CANADIAN THESES

THÈSES CANADIENNES

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

THIS DISSERTATION HAS BEEN MICROFILMED EXACTLY AS RECEIVED

LA THÈSE A ÉTÉ MICROFILMÉE TELLE QUE NOUS L'AVONS REÇUE

Canada

Static and Dynamic Profiles of Pascal Source Code: A Debugging Utility

Ruth Tedford

A Major Technical Report

In

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

August 1985

ISBN 0-315-30630-0

ABSTRACT

Static and Dynamic
Profiles of Pascal Source Code:
A Debugging Utility

Ruth Tedford

This report presents a profiling system which uses a syntax driven preprocessor to insert monitoring actions into a Pascal program source text. It overlays the statistics on the re-formatted source code. The profiling system is useful in analysis as well as for testing and validation of Pascal programs.

ACKNOWLEDGEMENTS

I should like to thank Larry Thiel, the Senior Analyst in the Computer Science Department at Concordia University for his help in acquiring and analysing the Pascal compiler used in this work.

I should also like to thank Prof. J. Opatrny for his guidance and patience during the early phases of this work.

I gratefully acknowledge the editorial assistance of Kevin O'Mara in the preparation of this report.

I am indebted to Prof. W. Jaworski both for his supervision and for the the use of his ABL/W4 methodology in the presentation of this report.

Finally, if it were not for the longstanding moral support and understanding of my husband John and our children, Kerry and Lisa, I would never have finished this work.

TABLE OF CONTENTS

CH.1 INTRODUCTION

1.10 SOFTWARE PRODUCTION DIFFICULTIES

The feeling of a software crisis within the computing community arose in the last decade from the realization that the reliability and productivity of software was not able to meet the advances and requirements of the evolution that confronted them in both the academic and industrial workplace [Boeh79]. Programs did not meet specification; did not meet production schedules; were not sufficiently adaptable and were not even efficient [Berg79a].

The software difficulties confronted computer science with very serious problems which, in retrospect (not surprisingly), were poorly handled and gave impetus to the birth of software engineering [Berg79a, Grie79, Grif79, Pete81, Wass78]. The structured programming methodology which was advanced as the computer scientist's version of the time-proven, systematic approach utilized by other engineering disciplines in times of need has not resolved this situation [Abra75, Berg79b]. The premises involved in the reincarnation of this panacea involved three essential factors : the elimination, or at least restriction of the GOTO; the adoption of top-down design; the emphasis on program "Quality". Industry has shown that this simple, Lysenko-like, solution to a complex problem was naive

[Broo77, Holt77, Knut79, Pete81]. In spite of our inability to approach the basic problems underlying the software crisis our industry has prospered with the flawed technology available to it. Our society may yet pay for this [Abra75].

As the information processing revolution spread, the industrial view of software production encompassed not only the design and implementation of a software product but also its maintenance, refinement and revision to an everwidening array of industrial applications [Wass80].

The concept of a software lifecycle emerged from the literature at the beginning of this decade [Boeh73, Wein71]. Boehm's model of this lifecycle is given in figure 1.1.

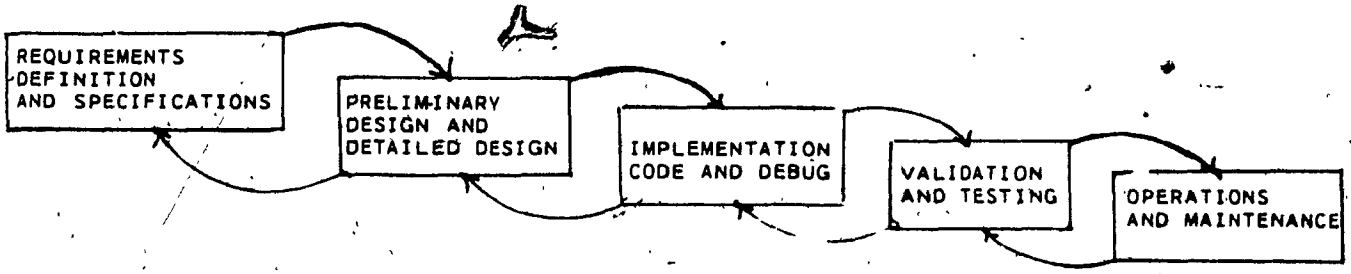


FIG. 1.1 A Model of the Software Lifecycle.

The importance of this concept rests in the realization that post-production maintenance costs of industrial software far exceed the initial development costs [Berg79a, Broo77]. To make matters worse, studies showed that the testing and integration phases of the development costs were very expensive. In fact 40-50% of development costs could be attributed to testing and integration [Boeh73, Boeh79]. Finally it became clear that the actual coding of a system was by far its most inexpensive and trivial phase. Development design considerations far exceeded coding phase expenses [Boeh79]. Typical decomposition figures from this period are given in figure 1.2.

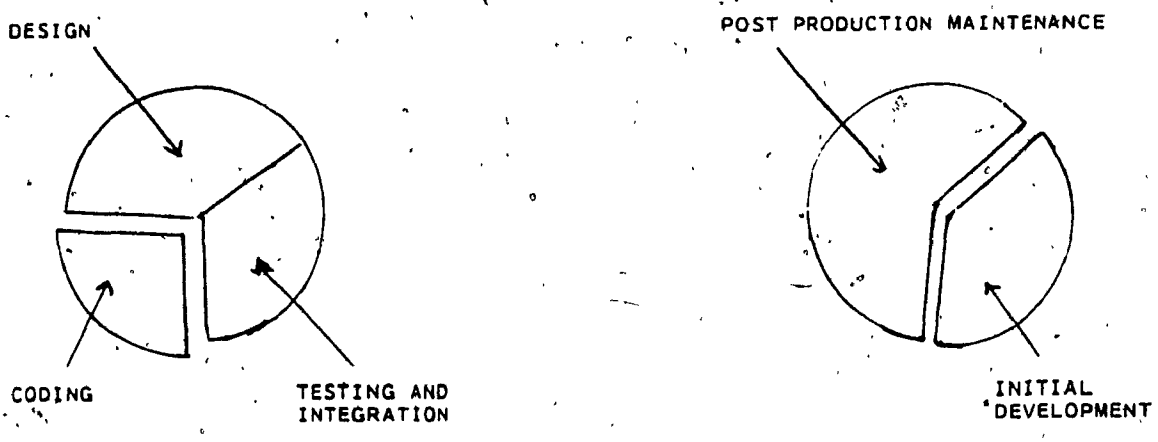


FIG. 1.2a Costs incurred during initial developement

Fig. 1.2b Costs incurred during software system lifetime

A recent issue of the Communications of the ACM offered a case study report [Prel84] which indicates that the problem of predicting and managing software production costs has not somehow resolved itself. Time and experience have shown the structured program methodolgy to be a naive approach which oversimplifies the solution to a complex reality [Shne75, Chap79, Maye81].

1.11 SOFTWARE QUALITY

The discipline of structured programming focused its attention on the production of "Quality" code. The production of a piece of code is the least expensive phase of the software lifecycle. The caveats and recomendations for code production within the structural program methodology do not treat the urgent needs for powerful tools and environments for the repair, maintenance and testing of software. Some of these issues were naively addressed [Dijk79, Shei81] by attempts at mathematical proofs of software. While such approaches are of unquestionable academic value, their effect and cost [Adri82, DeMi77] far outweighs their desirability in most situations. The proofs of such efforts must themselves be checked for errors in derivation and are of no use in locating errors in the design specification or those that arise as a misunderstanding in the code's implementation [Dijk72]. Mandatory testing is still needed for a program that has

been proven correct [Boeh79]. The development of expert systems for proving, testing and evaluating code performance will, it is hoped, alleviate most of these problems.

Although the implementation of an algorithm may be expunged of "undesirable" control constructs, a program's "Quality" cannot be guaranteed by adherence to doctrine. A program's form on the other hand can be greatly influenced by the discipline in which it was written. The structured programming methodology was disappointing to professionals in industry whose preoccupations were very different from those of the publication and proof of programs.

Software quality and the development of strategies for the comprehensive testing and validation of programs remains a major concern of industry.

1.12 SOFTWARE SALVAGE

It is clear that the software lifecycle involves a ceaseless iteration. Software engineering now views the problems involved in the transformation of an algorithm to a program with the same degree of value and attention as those inherent in the transformation of a piece of code into the design document of the algorithm that it implemented. The latter process is referred to as software salvage [Snee84]. It reflects an environment in which an enormous amount of

functional software exists in poorly documented, low level or archiac language. Industry is faced with the problems of upgrading and maintaining this code. It is unrealistic to expect this problem to evaporate. The disdain that we have for inherited software must be overcome in more productive ways. If our field advances as it should our legacy may be just as well appreciated by our successors! Software salvage, while not popular or really necessary in an academic environment, must be viewed as the response to cost/benefit analysis in the marketplace.

This report documents the work on the design and implementation of an experimental software utility which could be used in a system for software salvage. This utility's implementation and testing was completed in 1980. It was originally intended to simply serve as a de-bugging utility in a Pascal teaching environment. Since 1980, and concomitant with the rise in popularity of Algol-like languages, the software environment for learning and using Pascal has improved dramatically.

In the following chapters we shall discuss the software environment in which this utility was implemented and tested in addition to briefly discussing its present uses.

1.20 THE SOFTWARE ENVIRONMENT

Much in the same way that the products of human thought and understanding are bounded by domains [Shne77, Broo77, Wino77, Sche63], software, whether it be the UNIX operating system or a simple routine, is comprehensible, exists and executes in an abstraction that is referred to as its environment.

1.21 SOFTWARE ENGINEERING

Software engineering is a field of study that has grown from the realization that one can profit from the manipulation of the software environment [Shei81, Schn79]. Engineering works under the assumption that metrics, even in the presence of only partial understanding, are essential to the profitable manipulation of the environment.

