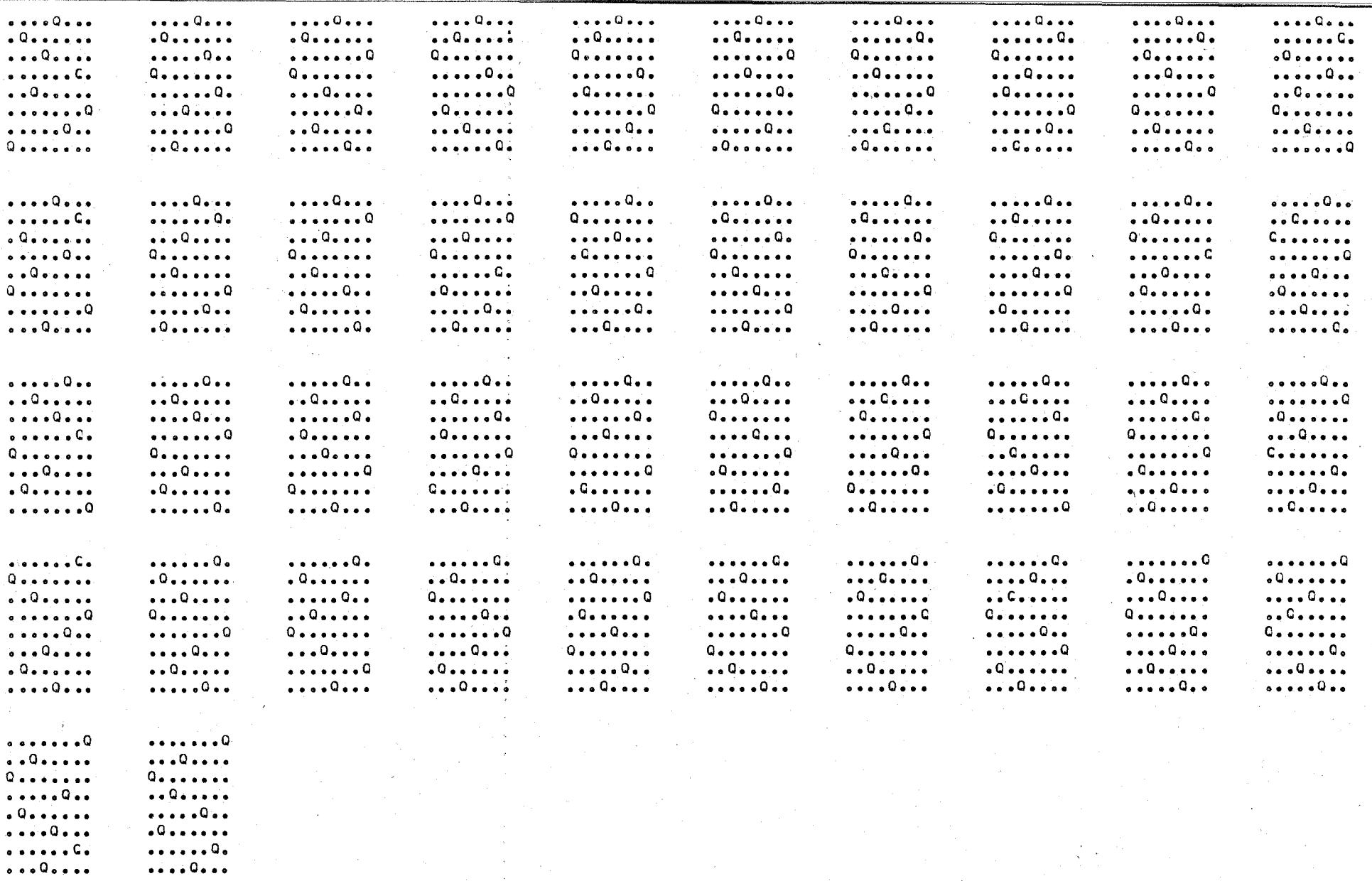
```
 1      //QUEEN
 2      #F 1024
 3
 4      STATIC [ M   : 0 ; //
 5               N   : 0 ; //
 6               ROW : 2 ; //
 7               R   : 0 ; //
 8               I   : 0 ; //
 9               J   : 0 ; //
10               MAT : 0 ; //
11               Q   : 0 ; //
12               COL : 0 ; //
13               CNT : 0   //
14             ] ;
15
16     LET MATRIX = VEC 640 AND QUEEN = TABLE "?",1,0,0,0,0,0,0,0 ;
17
18     LET SETMATRIX(I,J,CH) BE
19           [ STATIC [ OFFSET : 0 ] ;
20             OFFSET := (J-1)*80+I  ;
21             MATRIX!OFFSET := CH    ] ; // END PROCEDURE SETMATRIX
22
23     LET GETMATRIX(I,J) = VALOF
24           [ STATIC [ OFFSET : 0 ] ;
25             OFFSET := (J-1)*80+I  ;
26             RESULTIS MATRIX!OFFSET ] ; // END FUNCTION GETMATRIX
27
28     LET OUTPUT(N) BE
29           [ STATIC [ I:0 ; J:0 ; M:0 ; CH:" " ] ;
30             FOR J:=1 TO 8 DO
31               [ FOR M:=1 TO N/8 DO
32                   [ WRITESTR("     ":5) ;
33                     FOR I:=1 TO 8 DO
34                       [ CH:=GETMATRIX((M-1)*8+I,J) ;
35                         WRITESTR(CH:1)                ] ] ;
36                 WRITELN ] ;
37             WRITELN ;
38             WRITELN       ] ; // END PROCEDURE OUTPUT
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
39      #P
40         WRITESTR("1":1) ;
41         WRITELN ;
42         WRITELN ;
43      ONE :
44         FOR J:=1 TO 8 DO
45           FOR I:=1 TO 80 DO SETMATRIX(I,J,".") ;
46         MAT := 0 ;
47      TWO :
48         WHILE QUEEN!ROW=8 DO
49           [ QUEEN!ROW:=0 ;
50             IF ROW=1 DO GOTO THREE ;
51             ROW:=ROW-1                ] ;
52         QUEEN!ROW:=QUEEN!ROW+1 ;
53         IF ROW=1 DO
54           [ ROW:=2 ;
55             GOTO TWO ] ;
56         Q:=QUEEN!ROW ;
57         R:=1 ;
58         WHILE R<= ROW-1 DO
59           [ COL:=QUEEN!R ;
60             IF (C=COL) LOR
61                (C=(COL+ROW-R)) LOR
62                (C=(COL-ROW+R))      DO GOTO TWO ;
63             R:=R+1   ] ;
64         IF ROW<>8 DO
65           [ ROW:=ROW+1 ;
66             GOTO TWO     ] ;
67         N:=N+1 ;
68         M:=M+1 ;
69         FOR R:=1 TO 8 DO SETMATRIX(MAT+QUEEN!R,R,"Q") ;
70         IF MAT<>72 DO
71           [ MAT:=MAT+8 ;
72             GOTO TWO     ] ;
73         OUTPUT(80) ;
74         CNT:=CNT+1 ;
75         IF CNT=5 DO
76           [ WRITESTR("1":1) ; WRITELN ; WRITELN ] ;
77         M:=0 ;
78         GOTO ONE ;
79      THREE :
80         IF M<>0 DO OUTPUT(M*8) ;
81         WRITELN ;
82         WRITESTR(" NO OF SOL":10,"UTIONS = ":9) ;
83         WRITEN(N) ;
84         WRITELN ;
85      FINISH
```

```
PROGRAM CONTAINS 85 LINES AND 3 PROCEDURES
  PROGRAM STATISTICS :
   ** INSTRUCTIONS      :     377 WORDS
   ** RUN TIME  STACK   :    3072 WORDS
   ** CONSTANTS TABLE   :      29 WORDS
   ** FOR LOOP  STACK   :    1024 WORDS
   ** REGISTER  STACK   :     128 WORDS
   ** PROC-NAME STACK   :     128 WORDS
   ** LIBRARY   SIZE    :     456 WORDS
  TOTAL PROGRAM SIZE    :    5214 WORDS
END BCPL *** LEVEL 1 *** 0 ERRORS FOUND DURING COMPILATION . TIME USED :   2.408 SECONDS
```