1.22 TOOLS AND METRICS

An absolute essential to such endeavours are tools and metrics. Software engineering has profited from many pioneering efforts, which through trial and error, have led to our art of programming. Studies of the computational complexity and the time / space requirements of algorithms in addition to software metrics such as those of Halstead [Hals73], Orr [Orr 79], Zweber [Zweb79] and McCabe [McCa76]

have greatly enhanced our views of software. Concepts such as decision tables [Mont74], guarded commands [Dijk75] and D-D Graphs [Paig75] have aided the development of profiles of software. It appears that little consideration has yet been given to a formal semantic analysis of computer code.

Still, with the primitive tools available to it, software engineering has not only been able to decompose and profile the data and control flow of algorithms but has been able to isolate and characterize the operators and operands of an algorithm and to specify their number and relative proportion in hierarchial, apparently, fractal-like form [Hals73], in a program.

1.23 REPRESENTATIONAL DISTORTION

Representational distortion is an effect which has been noted throughout the history of study into human problem solving [Fico83]. This effect is perhaps best appreciated when one attempts to handle a clearly recursive problem through iteration (or vice versa). Similar examples present themselves in the choice of data structures or the harmony of something like a sorting routine and its data structure [Turn80]. Representational distortion occurs as an ill-posed or inappropriate analogy in an argument. Unfortunately a way of measuring the relative wisdom of a problem's statement still escapes us.

The goal of a metric for representational distortion is to allow one to be able to reduce or transform a problem to its simplest form. Having analysed and solved the simple problem one hopes to convey this solution in its transformed form to the actual problem. This technique has a long, eventful, and somewhat sordid, history in the linearization of mathematics.

1.24 HUMAN FACTORS

In our attempts to understand and compare problems and their solutions we often resort to principles such as common denominators. The effects of representation distortions in such situations can become severe [Shaw80], particularly if the analysis environment forces its user to interpret or think in a mode that is awkward, or even very awkward, to his expertise which may be quite refined. Such human factors often drive people, in desperation, to write their own computer language and thus in some small way contribute to our ever increasing Tower of Babel. Even such multi-faceted legends as Knuth [Knut79] have complained of the difficulties of representational distortion. Computer professionals once exposed to the luxury of sophisticated software development and support environment have a hard time returning to the reality of more thoughtless and poorly managed software environments. Pathetically it is often hard to determine if the support environment increases or

decreases the amount of trouble imposed on a user who is attempting to test, modify or create some software. This problem becomes more serious as the sophistication of the programmer and his/her software increases. Today's naive user is in many cases far more educated than the author of the software that encompasses or encombers him or her.

1.30 SOURCE CODE PROFILES

In natural language there is a clear cut difference between linguistic performance and competence. This divergence can also be observed in other coherent condensation of human thought such as programs that are actually executable by a machine.

The desire to relieve a programmer of menial tasks led to the development of high-level languages. These languages attempted to implement the details of coherent thought in a way which would not significantly defeat the efficiency of its computation. Serious constraints must be placed on the interpreters and compilers of such high level languages if one is to achieve an efficient representation in machine code of a language which can be analysed by available techniques in an environment which allows for the detection of error and analysis of performance in its encoding of some algorithm. This task involves the semantic decomposition and subsequent transcription of the code in a manner which

is both computationally efficient and understandable to its author and/or user.

Understandability is a concept which is best viewed in context. Because understandability is a locally defined, potentially universal ideal it has been the object of futile philosophical debate since Aristotle.

2.10 STRUCTURED DECOMPOSITIONS

Common to all phases of the software lifecycle is the human need to profile and decompose (or at a conceptual level to chunk [Newe72]) a problem and its algorithmic solution or implementation.

The type of decomposition best suited to the various phases of the software lifecycle depends on the task at hand. A decomposition of a specification or design document is not likely to offer the necessary and sufficient detail or understanding to a programmer who must profile a piece of code to maintain or upgrade its performance.

Throughout the coding, testing, de-bugging and maintenance phases of the software lifecycle, various software utilities have been developed for and by software engineers. The tasks of semantic modularization are helped [Broo77] even if only at the documentation level by devices such as flowcharts and decision tables. Such adjuncts, however, must be accurately and meticulously maintained if they are to be worth their trouble.

2.11 SEMANTIC MODULARIZATION

Semantic modularization is a very useful [Shne77] cognitive tool for aiding the software engineer's comprehension. Such modularization can allow one to quickly grasp an overview of the code's purpose in addition to providing the essential understanding of its control flow and to a lesser extent its structure.

2.12 PARTITION CRITERIA

Often times, when execution performance is an important consideration, the software engineer must use other criteria to efficiently modularize a program. Most of these criteria are based on expectations and knowledge of the program's data flow. Such partition criteria are sometimes dependent on the computer's hardware. Should parallel processors ever become popular, program profiles and decompositions will be much more concerned with the differential decomposition of sequential and parallel processes.

2.20 STATISTICAL DECOMPOSITIONS

The computer scientist has become gradually aware of the application of statistical decompositions to his art. Such apparently naive decompositions have long proven their power and utility in the analysis of natural languages.

Statistical decompositions in natural languages have been successfully pursued throughout the hierarchial levels of vocabulary [Kuce67], sentence [Wino85], paragraph [Sage85] and even story [Sowa85] structure. Hierarchial statistical decompositions of this sort have yet to be exploited in the analysis of computer programs.

2.21 SOFTWARE PHYSICS

Halstead [Hals77], his collegues [Zweb79] have undertaken the study of algorithms necessary to the development of his laws of software form and function. These researchers were able to show that these laws which describe the relation of operands to operators within a program also function within routines, modules and subroutines across the plethora of languages with which we are blessed and burdened. Not surprisingly, his laws also apply to modularization of non-terminating programs such as operating systems. These laws appear to be extendable to, or derivative from, the domain of hardware structure.

Halstead metrics which are based on his simple laws can be used to access the syntactic quality of an unknown piece of code. They allow one to access the quality of code, its modularization, level and conciseness [Hals77].

2.30 DATA AND CONTROL FLOW

Common to control and data flow decomposition and Halstead's Laws is the intrinsic and underlying concept of the fetch-execute cycle. This essential concept of computation has been known [Benn85] since the time of Babbage. Its cyclical rediscovery draws our attention to the power of this concept in the design and analysis of both machines and algorithms. Dijkstra's guarded command [Dijk79], the semaphore's p-v operators, and some of the various graphic forms of program decompositions [Paig75, Jawo84] as well as language independent environments such as ABL/W4 [Jawo84] are all essentially derivative forms of this concept. Decompositions of a program into fetch-execute cycles whether it be by statistical [Hals77], simple structured (D-D graphs) [Paig75] or semantically enhanced structural decompositions [Wass80, Yode78, Howd82] all involve the fetch-execute paradigm of computation.

2.40 SOURCE CODE UTILITIES

Realistically one must expect an ever-increasing plethora of computer languages and interpreters. Each of these languages has, at least in the eyes of its maker, important advantages over its contempories. Such a situation makes it very difficult to imagine source code utilities for profiling or decomposing code even in our

present software environment. The control flow decomposition of a program is partially dependent on the control constructs of the language in which it was written. Flexible source code utilities must thus resemble expert systems which have access to knowledge of each language, its BNF, operating system interface and machine dependent characteristics. Most of today's source code utilities, in the public sector at least, consist of simple cross-reference maps, autoflowcharting and diagnostic utilities which may be invoked at compilation. These utilities represent the results of a pragmatic and heuristic approach to the problems of debugging what are mostly syntax errors in source code. Run time utilities do exist and execution profiles including source code overlays are now quite common.

This report presents some work on a source code utility for Pascal which predated many of the compiler options for the versions of the Pascal compiler available on Concordia University's mainframe.

This utility overlays on the source code the depth at which a given line of code may be executed and the frequency with which it was executed at run-time. The former characteristic could use the code's tree structure information to drive a typeset utility thus providing a 'pretty-printer' like view of the source code. The latter

characteristic provides information on the relative path frequencies observed at run-time on some specific data.

The design considerations and objectives of this utility are presented in the next section of this report.

Ch. 3 INSTRUMENTED SOURCE CODE: DESIGN CONSIDERATIONS

3.10 OBJECTIVES

The "raison d'être" for software validation is to demonstrate that a computer program satisfies its specifications. On the other hand, validation which is performed by analyzing and testing the code, attempts to determine the degree of correspondance between the code's performance and its specifications. Many validation techniques have been developed. They include: structured walkthroughs [Adri82], static analysis [Adri82], dynamic testing [Rama77] symbolic execution [Good77], and proofs of correctness [Hoar69]. This paper presents some work on a profiling system which addresses the problems found in static analysis and dynamic testing.

The objectives in undertaking work on the profiling system reported here may be cited in terms of the ideal characteristics of such facilities. To quote, an ideal debugging system [Satt72] "leads and aids the programmer in rethinking the design of his algorithm. Thus it should not only help transform an incorrect program into a correct program, but it should also help transform a correct one into a good one." Satterthwaite noted that such systems provide results bounded by practicality. "In particular, unreasonable demands must not be placed on either the system

or the user. The user should not have to spend large amounts of time in preparing additional input or deciphering output. System costs should not discourage the use of the debugging tool." The exact choice of debugging tools to be included in a system of course must reflect the needs of their resident environment. The debugging facilities should be able to be invoked routinely to aid program development throughout the entire life of the program. Thus such systems aid the analysis and documentation of a system as well as the detection and removal of execution or program error. A user friendly environment would provide the infrastructure to assure that all communication could take place at the level of the source language without significantly impeding the compilation and execution of the user's software. The facility's environment should provide a range of debugging tools and techniques, all of which are compatible with the high level language used by the programmer. Needless to say the facility itself should be properly documented and error-free.

3.11 DESIGN CRITERIA

In consideration of these objectives, the following specific design criteria were selected from among those found in the literature [Satt72, Fox178] for the prototype debugging system:

1) The existing implementation of Pascal was used by

students and research workers in a university environment. Upon completion, PROFILE was to be accessible to the university community on Concordia's main frame computer.

2) All information presented to the user should be expressed in terms of his source program and the Pascal language.