NO OF SOLUTIONS = 92

APPENDIX G. THE NUMBER PI PROGRAM

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
 1      //PI
 2
 3        STATIC [ N : 1 ; //
 4                 I : 0 ; //
 5                 L : 0 ; //
 6                 P : 0   //
 7                       ] ;
 8
 9        LET PI = VEC 30
10        AND T  = VEC 30
11        AND T1 = VEC 30 ;
12
13        LET MULT(A,N,B) BE
14          [ STATIC [ K:0 ; I:0 ; J:0 ] ;
15            FOR I:=30 TO 2 BY -1 DO
16              [ J:=N*A!I+K ;
17                K:=J/100000 ;
18                B!I:=J-K*100000 ] ;
19            B!1:=N*A!1+K                ] ; // END PROCEDURE MULT
20
21        LET DIVID(A,N,B) BE
22          [ STATIC [ K:0 ; I:0 ; J:0 ] ;
23            FOR I:=1 TO 29 DO
24              [ J:=A!I+K*100000 ;
25                B!I:=J/N ;
26                K:=J-B!I*N         ] ;
27            B!30:=(A!30+K*100000-(N+1)/2)/N+1 ] ; // END PROCEDURE DIVID
28
29        LET ADD(A,B,C) BE
30          [ STATIC [ K:0 ; I:0 ; J:0 ] ;
31            FOR I:=30 TO 2 BY -1 DO
32              [ J:=A!I+B!I+K ;
33                K:=J/100000 ;
34                C!I:=J-K*100000 ] ;
35            C!1:=A!1+B!1+K            ] ; // END PROCEDURE ADD
36
37        LET OUTPUT() BE
38          [ STATIC [ I:0 ] ;
39            FOR I:=1 TO 25 DO
40              [ IF PI!I<10 DO
41                  [ WRITESTR("0000":4) ; WRITEN(PI!I) ] ;
42                IF PI!I<100 DO
43                  [ WRITESTR("000":3)  ; WRITEN(PI!I) ] ;
44                IF PI!I<1000 DO
45                  [ WRITESTR("00":2)   ; WRITEN(PI!I) ] ;
46                IF PI!I<10000 DO
47                  [ WRITESTR("0":1)    ; WRITEN(PI!I) ] ;
48                IF PI!I>=10000 DO WRITEN(PI!I)              ] ] ; // END PROC OUTI
49
50        PI!1:=50000 ;
51        T !1:=50000 ;
52        T1!1:=    0 ;
53        FOR I:=2 TO 30 DO
54          [ PI!I:=0 ;
55            T !I:=0 ;
56            T1!I:=0  ] ;
57      ONE :
58        MULT(T,2*N-1,T) ;
59        DIVID(T,8*N,T) ;
60        DIVID(T,2*N+1,T1) ;
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
61        ADD(PI,T1,PI) ;
62        N:=N+1 ;
63        L:=1 ;
64        WHILE L<=26 DO
65          [ IF T!L<>0 DO GOTO ONE ;
66            L:=L+1                ] ;
67        MULT(PI,6,PI) ;
68        P:=PI!1/100000 ;
69        PI!1:=PI!1 REM 100000 ;
70        WRITESTR(" NUMBER PI":10 , " BY USING ":10 , "FIRST ":6) ;
71        WRITEN(N) ;
72        WRITESTR(" TERMS OF ":10 , "THE SERIES":10) ;
73        WRITELN ;
74        WRITELN ;
75        WRITESTR(" ":1) ; WRITEN(P) ; WRITESTR(".":1) ; OUTPUT() ; WRITELN ;
76        FINISH
```

```
PROGRAM CONTAINS 76 LINES AND 4 PROCEDURES
  PROGRAM STATISTICS :
   ** INSTRUCTIONS      :     390 WORDS
   ** RUN TIME  STACK   :    3072 WORDS
   ** CONSTANTS TABLE   :      33 WORDS
   ** FOR LOOP  STACK   :      48 WORDS
   ** REGISTER  STACK   :     128 WORDS
   ** PROC-NAME STACK   :     128 WORDS
   ** LIBRARY   SIZE    :     429 WORDS
  TOTAL PROGRAM SIZE    :    4228 WORDS
END BCPL *** LEVEL 1 *** 0 ERRORS FOUND DURING COMPILATION . TIME USED :    2.425 SECONDS


NUMBER PI BY USING FIRST 215 TERMS OF THE SERIES

3.14159265358979323846264338327950288419716939937510582097494459230781640628620899862803482534211706798214808651328230664709384
```

APPENDIX H. QUICK SORT PROGRAM

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
 1      // QUICK
 2      #N 10000
 3      #S 80000
 4
 5      STATIC   [ I:0 ; CH:" " ; VAL:0 ; TIME1:0 ; TIME2:0 ;
 6                 TIME:0 ; MASK1:$0000000000077777777 ;
 7                          MASK2:$0000000000000007777 ] ;
 8
 9      LET X = VEC 1000
10
11      AND
12
13      QUICK (LB,UB,X) BE
14        [ STATIC [ J:0 ] ;
15          LET REARRANGE (LB,UB,X) BE
16            [ STATIC [ UP:0 ; DOWN:0 ; A:0 ; TEMP:0 ] ;
17              A:=X!LB ;
18              J:=LB ;
19              UP:=UB ;
20              DOWN:=LB ;
21              REPEAT
22                WHILE (UP>DOWN) LAND (X!UP>=A) DO UP:=UP-1 ;
23                J:=UP ;
24                IF UP<>DOWN DO
25                  [ TEMP:=X!DOWN ;
26                    X!DOWN:=X!UP ;
27                    X!UP:=TEMP   ;
28                    WHILE (DOWN<UP) LAND (X!DOWN<=A) DO DOWN:=DOWN+1 ;
29                    J:=DOWN ;
30                    IF DOWN<>UP DO
31                      [ TEMP:=X!DOWN ;
32                        X!DOWN:=X!UP ;
33                        X!UP:=TEMP     ] ; ] ;
34              UNTIL DOWN=UP ; ] ; // END REARRANGE
35          IF LB<UB DO
36            [ REARRANGE(LB,UB,X) ;
37              QUICK(LB,J-1,X) ;
38              QUICK(J+1,UB,X) ] ; ] // END QUICK
39
40      AND
41
```

SOURCE LISTING OF BCPL PROGRAM *** BCPL TRANSLATOR / LEVEL 1 ***

```
42      #P
43      OUTPUT(X) BE
44        [ STATIC [ I:0 ] ;
45          FOR I:=1 TO 1000 DO
46           [ IF (I REM 501)=0 DO [ WRITELN ;
47                                   WRITESTR("1":1) ;
48                                   WRITELN ; WRITELN ; WRITELN ] ;
49             TEST X!I < 10
50                THEN WRITESTR("          ":6)
51                OR   TEST X!I<100
52                       THEN WRITESTR("          ":5)
53                       OR   TEST X!I<1000
54                              THEN WRITESTR("          ":4)
55                              OR   TEST X!I<10000
56                                     THEN WRITESTR("          ":3)
57                                     OR   TEST X!I<100000
58                                            THEN WRITESTR("          ":2)
59                                            OR   WRITESTR("          ":1) ;
60              WRITEN(X!I) ;
61              IF (I REM 10)=0 DO WRITELN ] ; ] ; // END OUTPUT
62
63      FOR I:=1 TO 1000 DO
64        [ READLN ;
65          READCH(CH) ;
66          WHILE CH=" " DO READCH(CH) ;
67          VAL:=0 ;
68          REPEAT
69            VAL:=VAL*10+(CH-"0") RSHIFT 54 ;
70            READCH(CH)
71          UNTIL CH=" " ;
72          X!I:=VAL ] ;
73      WRITESTR("1UNSORTED ":10,"DATA":4) ; WRITELN ; WRITELN ;
74      OUTPUT(X) ;
75
76      COMPASS       TIME   B2+7
77
78      QUICK(1,1000,X) ;
79
80      COMPASS       TIME   B2+8
81
82      TIME1:=((TIME1 RSHIFT 12) LAND MASK1)*1000+(TIME1 LAND MASK2) ;
83    , TIME2:=((TIME2 RSHIFT 12) LAND MASK1)*1000+(TIME2 LAND MASK2) ;
84      TIME := TIME2-TIME1 ;
85      WRITESTR("1SORTED DA":10,"TA TIME US":10,"ED ":3) ;
86      WRITEN(TIME) ;
87      WRITESTR(" MILISECON":10,"DS":2) ;
88      WRITELN ;
89      WRITELN ;
90      OUTPUT(X) ;
91      FINISH
```

```
PROGRAM CONTAINS 91 LINES AND 3 PROCEDURES
  PROGRAM STATISTICS :
   ** INSTRUCTIONS      :     411 WORDS
   ** RUN TIME  STACK   :   80000 WORDS
   ** CONSTANTS TABLE   :      32 WORDS
   ** FOR LOOP  STACK   :      48 WORDS
   ** REGISTER  STACK   :     128 WORDS
   ** PROC-NAME STACK   :   10000 WORDS
   ** LIBRARY   SIZE    :     763 WORDS
  TOTAL PROGRAM SIZE    :   91382 WORDS
END BCPL *** LEVEL 1 *** 0 ERRORS FOUND DURING COMPILATION . TIME USED :    2.570 SECONDS
```