3) The additional resource requirements placed upon the computing system should be moderate. With this constraint in mind the facility was designed for a batch-processing environment. The profile's storage requirements were kept to a minumum by the dynamic allocation of the counter array at execution time.

4) The profile should be easy to use. It should produce information which encourages the user to reexamine his entire program.

The objectives and design criteria of the profiling facility address not only the potential uses of the systems but the human factors underlying its acceptance within a teaching environment. Considerations of its value and use throughout the software lifecycle involve designing a flexible tool which may be used as a debugging, testing, documentation and validation aid. The overhead required to use this tool should not place excessive demands on the computer system or its users. The facility's ability to

partially disassemble control flow from the source code, make it useful to work on program comparison and software salvage. A cost benefit analysis of such software tools is beyond the scope of this report. Consequently we shall not consider the important questions of what necessary and sufficient information is needed to fully profile a program. Nor will attention be placed in this report on the cost and criteria for triage within such a system.

3.20 DE-BUGGING UTILITIES

3.21 STATIC ANALYSIS

Static analysis and dynamic testing are best viewed as complementary approaches to software validation. Static analysis provides global information concerning the program structure whereas dynamic testing investigates the runtime behavior of the program. When performed prior to dynamic testing, static analysis can provide information useful for test planning, thus providing for an economy in the selection of the number and importance of test cases deemed necessary for sufficient testing.

The following information can be easily obtained from static analysis :

1) number of occurences of source statements by type
2) initialized variables

3) variables set but not used

4) subroutines and functions called by each routine

5) analysis of how the identifiers are used in each statement

6) cross-reference maps of identifier usage

Most static analyzers are experimental prototypes developed to investigate the feasibility of automated static analysis. These systems allow experimentation with static analysis theorems. Encouragingly a number of useful static analyzers have been implemented [Myer79].

3.22 LIMITATION OF STATIC ANALYSIS

Decidability results state that theoretically "there is no procedural way to detect the semantic paths through an arbitrary program written in a general purpose programming language." [Fair78] Static analysis is a powerful, but inherently limited cognitive adjunct. In fact, oftentimes dynamic testing can can be used to acquire information that is difficult or impossible to obtain by static analysis. For example, subscripts and pointers can not be evaluated and thus we are unable to distinguish between elements of an array or members of a list.

3.23 PROGRAM GRAPHS

Control graphs have been used to measure program complexity [Paig75]. Graphs in which the graph's nodes represent program units are quite useful in dynamic testing in that they provide a basis for test-case planning because they illustrate the paths through the graph's nodes that have been executed by various test cases.

3.24 TIME MEASURE

Timing information is obviously essential for improving the efficiency of a program. It has been found that a greater proportion of the execution time of a program is spent in a small portion of the code [Rama77]. Several measures of timing have been traditional in software metrics. Response, CPU and elapsed time, in addition to the relative time in each iteration or statement, provide useful statistics for the software engineer. Among the information the profile should provide is an accurate metric of the absolute and relative access frequency of the source code segments. This information, in conjunction with a simple analysis of the computational complexity of the code's segments, will allow the user to estimate the relative CPU time spent in these code segments.

3.25 TEST CASES

Test cases are essentially samples from the space of all possible execution sequences. While it is true that testing cannot guarantee correct operation of software under all operating conditions, a well structured set of tests is often more convincing than a correctness proof.

3.30 RUN-TIME ERRORS: DYNAMIC TESTING

Function tests are intended to demonstrate that the software performs satisfactorily under normal operating conditions. This involves computing nominally correct output values from nominal input values.

Logical tests are concerned with the manner in which the code performs its computation. "Ideally, every logical path through the code should be executed to determine nominal behavior and exception handling on that path. Of course this is usually infeasible because of the vast number of test cases required for even relatively simple programs" [Fair78].

Although there is no agreement on the definition of a "thorough" test, a "minimumlly thorough" test requires that each program branch of the code be exercised at least once during the testing process [Huan77]. Clearly, if a

statement has not even been executed during the testing process, the action performed by that statement has not been tested. Unfortunately, executing every program branch during testing does not imply program correctness because it does not guarantee that every path has been traversed. Since formal program verification is often impractical, the main emphasis is placed on testing as a technique for improving software reliability.

3.31 DYNAMIC TESTING METHODOLOGY

Three strategies that have been used [Leve82, Boeh79, Pete81] to systematically test a software system are: bottom-up, top-down, and mixed mode testing.

3.32 AUTOMATED DYNAMIC TESTING

Various authors [Bagg78, Wass80] are of the opinion that specific information may be used for testing purposes. In particular they cite: structural information; execution summary statistics; control-flow, data-flow, and environmental information: dynamic assertion checks. By structural information they mean the static structure of the program text including static nesting levels. Execution summary statistics include: statement execution counts. timing estimates, and the ranges of variables, Control-flow information is consistent with the sequences of statements

executed. Data-flow information is determined from the sequences of values assigned to variables. Assertions are checked by comparing asserted behavior to actual behavior.

3.40 VERIFICATION

Typically there are two situations in which systematic verification occur: system verification which is undertaken prior to the release of the software and reverification which must keep pace with software maintenance. Maintenance is required in order to correct a problem or to modify the capabilities of an evolving system. Reverification must fulfill two distinct goals. The first of these involves verifing the corrected or new capability of the system. Revalidation must also verify that no other capability of the system has been adversely affected by the side effects of the modification.

3.41 TEST STANDARD

In any systematic validation effort, the specifications are the standard against which code is verified. Two types of specifications that a program must satisify are: functional specifications which state what a program should do and design specifications which state how the program should do it. Consequently, two categories of errors may be discovered: functional errors which are departures from

requirements and logical errors which are errors in implementing the design.

Many of the problems of verifying software are caused by faulty specifications [Boeh79]; however, the size, complexity and evolutionary nature of software systems make the development of satisfactory specifications a very difficult task indeed [Pete81]. An ultimate goal of the testing process is to so thoroughly test the code that the likelihood of any logical errors remaining in the program is extremely unlikely. This distinguishes testing from program proving, in which the objective is to demonstrate that no logical errors exist in the program relative to the given specification.

Program testing is recognized as a critical component in the software development process [Boeh79]. Unfortunately while program testing is an essential phase in the production of reliable software, minimal standards are often not met. In fact, "tested software is often delivered to a user containing statements or branches which have never been exercised during testing" [Prob80]. J. Huang [Huan77] cites a realistic criterion when he suggests that every arc in the program graph be traversed at least once by some test case.

3.50 AUTOMATED TOOLS FOR IMPROVING SOFTWARE QUALITY

A common tool for improving the quality of code are software probes. Such probes in the form of source language statements can be judicously inserted into the source code to gather statistics during program execution [Paig74a, Rama77]. By inserting the software equivalent to "designed sensors" into the subject program each time that the software under the testing strategy performs a "significant event", the occurrence of this event is recorded. Subsequently, these probes can be used to gather run-time statistics about the exercising of all portions of the program during the testing process. This information may be printed out immediately or stored for later retrieval and report generation.

Software probes can provide insight into many aspects of algorithmic behavior beyond that of a simple analysis of control flow . Probes may be thus designed in tandem with testing strategies to locate specific faults in the code. For example, a simple FORTRAN implementation of a Program Evaluator and Tester (PET) has demonstrated the value of a self-metric testing approach for higher-level languages [Stuc77]. Automated tools that involve the expansion of this concept are now being explored as a vehicle for improving software quality. Such tools provide dynamic program analysis by using a self-metric (self-measuring)

software.

Although probes can perform several functions, in this report, we consider only the thoroughness test function.

Program instrumentation is the process of inserting software probes into a program. To minimize the execution and memory overhead, it is, of course, desirable to insert the fewest number of probes which can be used to provide the necessary coverage. We shall examine this in more detail in the next chapter.

3.60 ABL/W4 REPRESENTATION

The principles and practice of the ABL/W4 methodology have been reported in detail elsewhere [Jawo82, Fico83, Hint81, Fico85, Jawo84]. Suffice it to say that this language independent methodology allows one to provide the necessary and sufficient documentation to describe an algorithm to an arbitrary level of detail. The ABL/W4 methodology also cleanly isolates an algorithm's control flow from its code. Both of these attributes will be exploited in the Appendix where we will describe the details of the external procedure call WPROFIL and the profile program MONITOR.

PROFILE programs and their procedures will be

documented in the ABL/W4 tabular format in which the control flow of the algorithm is specified in the strategy matrix. The alternatives or columns of this matrix correspond to the concept of commands. The set of alternatives associated with a given row or cluster of the strategy matrix represents all possible paths leading from that decision point (or cluster of alternatives) in the program.

The paths destinations are specified by the next state vector associated with each alternative, or column in the ABL/W4 tabular form. All such paths are restricted in that they may only lead to one of the program's decision points (or cluster of alternatives). The next state vector in the tabular form of ABL/W4 points to a row number index of the strategy matrix. Of course, the paths specified in this manner are the set of arcs or edges E_i in the control graph $G(V,E)$ given by E and the set of nodes or decision points V of the strategy matrix.

Within a given cluster the alternative chosen at execution is specified by the cluster's guard. This guard is a subset of the set of predicates P used in the algorithm. Once an alternative has been specified, a subset of the program's actions, A, is executed in a specific sequence which is given by the ascending numerical order of the actions specified within each column of the ABL/W4 tabular form.

Having briefly described the sets of clusters, alternatives, predicates and actions, the concepts of guarded commands, the strategy matrix and its next state vector, it is now possible to consider the abstract machine specified by some set of predicates and actions. This abstract machine may be used by many different abstract programs. Each of these programs could be different in terms of the action sequences of their alternatives or in terms of the guards on which alternatives within a cluster are selected. Of course, abstract programs could also differ solely on the grounds of the strategy in which guarded commands / alternatives are concatenated together.

PROFILE presented in the next section of the report describes the infrastructure necessary to automatically transcribe standard Pascal into its ABL/W4 representation. PROFILE also provides the statistics on how frequently each path in the program's graph G(V,E) is used during execution. Throughout this refinement, PROFILE was used to analyse itself. PROFILE is documented in ABL/W4 form in the Appendix.