UNSORTED DATA

```
 65379 247452 805718 462424 508883 262521  36541 139448 680157 509497
301607 307941 403269 728493 215551 377127  22192 544713 650498 654368
 57761 306442 914662 586425  60162 582676 707819 515733 851900 668147
977664 191825 358772 365646 444682 789900 336234 214950 737796 765550
722555 437073 839487 458320 232922  72017 365569  75879 232207 705472
 27145 861654 907923  90079 803737 375571 266115 842616 697983 591673
385918 899543 681859 816517 525499 891428 257930 324996 842431  68963
 59970 349384 382390 650325 656110  84378 245868 787576 182843 314665
 45437 967212  55502 948437 435489 267909 633421 154220  89325  17698
666862 140828 409607 165371 355982 270028 819345 958526  85618 715075
961579 838042 976659  29568 743475  38631 366000 711382  39454 137599
680958 264833 100082 316599 836186 696935 318403 679395 614937 334722
940141 574768 302781  11353 463360 168241 358772 918139 688671 836565
812476 824081 422564 360328 478429 177823 410830  85121 533256 270718
830977 620643 493223 782583 183932 347567 587081 470385 215855 789136
804054 498262  27402 250877 810474 136415 184998 197801 209162 655417
768483 471469 438977 777817 141509 184735 591252 774340  22278 192235
605999 149091 307167 330163 191888 975743 140503 747039 316073 641140
949798 893293 596087 528034 534095  74155 197092 716206 698313 557809
132054 979798 200617 416495 855628 274869 692356 597681 848062 798767
588166 897914  53946   3337  51799 953897 560480 286166 103283 171269
196896 284398 764590 466629 674011 758044 621139 275307 761343 165572
579748 636239 557217 582804 282059 153885 320389 163135 855572 790445
213440 636828 659214 982409 709110 340873 551433 413885 819268  87747
 58116 512808 648119 137551 664765 650187 409712 137544 259299  89738
785749 827269 403194 615604 571316 455369 441385 807149 639212 281995
244981 167154 461420 502565 322486 465719 309618 865766 779273 147105
803852 800077 733700 558100 115247 825162 830925 879578 718354 411081
889212 450305 432032 450845 862918 823913 323064 509307 543323 243969
 49713 570092  12717 793386   6419 876201  25486 495649 246666 590423
357158  26738  96876 517712 820218  35698 555950 349562 113757 653907
365876 314178 909553 570686 839628 437149 474606 726243 381497 714393
521449 569729 807956 438754 632308 985409 187300 957634 228379 126503
514586 712916 295670 924060 112869 155072 516089  55106 628420 847080
606181 445320 729609 784638 993315 419383 427317 848460 854547 556948
 47376 751071 338996  11671 478378 411023 759882  40037   4551 135552
884339 988387 864050 259922 312376 910955 136204 106963 402848 936249
582847 609890 293614 749169  12023 189471 232222  84346 397990 184247
576926 592476 617712 178794 277800 193675 926826  26326 173495 105112
310593 472176 748021 847154 158388 495295 728187 103055 312194 424723
648146 429562 856028 373490 782368 274813 907600 567446 109258  28219
446532 922545 609250  64833 603419 119980 555360 801662  16727  10755
790590 190615 453214 203622 512157  54960 933041  94257 941844 143503
591300 507996 536882 326713 369091 869800 106163 277038 722823 887277
299332 234216 670831 826971 325901 715650 900091 459599 270036 639631
  8377 712987 763579 320397 372125 255898 772277 891452 158130  83224
482536 327347 626471 312130 330451 461941 528356 645887 610776  28102
949301 321419 298057 242621 685916  19348  99807 990892 911900 560381
674357  76305 661361 805821 653546 824541 148643 776515 819556 795580
204156 545240 640632 366549 623611  71198 454128 104028 929309 148319
```

916806 430405 290012 535500 878665 77910 941217 817963 472244 876180
628627 493654 776464 882420 40093 253337 682077 772340 8501 681731
261787 438034 10825 91058 676546 795630 145164 536647 12563 738886
763707 73138 960569 769788 138675 993373 377033 66500 240952 341453
900759 372750 655809 785802 269677 458622 22483 543688 61223 545910
777775 219416 472476 984303 458994 95826 607722 17930 866831 731567
633416 242782 929812 981460 113116 609450 512241 277800 804851 545126
467634 878308 85224 614020 198019 680349 790184 454006 511731 187510
401572 554422 791948 547546 868065 498366 945744 701407 573749 992502
287405 220594 639109 446174 845741 180703 377378 824497 436519 426748
790900 459836 508713 22005 872004 622803 995263 915669 502857 775264
351241 481964 198463 88960 258090 328314 13278 236998 162580 584330
779597 561621 391740 895428 546698 88184 965946 871435 663432 777112
784915 285323 350464 855618 386330 571762 736262 709127 212513 18326
15192 986468 942666 540397 396312 318003 70163 337877 554591 84710
76752 489350 16979 418931 266835 275982 895884 161688 822 559852
141897 632350 570308 873323 482863 256647 590208 198922 842384 508689
282350 112027 619374 99478 567073 587628 31069 423537 77459 91717
981833 179312 115707 528650 763838 718204 671051 327154 405136 649034
839431 735878 675452 814559 690039 768413 597113 937162 613658 878961
682525 661933 700749 242210 862468 754966 409234 520865 950800 877786
806152 485637 217903 378094 554659 246656 861961 322731 556448 791227
455943 648385 175361 816889 328476 604428 124038 504935 765709 82961
692720 89118 293287 133751 775973 216256 444734 794079 285048 736241
614395 860570 633985 452372 36621 614643 107285 450484 772715 695338
544251 35886 905196 890687 835704 357063 954823 876655 586063 988332
806536 843527 748420 771651 50517 173813 330604 959319 446069 220711
72076 220976 968393 817289 418074 633835 308695 146278 719557 806005
410910 429023 353437 417872 225689 647263 813160 522360 863712 978245
628073 925203 192903 274779 532157 62721 42235 917514 356692 385792
880571 554416 843214 911167 401944 157534 575846 68982 75821 692110
566679 199545 556785 418224 182448 162249 974747 28029 1267 831120
563742 747430 197723 268981 791698 172442 974212 786898 19678 627555
624034 786386 579229 573932 703481 315201 779889 94314 619089 897300
452905 461728 253169 865201 846122 670707 44034 757304 590976 670335
632755 160285 585045 673215 648983 681238 429678 796049 735388 717290
926480 576145 241571 730845 400738 451341 329938 402793 553962 623922
141142 77929 358408 74514 216053 277786 338481 760079 53213 865070
519148 818438 421027 17256 379544 195549 205885 8614 287877 483565
961767 601308 5971 567825 171918 486278 770445 276846 214729 759647
952126 646413 257197 124396 72816 369578 331190 811398 663566 918931
668359 747472 663197 793591 404761 77366 299354 146120 879392 775369
124136 156633 242595 952037 486598 102927 699017 951119 547404 974179
611251 7747 246027 226715 497735 854989 625241 106317 722719 163615
745708 772600 885421 208101 167349 854760 495642 986976 653297 329779
837392 691563 323766 716452 771297 605250 845481 181642 848906 104714
219576 606629 183440 491269 96574 996360 155083 158741 8141 555605
775633 545313 106481 713560 385258 49608 525376 449970 729214 432211
841546 246582 227821 176860 778333 330273 709608 42040 650081 748728
763629 972528 427141 73696 358194 228117 977648 695845 83096 777051