Ch. 4 UTILITY DESCRIPTION

4.1: MONITOR: AN OVERVIEW

An experimental batch software system was designed and implemented to collect and display the execution behavior of batch-mode Pascal source programs. The system described in this report performs four functions :

1. it inserts global counters and increments them
2. it adds the source code necessary to insert the counters
3. it keeps track of all statements added
4. it appends a Pascal subroutine which prints out a listing with frequency counts and logic levels at the end of the program

Monitor, a preprocessor, was implemented to perform a static analysis of the source program. Three files are produced by this process:

- XXXFILE the instrumented source code
- XXXCOMM the control file
- XXXAAA the source code file

The three files produced when profiling the program HANOI (fig 3.1) are shown in fig 3.2.

```
E.T.H. ZUERICH /, UNIVERSITY OF MINNESOTA.                    PASCAL 6000 V3.4.0.  85/08/19. 20.4
 CONCORDIA UNIVERSITY COMPUTER SCIENCE                          NOS 1.4              (84/05/24)  P

000004      1 PROGRAM HANOI(OUTPUT);
000040      2 VAR
000040      3   HEIGHT : INTEGER;
000041      4 PROCEDURE MOVE(HEIGHT : INTEGER; FROMPEG, TOPEG, USINGPEG : CHAR);
000006      5 BEGIN
000006      6 IF HEIGHT = 1 THEN WRITELN('MOVE A DISK FROM ',FROMPEG,' TO ',TOPEG)
000033      7 ELSE BEGIN MOVE(HEIGHT-1,FROMPEG,USINGPEG,TOPEG);
000044      8 WRITELN('MOVE A DISK FROM ',FROMPEG,' TO ',TOPEG);
000073      9 MOVE(HEIGHT-1,USINGPEG,FROMPEG,TOPEG)
000102     10 END
000103     11 END;(* MOVE*);
000120     12 BEGIN
000120     13 MOVE(4,'A','C','B')
000025     14 END.

           COMPILER-ESTIMATED 'W' OPTION = 002450B.
```

fig. 3.1

```
 * *  XXXCOMM   * *      * - - - - - XXXFILE - - - - - - - - - - >

  1     1   1   1     0     PROGRAM HANOI(OUTPUT
  0     0   0   0     0     ,XXXFILE,XXXRR,XXXCOMM
  1     2   1   1     0     );
  0     0   0   0     0     (*$I'XXXTRA'/'XXXAAA'*)
  2     3   1   1     0     VAR
  3     4   1   1     0       HEIGHT   INTEGER;
  0     0   0   0     0     (*$X4*)
  0     0   0   0     0     PROCEDURE WPROFIL(XXXKOUNTER DYNAMIC XXXTYPE;XXXCTRNO INTEGER;
  0     0   0   0     0      XXXPROGID ALFA;VAR XXXFILE, XXXRR, XXXCOMM   TEXT);
  0     0   0   0     0      EXTERN;
  0     0   0   0     0     (*$X= *)
  0     0  99   0     0
  4     5   1   1     0     PROCEDURE MOVE(HEIGHT   INTEGER; FROMPEG, TOPEG, USINGPEG   CHAR);
  0     0  99   0     0
  5     6   1   1     1     BEGIN
  0     0   0   0     0     XXXKOUNTER[1]  = XXXKOUNTER[1] + 1;
  6     7   1   2     1     IF HEIGHT = 1
  6     8   2   3     2     THEN
  0     0   0   0     0     BEGIN
  0     0   0   0     0     XXXKOUNTER[2]  = XXXKOUNTER[2] + 1;
  6     9   2   4     2     WRITELN('MOVE A DISK FROM ',FROMPEG,' TO ',TOPEG)
  0     0   0   0     0     ;
  0     0   0   0     0     END
  7    10   2   3     3     ELSE
  7    11   2   4     3     BEGIN
  0     0   0   0     0     XXXKOUNTER[3]  = XXXKOUNTER[3] + 1;
  7    12   2   5     3     MOVE(HEIGHT-1,FROMPEG,USINGPEG,TOPEG);
  8    13   2   5     3     WRITELN('MOVE A DISK FROM ',FROMPEG,' TO ',TOPEG);
  9    14   2   5     3     MOVE(HEIGHT-1,USINGPEG,FROMPEG,TOPEG)
  0     0   0   0     0     ;
 10    15   2   4     3     END
  0     0   0   0     0     ;
  0     0   0   0     0     XXXKOUNTER[4]  = XXXKOUNTER[4] + 1;
 11    16   1   1     4     END (*,MOVE*);
  0     0  99   0     0
 12    17   1   1     5     BEGIN
  0     0   0   0     0     XXXKOUNTER[5]  = XXXKOUNTER[5] + 1;
 13    18   1   2     5     MOVE(4,'A','C','B')
  0     0   0   0     0     ;
  0     0   0   0     0     XXXPROGID  = 'HANOI     ';
  0     0   0   0     0     WPROFIL(XXXKOUNTER,   5,XXXPROGID,XXXFILE,XXXRR,XXXCOMM);
 14    19   1   1     5     END.

XXXAAA - - ->  XXXTRA
                TYPE XXXTYPE= ARRAY[0..   5] OF INTEGER;
                CONST XXXCTRNO =    5;
                VAR XXXKOUNTER :.XXXTYPE;
                XXXFILE, XXXRR, XXXCOMM : TEXT;
                XXXPROGID : ALFA;
                VALUE XXXKOUNTER = (   6 OF 0);
```

fig 3.2

XXXFILE: INSTRUMENTED SOURCE CODE

All "executable" statements are aligned on the left margin, one statement per line, while the original spacing of the declarations is preserved. The insertion of the software probes requires the addition of statements necessary to satisfy the syntactical requirements of the Pascal language. Implied BEGIN-ENDs must be replaced with actual BEGIN-END pairs and semi-colons must be inserted to seperate statements. The command to include the source code file XXXAAA must be inserted as well as the declaration of the external procedure WPROFIL.

XXXCOMM: CONTROL FILE

For every line in XXXFILE, a line containing the following must be written on XXXCOMM; the source code line number, the line number in the profile, the logical level, the print level (i.e. indentation level), and an index (CtrNo) to the array of counters. Lines inserted into the source code are, with one exception, given zero values. A logic level line number of 99 is used to indicate the beginning or ending of a block at logic level one.

XXXAAA: SOURCE CODE FILE

Source code lines to be added to the instrumented program at compile time are written on this file. These include the declarations necessary to call WPROFIL. The number of counters is set by MONITOR when it completes the preprocessing of the instrumented file.

The instrumented program is then compiled and executed under the the control of the procedure file. This is transparent to the user; he uses the two commands described in APPENDIX A.1.

4.20 PLACEMENT OF SOFTWARE PROBES

The number of software probes was minimized as much as possible. Compound statements in Pascal, with the exception of statements included in a REPEAT-UNTIL loop must be bracketted by BEGIN-END pairs. Thus counters were added after every BEGIN, and after every END that did not satisfy one of the following conditions: an END followed by an ELSE, the END of an alternative in a CASE statement, the END of a procedure or function, or the END of the program. If an implied BEGIN-END pair was encountered, the counter was inserted after the addition of a BEGIN. An END had, of course, to be added in that case. Software probes were also inserted after every REPEAT statement and every UNTIL which did not violate the conditions already mentioned for the END.

Although the number of software probes was minimized, a number of nested statements ending at the same logical point resulted in redundancy. Each END in the nested logic generated a new counter, but only one was utilized.

4.30 - EXTERNAL PROCEDURE WPROFIL: AN OVERVIEW

WPROFIL - ia an external procedure which is transparent to the user and which reads two files (XXXFILE, XXXCOMM) produced during the pre-processing of the host Pascal program. It treats the control file XXXCOMM as a "driver" and sequentially processes each line in this file in tandem with a line from the edited source code file XXXFILE. The processing involves reading a pointer to a value in the array of counters passed from the host program. If the pointer or counter number has changed, a state transition has occurred and the value in the array of counters pointed to by the index (CtrNo) is overlayed on the output file (XXXRR). This occurs whenever the logic level of the program changes. The external procedure prints a right parenthesis to indicate that the counter value for that line is the same as its immediate predecessor.

Since XXXFILE contains only one simple Pascal statement per line (a line in the source program such as IF C1 THEN S1 ELSE S2 is written as three lines on XXXFILE), WPROFIL is able to generate a "pretty printer" output on the profile file XXXRR.

On this output it overlays :

a. the source code line number

b. the profile line number

c. the logic level

d. the value of the counter or ')'

Ch. 5 RESULTS AND CONCLUSIONS

5.1 RESULTS

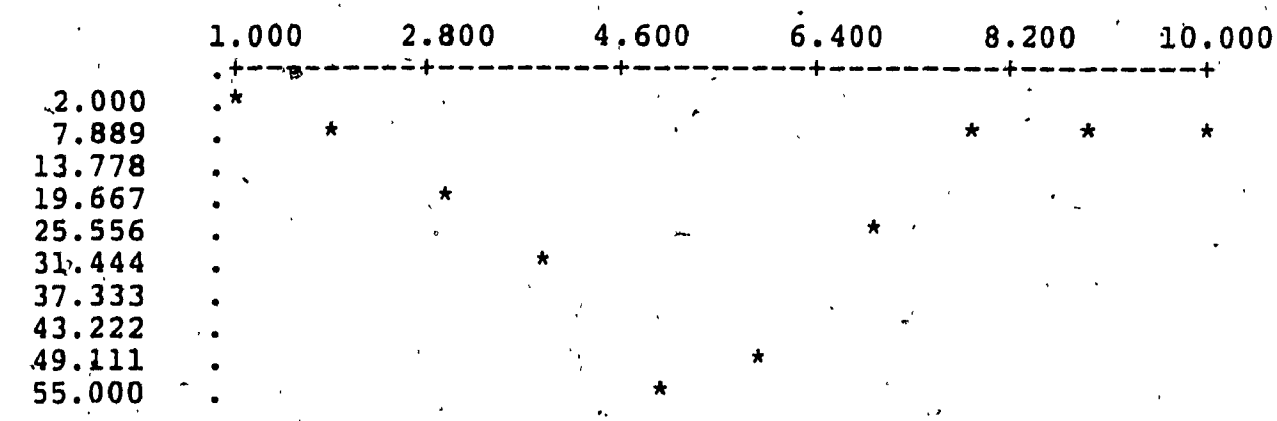
The original program was modified to output the values of the array of execution frequencies so that experimental results could be obtained. Logic level was redefined in the following manner: the main block of the program was considered to be level 1, procedures defined at this level were considered to be level 2, and nested procedures, as 3, 4, 5, etc. depending on the level of nesting. This was added to the original level number.