SORTED DATA TIME USED 986 MILISECONDS

822	1267	3337	4551	5971	6419	7747	8141	8377	8501
8614	10755	10825	11353	11671	12023	12563	12717	13278	15192
16727	16979	17256	17698	17930	18326	19348	19678	22005	22192
22278	22483	25486	26326	26738	27145	27402	28029	28102	28219
29568	31069	35698	35886	36541	36621	38631	39454	40037	40093
42040	42235	44034	45437	47376	49608	49713	50517	51799	53213
53946	54960	55106	55502	57761	58116	59970	60162	61223	62721
64833	65379	66500	68963	68982	70163	71198	72017	72076	72816
73138	73696	74155	74514	75821	75879	76305	76752	77366	77459
77910	77929	82961	83096	83224	84346	84378	84710	85121	85224
85618	87747	88184	88960	89118	89325	89738	90079	91058	91717
94257	94314	95826	96574	96876	99478	99807	100082	102927	103055
103283	104028	104714	105112	106163	106317	106481	106963	107285	109258
112027	112869	113116	113757	115247	115707	119980	124038	124136	124396
126503	132054	133751	135552	136204	136415	137544	137551	137599	138675
139448	140503	140828	141142	141509	141897	143503	145164	146120	146278
147105	148319	148643	149091	153885	154220	155072	155083	156633	157534
158130	158388	158741	160285	161688	162249	162580	163135	163615	165371
165572	167154	167349	168241	171269	171918	172442	173495	173813	175361
176860	177823	178794	179312	180703	181642	182448	182843	183440	183932
184247	184735	184998	187300	187510	189471	190615	191825	191888	192235
192903	193675	195549	196896	197092	197723	197801	198019	198463	198922
199545	200617	203622	204156	205885	208101	209162	212513	213440	214729
214950	215551	215855	216053	216256	217903	219416	219576	220594	220711
220976	225689	226715	227821	228117	228379	232207	232222	232922	234216
236998	240952	241571	242210	242595	242621	242782	243969	244981	245868
246027	246582	246656	246666	247452	250877	253169	253337	255898	256647
257197	257930	258090	259299	259922	261787	262521	264833	266115	266835
267909	268981	269677	270028	270036	270718	274779	274813	274869	275307
275982	276846	277038	277786	277800	277800	281995	282059	282350	284398
285048	285323	286166	287405	287877	290012	293287	293614	295670	298057
299332	299354	301607	302781	306442	307167	307941	308695	309618	310593
312130	312194	312376	314178	314665	315201	316073	316599	318003	318403
320389	320397	321419	322486	322731	323064	323766	324996	325901	326713
327154	327347	328314	328476	329779	329938	330163	330273	330451	330604
331190	334722	336234	337877	338481	338996	340873	341453	347567	349384
349562	350464	351241	353437	355982	356692	357063	357158	358194	358408
358772	358772	360328	365569	365646	365876	366000	366549	369091	369578
372125	372750	373490	375571	377033	377127	377378	378094	379544	381497
382390	385258	385792	385918	386330	391740	396312	397990	400738	401572
401944	402793	402848	403194	403269	404761	405136	409234	409607	409712
410830	410910	411023	411081	413885	416495	417872	418074	418224	418931
419383	421027	422564	423537	424723	426748	427141	427317	429023	429562
429678	430405	432032	432211	435489	436519	437073	437149	438754	438977
441385	444682	444734	445320	446069	446174	446532	449970	450305	450484
450845	451341	452372	452905	453214	454006	454128	455369	455943	458320
458622	458994	459599	459836	461420	461728	461941	462424	463360	465719
466629	467634	470385	471469	472176	472244	472476	474606	478378	478429
481964	482536	482863	483565	485637	486278	486598	488034	489350	491269
493223	493654	495295	495642	495649	497735	498262	498366	502565	502857