The number of statements at each logic level was calculated by eliminating BEGIN-END statements as delimiters. An alternative was defined as an increase in the logic level or a change in the index to the array of execution frequencies at the same logic level. The following results were obtained.

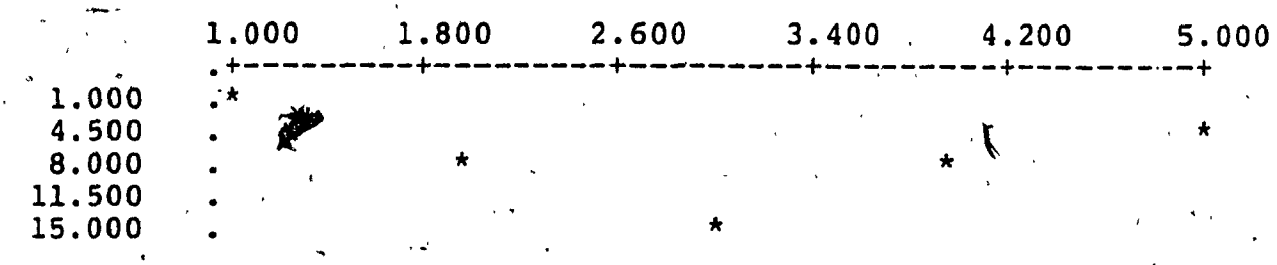
Fig. 5.1, which is a cross tabulation of the number of alternatives vs. the logic level shows the existance of a log n normal distribution between these two variables. This relationship is best observed in the data obtained from the analysis of the Pascal compiler found in fig. 5.1c. The distributions observed for the much smaller examples, the Groupcount program and the Profile program, are less

FIG. 5.1 BASE - NUMBER OF ALTERNATIVES
CROSS-VARIABLE - LOGIC LEVEL

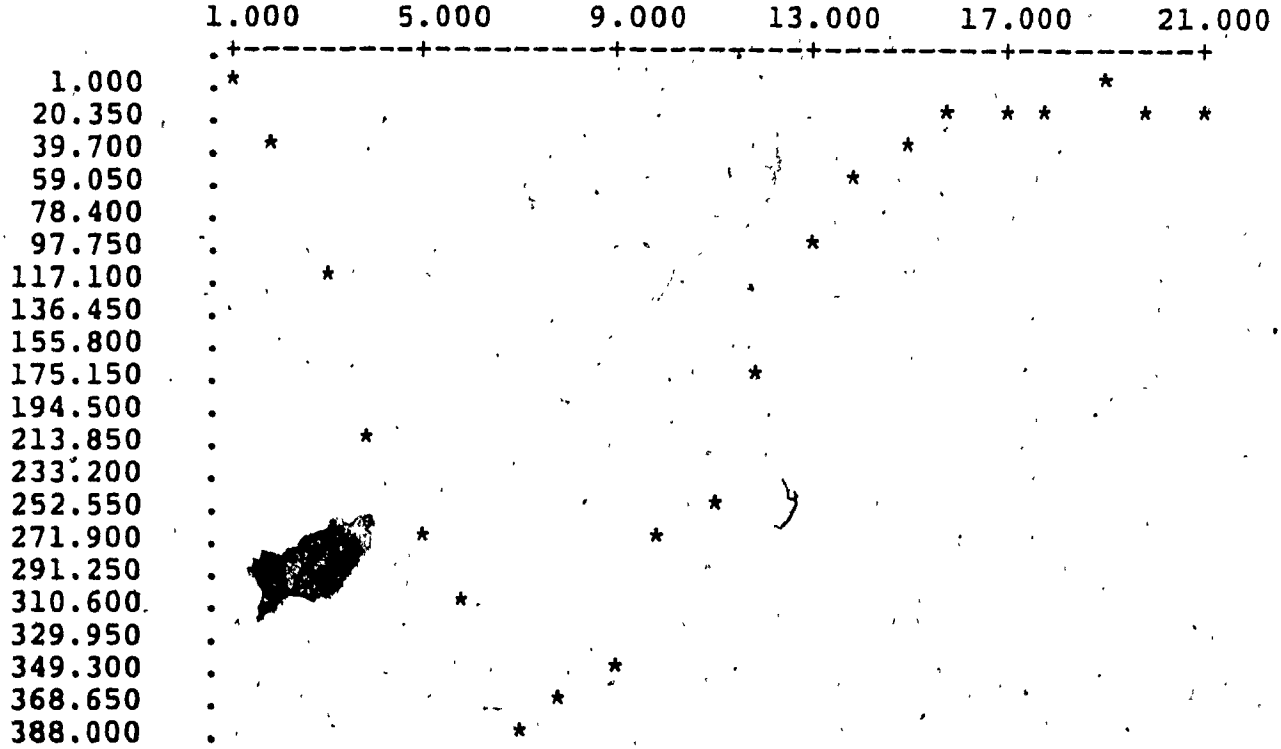
(A) PROFILE



(B) GROUPCOUNT



(C) PASCAL COMPILER



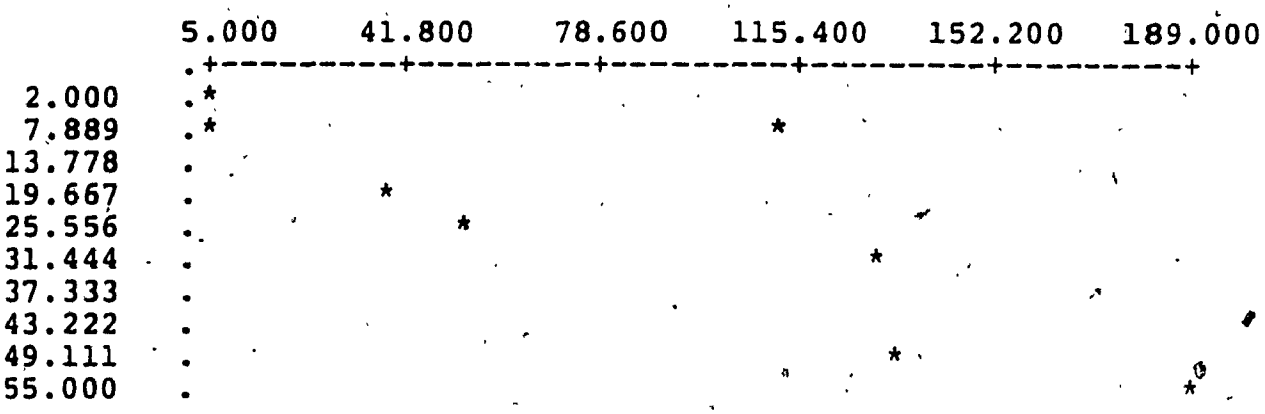
striking than that obtained on the approximately 10000 line Pascal compiler. The distributions observed in figures 5.1a and 5.1b may be Gaussian whereas it is clear that fig. 5.1c is a log-normal distribution.

Fig. 5.2 illustrates the results of cross tabulating the number of alternatives against the number of statements per alternative. Once again the results obtained on the analysis of the Pascal compiler shows a strong clean relationship (correlation coefficient 0.992) whereas the smaller examples demonstrate the effect more weakly (r for Profile - 0.880 and r for Groupcount - 0.768). This may be due to programming style or the effects of outliers which would mask a linear relationship in a small data sample.

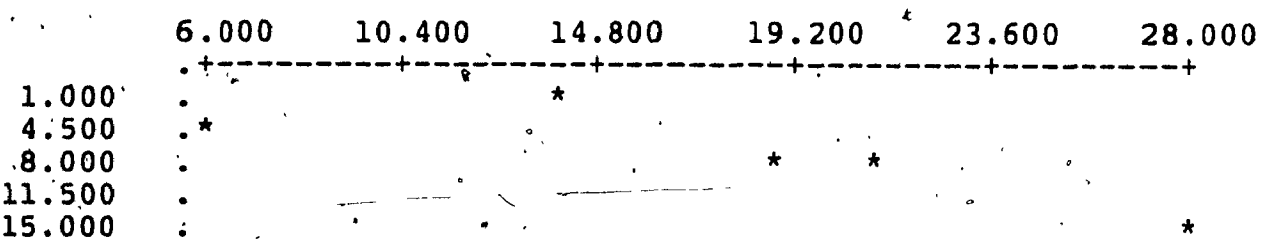
Fig. 5.3 illustrates the observed frequency of execution of the program logic levels. This graph illustrates an interesting effect. There appears to be two independent frequency relationships operating within the sample programs that were tested. This effect is highlighted by the twin exponential curves which have been superimposed on the data points. Obviously there appears to be two classes of logic levels operating within these programs. Both exhibit exponential frequency distributions: one is however of smaller magnitude and the opposite sign from the other. The two exponentials converge to steady state frequency in all three cases. Once again this effect