504935 507996 508689 508713 508883 509307 509497 511731 512157 512241
512808 514586 515733 516089 517712 519148 520865 521449 522360 525376
525499 528034 528356 528650 532157 533256 534095 535500 536647 536882
540397 543323 543688 544251 544713 545126 545240 545313 545910 546698
547404 547546 551433 553962 554416 554422 554591 554659 555360 555605
555950 556448 556785 556948 557217 557809 558100 559852 560381 560480
561621 563742 566679 567073 567446 567825 569729 570092 570308 570686
571316 571762 573749 573932 574768 575846 576145 576926 579229 579748
582676 582804 582847 584330 585045 586063 586425 587081 587628 588166
590208 590423 590976 591252 591300 591673 592476 596087 597113 597681
601308 603419 604428 605250 605999 606181 606629 607722 609250 609450
609890 610776 611251 613658 614020 614395 614643 614937 615604 617712
619089 619374 620643 621139 622803 623611 623922 624034 625241 626471
627555 628073 628420 628627 632308 632350 632755 633416 633421 633835
633985 636239 636828 639109 639212 639631 640632 641140 645887 646413
647263 648119 648146 648385 648983 649034 650081 650187 650325 650498
653297 653546 653907 654368 655417 655809 656110 659214 661361 661933
663197 663432 663566 664765 666862 668147 668359 670335 670707 670831
671051 673215 674011 674357 675452 676546 679395 680157 680349 680958
681238 681731 681859 682077 682525 685916 688671 690039 691563 692110
692356 692720 695338 695845 696935 697983 698313 699017 700749 701407
703481 705472 707819 709110 709127 709608 711382 712916 712987 713560
714393 715075 715650 716206 716452 717290 718204 718354 719557 722555
722719 722823 726243 728187 728493 729214 729609 730845 731567 733700
735388 735878 736241 736262 737796 738886 743475 745708 747039 747430
747472 748021 748420 748726 749169 751071 754966 757304 758044 759647
759882 760079 761343 763579 763629 763707 763838 764590 765550 765709
768413 768483 769788 770445 771297 771651 772277 772340 772600 772715
774340 775264 775369 775633 775973 776464 776515 777051 777112 777775
777817 778333 779273 779597 779889 782368 782583 784638 784915 785749
785802 786286 786898 787576 789136 789900 790184 790445 790590 790900
791227 791698 791948 793386 793591 794079 795580 795630 796049 798767
800077 801662 803737 803852 804054 804851 805718 805821 806005 806152
806536 807149 807956 810474 811398 812476 813160 814559 816517 816889
817289 817963 818438 819268 819345 819556 820218 823913 824081 824497
824541 825162 826971 827269 830925 830977 831120 835704 836186 836565
837392 838042 839431 839487 839628 841546 842384 842431 842616 843214
843527 845481 845741 846122 847080 847154 848062 848460 848906 851900
854547 854760 854989 855572 855618 855628 856028 860570 861654 861961
862468 862918 863712 864050 865070 865201 865766 866831 868065 869800
871435 872004 873323 876180 876201 876655 877786 878308 878665 878961
879392 879578 880571 882420 884339 885421 887277 889212 890687 891428
891452 893293 895428 895884 897300 897914 899943 900091 900759 905196
907600 907923 909553 910955 911167 911900 914662 915669 916806 917514
918139 918931 922545 924060 925203 926480 926826 929309 929812 933041
936249 937162 940141 941217 941844 942666 945744 948437 949301 949798
950800 951119 952037 952126 953897 954823 957634 958526 959319 960569
961579 961767 965946 967212 968393 972528 974179 974212 974747 975743
976659 977648 977664 978245 979798 981460 981833 982409 984303 985409
986468 986976 988332 988387 990892 992502 993315 993373 995263 996360