FIG. 5.2 BASE - NUMBER OF ALTERNATIVES
CROSS VARIABLE - NUMBER OF STATEMENTS

(A) PROFILE



(B) GROUPCOUNT



(C) PASCAL COMPILER

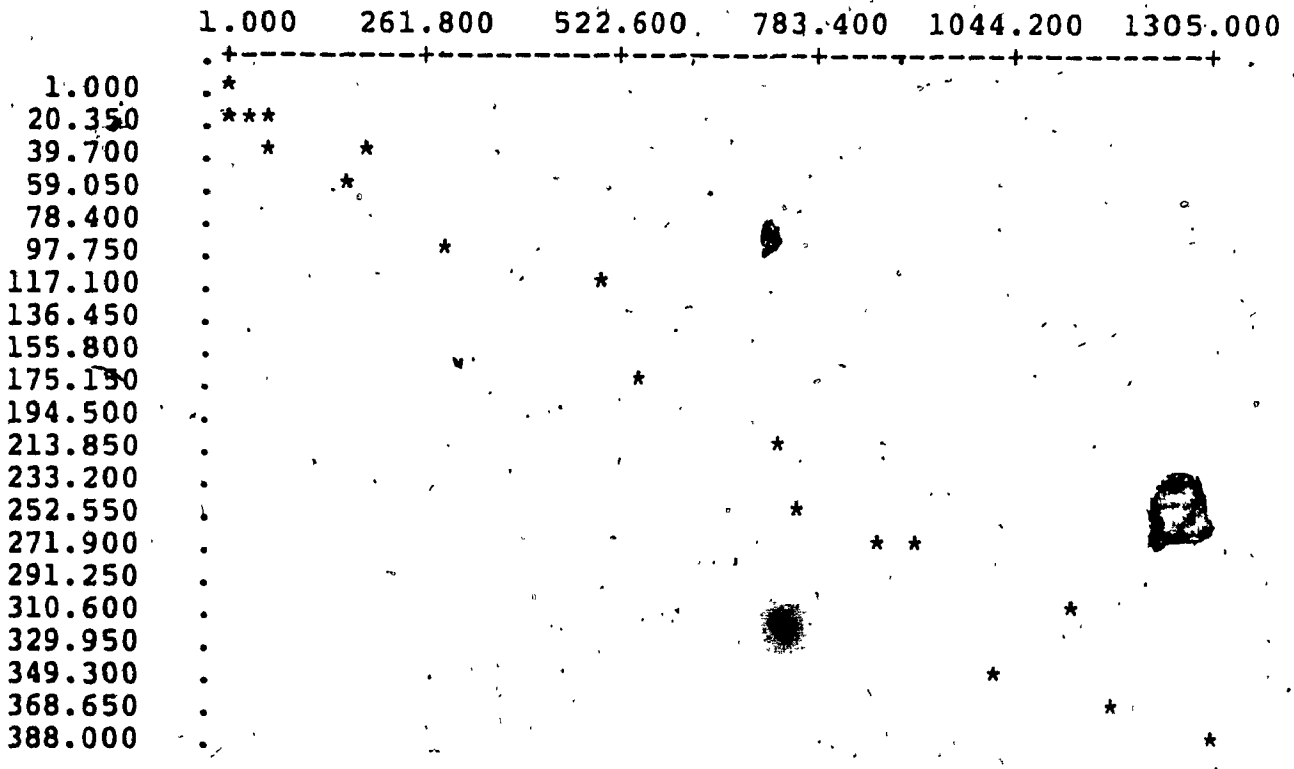
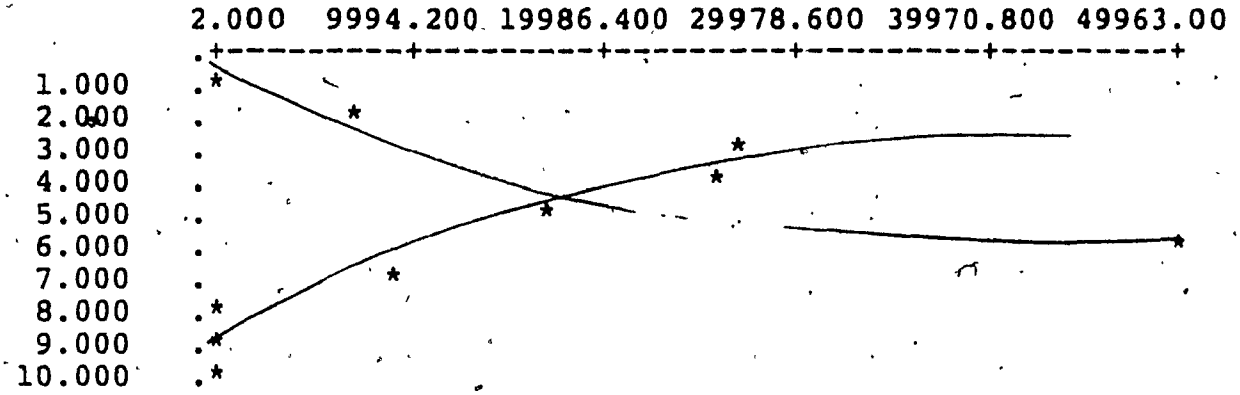
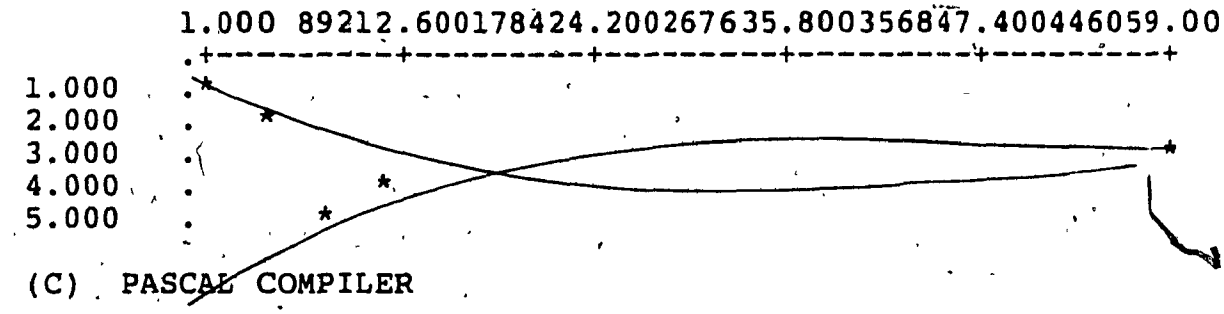


FIG. 5.3 BASE LOGIC LEVEL, CROSS VARIABLE - FREQUENCY

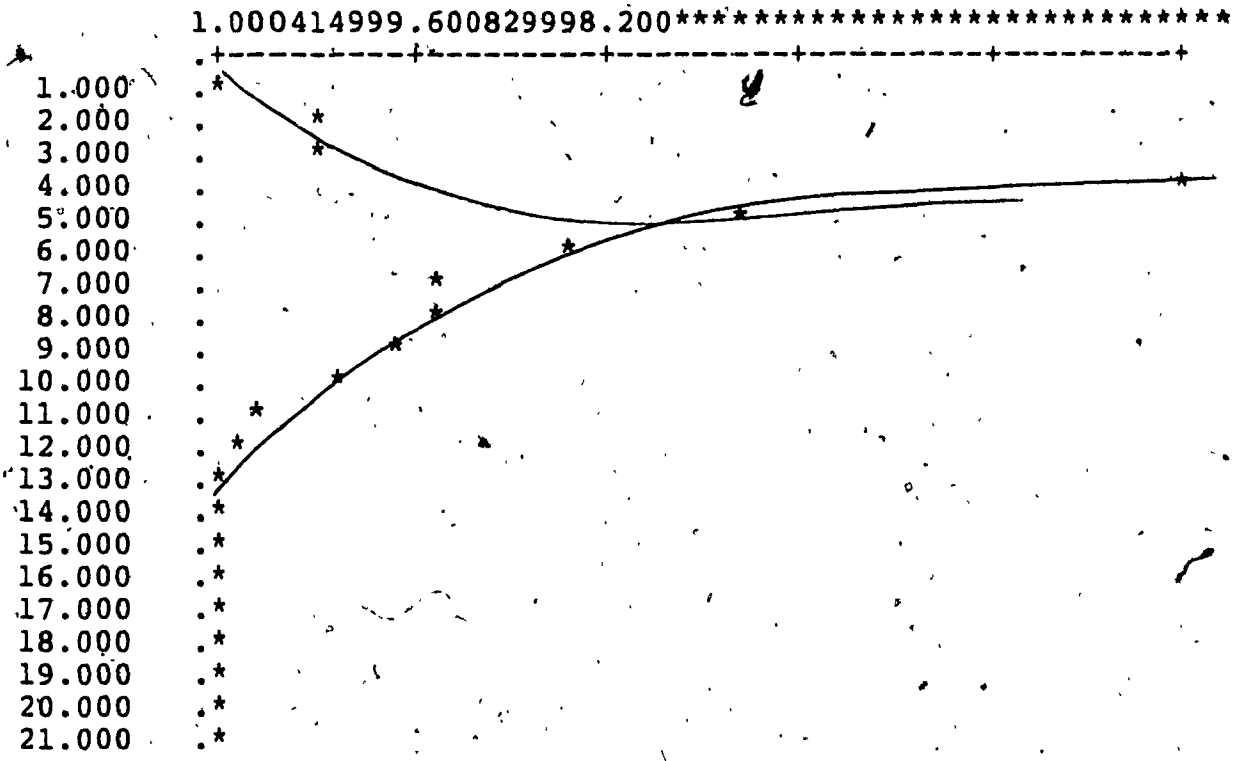
(A) PROFILE



(B) GROUPCOUNT



(C) PASCAL COMPILER



is best observed in the results obtained for the Pascal compiler. The phenomenon that underlies these effects remain to be explained.

Fig 5.4 illustrates the frequency distribution of the alternatives in the three examples studied. There is no clean relationship observable in the data presented. This may be attributed to either a random process or the manner in which the alternative logic level was assigned.

Fig 5.5 illustrates the correlation matrices and elementary statistics for the three cases considered. In these figures:

variable 1 - logic levels

variable 2 - number of alternatives at that logic level

variable 3 - number of statements at that logic level

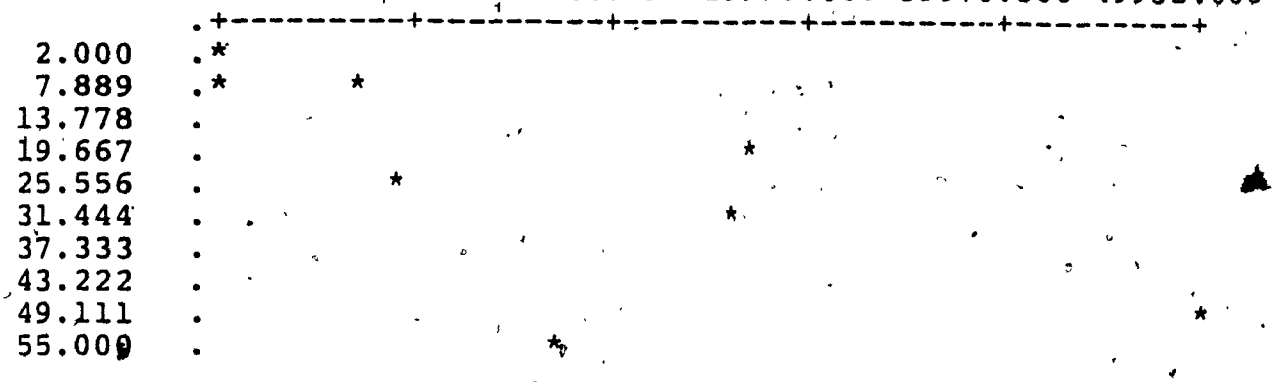
variable 4 - execution frequency of the statements at that logic level

The raw data is presented in fig. 5.6.

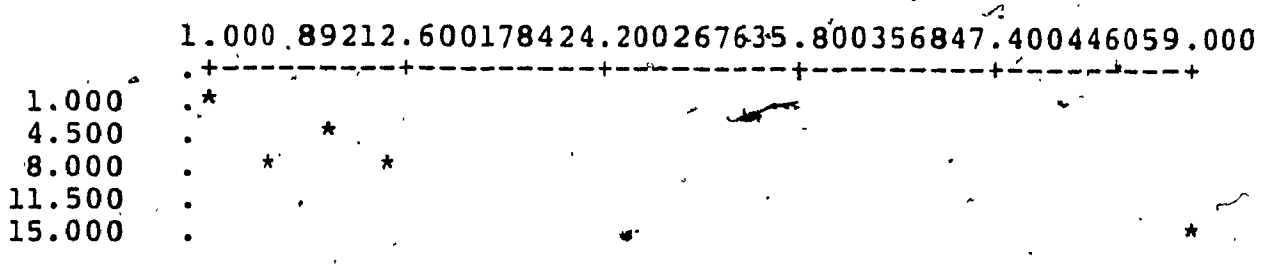
These results illustrate the power and potential of the simple metrics provided by PROFILE in analyzing the statistics and particular behavior of the code. An analysis of these effects is well beyond the scope of this report.

FIG. 5.4 BASE - NUMBER OF ALTERNATIVES
CROSS VARIABLE - EXECUTION FREQUENCY

(A) PROFILE



(B) GROUPCOUNT



(C) PASCAL COMPILER

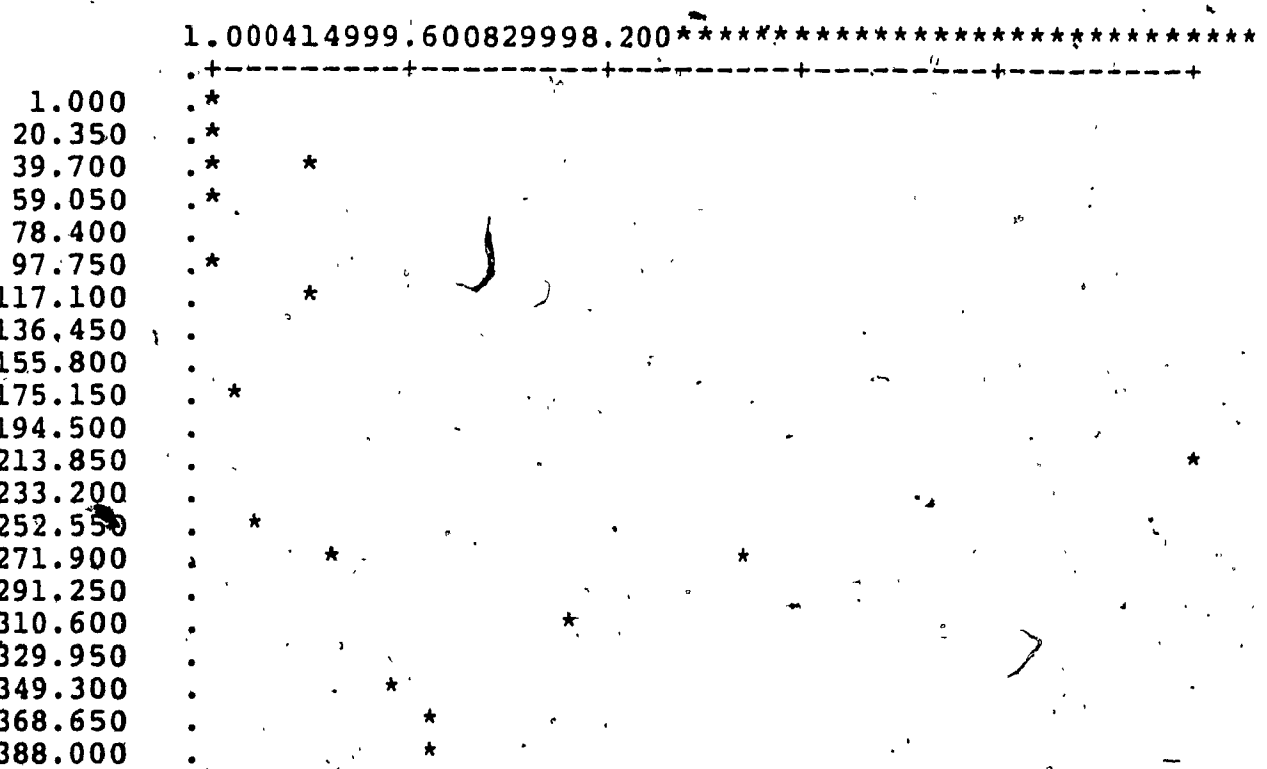


FIG. 5.5 CORRELATION MATRICES AND ELEMENTARY STATISTICS

(A) PROFILE CORRELATION MATRIX

ROW 1
1.000 -.098 -.318 -.233

ROW 2
-.098 1.000 .880 .751

ROW 3
-.318 .880 1.000 .637

ROW 4
-.233 .751 .637 1.000

VARIABLE	MEAN	STD.DEV.	STD.ERROR	MAXIMUM	MINIMUM	RANGE
1	5.500	3.028	.957	10.000	1.000	9.000
2	18.700	19.698	6.229	55.000	2.000	53.000
3	68.600	67.735	21.420	189.000	5.000	184.000
4	13938.900	16658.154	5267.771	49963.000	2.000	49961.000

(B) GROUP COUNT CORRELATION MATRIX

ROW 1
1.000 .240 -.347 .147

ROW 2
.240 1.000 .768 .959

ROW 3
-.347 .768 1.000 .694

ROW 4
.147 .959 .694 1.000

VARIABLE	MEAN	STD.DEV.	STD.ERROR	MAXIMUM	MINIMUM	RANGE
1	3.000	1.581	.707	5.000	1.000	4.000
2	6.400	5.273	2.358	15.000	1.000	14.000
3	17.600	8.204	3.669	28.000	6.000	22.000
4	123235.400	183049.606	81862.272	446059.000	1.000	446058.000

(C) PASCAL COMPILER CORRELATION MATRIX

ROW 1
1.000 -.490 -.546 -.531

ROW 2
-.490 1.000 .992 .513

ROW 3
-.546 .992 1.000 .560

ROW 4
-.531 .513 .560 1.000

VARIABLE	MEAN	STD.DEV.	STD.ERROR	MAXIMUM	MINIMUM	RANGE
1	11.000	6.205	1.354	21.000	1.000	20.000
2	136.762	136.820	29.857	388.000	1.000	387.000
3	475.048	461.803	100.774	1305.000	1.000	1304.000
4	298606.571	507278.514	110697.247	2074994.000	1.000	2074993.000

FIG. 5.6 RAW DATA

LOGIC LEVEL	NUMBER OF ALTERNATIVES	NUMBER OF STATEMENTS	EXECUTION FREQUENCY
(A) PROFILE			
1	2	5	2
2	7	112	7697
3	14	39	27788
4	30	132	26612
5	55	189	17815
6	48	134	49963
7	22	55	9331
8	3	8	57
9	3	6	120
10	3	6	4
(B) GROUPCOUNT			
1	1	14	1
2	5	21	29555
3	15	28	446059
4	7	19	81292
5	4	6	59270
(C) PASCAL COMPILER			
1	1	12	1
2	32	204	235476
3	114	509	224924
4	202	747	2074994
5	257	923	1151241
6	293	1143	765664
7	388	1305	468842
8	367	1189	469653
9	330	1024	394879
10	257	871	287328
11	237	775	120031
12	175	568	46765
13	94	310	18331
14	58	176	8160
15	27	71	1954
16	19	77	905
17	13	36	626
18	2	6	850
19	1	1	38
20	2	21	38
21	3	8	38

5.2 CONCLUSIONS

The software profiles presented in this report provide the user with reformatting capabilities, execution statistics and control flow information on Pascal programs. It tab sets and reformats the program in accordance to conventional fashion [Grog78]. The branch structure of the code is given by the statement's logic level which is overlayed on the reformatted source code. Decision to decision point path statistics recorded during the code's execution are used to demarcate the code's logic level at transition points. The actual frequency statistics are overlayed on the reformatted source code. This frequency data is attached to the logic level of the code.