APPENDIX I. SYNTACTICALLY ERRONEOUS EXAMPLE PROGRAM

```
SOURCE LISTING OF BCPL PROGRAM                    *** BCPL TRANSLATOR / LEVEL 1 ***

** POSITION  1 ** ERROR  1 ** OCTAL DIGIT EXPECTED
** POSITION  2 ** ERROR  3 ** NULL STRING NOT ALLOWED
     1      MANIFEST [ C1:12 ; O1:$100370085 ; S1:"" ; C2:345 ] ;
                                  1                    2
     2      STATIC   [ A:C1 ; B:0  ; MASK:$77770000 ; STRING:"LINE" ; CH1:" " ] ;
     3      LET PROC(A,B,C) BE
     4        [ IF A=0 DO A:=12 ;
     5          TEST B<>0
     6            THEN [ C:=A*B ;
     7                   LABO : IF C=A DO GOTO LAB1 ;
     8                   C:=C+1 ;
     9                   GOTO LABO ]
    10            OR     B:= 0 ;
** POSITION  1 ** ERROR 33 ** LABEL LAB1        REFERANCED BUT NOT DECLARED
    11          ] ; // END PROCEDURE <PROC>
                  1
    12      READLN ;
    13      READCH(CH1) ;
** POSITION  1 ** ERROR 37 **   STRING   FOUND WHERE     DO      EXPECTED
** POSITION  1 ** ERROR 27 ** COMMAND EXPECTED
** POSITION  2 ** ERROR  6 ** SKIPPED UNTIL HERE
** POSITION  3 ** ERROR 24 **  ; EXPECTED
** POSITION  3 ** ERROR 27 ** COMMAND EXPECTED
** POSITION  4 ** ERROR  6 ** SKIPPED UNTIL HERE
    14      LABO : IF CH1 " " DO [ READCH(CH1) ; GOTO LABO ] ;
                       1                       2            3 4
    15      A:=( CH1 - "0" ) RSHIFT 54 ;
    16      IF A=0 DO A:=99 ;
** POSITION  1 ** ERROR 44 ** TOO MUCH PARAMETERS
** POSITION  1 ** ERROR  6 ** SKIPPED UNTIL HERE
    17      PROC(A,B,B,CH1) ;
                       1
    18      WRITESTR(STRING:4) ;
    19      WRITEN(B) ;
    20      WRITELN ;
    21      FINISH
```

```
PROGRAM CONTAINS 21 LINES AND 1 PROCEDURES
  PROGRAM STATISTICS :
   ** INSTRUCTIONS      :      98 WORDS
   ** RUN TIME  STACK   :    3072 WORDS
   ** CONSTANTS TABLE   :      12 WORDS
   ** FOR LOOP  STACK   :      48 WORDS
   ** REGISTER  STACK   :     128 WORDS
   ** PROC-NAME STACK   :     128 WORDS
   ** LIBRARY    SIZE    :     701 WORDS
  TOTAL PROGRAM SIZE    :    4187 WORDS
END BCPL *** LEVEL 1 *** 11 ERRORS FOUND DURING COMPILATION . TIME USED :    0.672 SECONDS
```

BIBLIOGRAPHY

1. Gries, David, Compiler Construction for Digital Computers, New York, John Wiley and Sons Inc., 1971.

2. Bornat, Richard, Understanding and Writing Compilers, London, The Macmillan Press ltd., 1979.

3. Davie, A.J.T. and Morrison, R., Recursive Descent Compiling, West Sussex, Ellis Horwood Limited, 1981.

4. Richards, Martin, The BCPL Reference Manual, Technical Memorandum No. 69/1-2, The Computer Laboratory, Cambridge, 1971.

5. Ammann, Urs, "On Code Generation in a Pascal Compiler", Softw. Pract. Exp., Vol. 7, No. 3, pp. 391-423, 1977.

6. Balman, Tunç, An Introduction to Compiler Design Techniques, Istanbul, Boğaziçi University, 1980.

7. Turner, D.A., "Error Diagnosis and Recovery in One Pass Compilers", Information Processing Letters, Vol. 6, No. 4, pp. 113-115, 1977.

8. Tenenbaum, A.M. and Augenstein, M.J., Data Structures Using Pascal, New Jersey:Englewood Cliffs, Prentice Hall Inc., 1981.

9. Knuth, Donald E., The Art of Computer Programming, Vol. 1, Massachusetts, Addison-Wesley Publishing Company, 1968.

REFERENCES NOT CITED

1. Rohl, J.S., An Introduction to Compiler Writting, London, Macdonald and Jane's, 1975.

2. Agarwal, R.K. and Chanson, S.T., "A Space Efficient Code Generation Scheme for BCPL", Softw. Pract. Exp., Vol. 10, No. 1, pp. 77-95, 1980.

3. Jensen, Kathleen ano Wirth, Niklaus, PASCAL User Manual and Report, New York, Springer Verlag, 1974.

4. Pascal Version 1 Reference Manual, CDC Publications, No. 60497700.

5. COMPASS Version 3 Reference Manual, CDC Publications, No. 60492600.

6. NOS Version 2 Reference Manual, Volume 4, Program Interface, CDC Publications, 60459690.