The data provided by the source code profile are valuable in the testing of syntactically correct Pascal programs which contain no run-time errors. For example, if the execution frequency of a branch is zero, then something may be wrong. [Rama77] That is, unless the branch handles error conditions, rare cases or defaults then it should be traversed in the execution of the program. Similarly, high frequency branch paths are those segments of the code that should be optimized. Finally, nested control flow statements can be re-ordered in the source code in a manner which assigns priority to the most frequently used branches or to clarity of the source code. Examples of source code

profiles of sample programs are to be found in the appendix of this report. A source code profile of the profile program itself is given along with the profiles of the Pascal compiler and a few data processing routines.

The profile information is very useful to the ABL/W4 user in that it provides a metric - the execution frequencies - of the alternatives, or arcs, in a program's control flow graph. Such information is useful in the testing, design, optimizing, operation and maintenance phases of the software life cycle. Frequency statistics may be used to establish criteria for the clustering or conversely, the node-splitting process of step-wise refinement. Optimization techniques would certainly pay attention to the details of the most frequently used alternative within a cluster and thus the most frequently occuring paths through the code observed at its execution.

Testing strategies which attempt to exercise at least all paths through a program benefit greatly from access to such statistics.

The profile's statistics themselves are obtained by carefully decomposing or stripping the code of the Pascal language's control flow constructs. This means that the analysis provided by the profile on standard Pascal code provides the control flow decomposition needed to

automatically convert Pascal code into guarded/command style. The profile also provides statistics on the frequency of execution of the alternatives found in the code's guarded/command style. The profile provides a piece of the software infrastructure needed to synthesize, evaluate and analyze the ABL/W4 version of standard source code. Such an infrastructure is essential to any expert system that would hope to develop testing and optimizing strategies for source code.

The methods used in this report could be extended to handle other languages and would ideally deal with the BNF of the language rather than its reserved words and their syntax.

Recently there has been significant progress in the development of an interesting prototype system for the production of tested source code. This system, which was developed at Concordia, is based on an "executable" specification of a Pascal program. This system automatically generates instrumented source code and appropriate profiles. An illustrated example of this system has been made available to me [Mort85], and is shown in Appendix E: From Executable Specification to Source Code.

BIBLIOGRAPHY

[Abra75] Abrahams, P., "Structured Programming Considered Harmful.", SIGPLAN Notices, Vol.10, No. 4, (April,1975) 13 - 24.

[Adri82] Adrion, W.R., Branstad,M.A. and Cherniavsky,J.C. "Validation, Verification and Testing of Computer Software.", ACM Computing Surveys, Vol. 14, No. 2 (June,1982) 159 - 192.

[Bagg78] Baggi, D.L. and Shooman, M.L. "An Automatic Driver for Pseudo-Exhaustive Software Testing.", Proceedings of the COMPCON (Spring,1978) 278 - 282.

[Barn80] Barnhardt, R.S., "Implementing Relational Data Bases.", Datamation, Vol. 26, No. 10 (October,1980) 161 - 172.

[Benn85] Bennet, C. and Landauer, R., "The Fundamental Physical Limits of Computation.", Scientific American (July,1985) 48 - 56.

[Berg79a] Bergland, G.D. and Gordon, R.D., "Software Design Strategies.", in Tutorial: Software Design Strategies, Glenn D. Bergland and Ronald D. Gordon (eds.), IEEE Computer Society Press (1979) 1 - 14.

[Berg79b] Bergland, G.D. "Structured Design Methodologies.", in Tutorial: Software Design Strategies, Glenn D. Bergland and Ronald D. Gordon (eds.), IEEE Computer Society Press (1979) 162 - 181.

[Boeh73] Boehm, B.W., "Software and Its Impact: A Quantitative Assessment.", Datamation, Bol. 19, No. 5 (May,1973) 5 - 24.

[Boeh79] Boehm, B.W., "Software Engineering.", in Tutorial: Software Design Strategies, Glenn D. Bergland and Ronald D. Gordon (eds.), IEEE Computer Society Press (1979) 225 - 240.

[Broo77] Brooks, R., "Towards a Theory of the Cognitive Processes in Computer Programming.", International Journal of Man - Machine Studies, Vol. 9 (1977) 737 - 751.

[Chap79] Chapin, N., "A Measure of Software Complexity.", AFIPS Conference Proceedings, Vol. 48 (1979) 995 - 1002.

[DeMi77] DeMillo, R.A., Lipton, R.J. and Perlis, A.J. "Social Processes and Proofs of Theorems and Programs.", Conference Record of the Fourth ACM Symposium on Principles of Programming Language (January, 1977) 206 - 214.

[Dijk72] Dijkstra, E., "The Humble Programmer.", Communications of the ACM, Vol. 15, No. 10 (October,1972) 859 - 866.

[Dijk75] Dijkstra, E.W., "Guarded Commands, Nondeterminacy and Formal Derivation of Programs.", Communications of the ACM, Vol. 18, No. 8 (August,1975) 453 - 457.

[Dijk79] Dijkstra, E.W., "On the Interplay Between Mathematics and Programming.", Lecture Notes in Computer Science, Vol. 69 Program Construction, Bauer, F.L. and Broy, M. (eds) (1979) 35 - 46

[Evan82] Evans, M., "Software Engineering for the Cobol Environment.", Communications of the ACM, Vol. 25, No. 12 (December,1982) 874 - 882.

[Fair78] Fairley, R., "Tutorial: Static Analysis and Dynamic Testing of Computer Software, IEEE Computer (April,1978) 14 - 23.

[Fico83] Ficocelli, L., "Problems to Programs: A Humanistic Approach (An Introduction to ABL Methodology)", Thesis, Department of Computer Science, Concordia University, Montreal, Quebec, Canada (1983).

[Fico85] Ficocelli, L., Jaworski, W.M., O'Mara, K.S., "The ABL/W4 Approach: A View of Representational Distortion, Documentation and Software Pragmatics.", submitted to INFOR (1985).

[Fox178] Foxley, E. and Morgan, D.J., "Monitoring the Run-time Activity of Algol 68-R Programs.", Software Practice and Experience, Vol. 8, (1978) 29 - 34.

[Good77] Goodenough, J.B. and Gerhart, S.L., "Toward a Theory of Testing: Data Selection Criteria.", in Current Trends in Programming Methodology, Vol. II, Yeh, R.T. (ed.), Prentice-Hall, New Jersey (1977) 44 - 78.

[Grie79] Gries, D., "Current Ideas in Programming Methodology.", Lecture Notes in Computer Science, Bauer, F.L. and Broy, M. (eds.), Vol. 69 Program Construction, Springer-Verlag (1979).

[Grif79] Griffiths, S.M., "Design Methodologies - A Comparison.", in Tutorial: Software Design Strategies, Glenn D. Bergland and Ronald D. Gordon (eds.), IEEE Computer Society Press (1979) 189 - 213.

[Grog78] Grogono, P., "Programming in Pascal", Addison-Wesley (1978).

[Hals73] Halstead, M. and Bayer, R., "Algorithm Dynamics.", Proceedings of the ACM (1973) 126 - 135.

[Hals77] Halstead, M., "Elements of Software Science.", Elsevier, New York (1977).

[Hend77] Henderson, P.B., "Structured Program Testing" in Current Trends in Programming Methodology, Vol. II, Yeh, R.T. (ed.), Prentice-Hall, New Jersey (1977) 1 - 15.

[Hint81] Hinterberger, H. and Jaworski, W.M., "Controlled Program Design by use of the ABL Programming Concept.", Angewandte Informatik (Applied Informatics), Weisbaden, Germany (July,1981) 302 - 310.

[Hoar69] Hoare, C.A.R., "An Axiomatic Basis for Computer Programming.", Communications of Computer Science, Vol. 12, No. 10 (October,1969) 576 - 583.

[Holt77] Holton, J.B. "Are the New Programming Techniques Being Used.", Datamation, Vol. 23, No. 7 (July,1977) 97 - 103.

[Howd82] Howden, W.E., "Contempory Software Developments.", Communications of the ACM, Vol. 25, No. 5 (May,1982) 318 - 329.

[Huan77] Huang, J.C., "Error Detection Through Program Testing.", in Current trends in Programming Methodology, Vol. II, Yeh, R.T. (ed.) Prentice-Hall, New Jersey (1977) 112 - 150.

[Jawo82] Jaworski, W.M. and Williams, A.J., "Representation of Therapeutics Strategies in Dermatology.", Canadian Dermotological Association Conference, Edmonton (July,1982).

[Jawo84] Jaworski, W.M., Ficocelli, L., O'Mara, K.S., "ABL/W4: System for Software Modelling.", Concordia University, submitted to Systems Research (1984).

[Knut71] Knuth, D., "An Empirical Study of Fortran Programs.", Software - Practice and Experience, Vol. 1 (January,1971) 105 - 133.

[Knut74] Knuth, D.E., "Computer Programming as an Art.", Communications of the ACM, Vol. 17, No. 12 (December,1974) 667 - 673.

[Knut79] Knuth, D.E., "Structured Programming with GOTO Statements.", in Classics in Software Engineering, Yourdon, E.N. (ed.), New York (1979).

[Kuce67] Kucera, H. and Francis, W.N., "Computational Analysis of Present Day American English, Brown University Press, Providence, R.I. (1967).

[Leve82] Levene, A.A. and Mullery, G.P., "An Investigation of Requirement Specification Languages: Theory and Practice.", Computer, Vol. 15, No. 5 (May,1982) 50 - 59.

[Maye81] Mayer, R. E. and Bayman, P., "Psychology of Calculator Languages: A Framework for Describing Differences in Users' Knowledge.", Communications of the ACM, Vol. 24, No. 8, (August, 1981) 511 - 520.

[McCa76] McCabe, T.J., "A Complexity Measure.", IEEE Transactions on Software Engineering, Vol. SE-2, No. 4 (December,1979) 308 - 320.

[Mont74] Montalbano, Michael, "Decision Tables.", Science Research Associates Inc. (1974).

[Mort85] Morton, David, private communication, Department of Computer Science, Concordia University (1985).

[Myer76] Myers, G., "Software Reliability: Principles and Practices.", John Wiley, New York (1976).

[Myer79] Myers, G., "The Art of Software Testing.", John Wiley, New York (1979).

[Newe72] Newell, A. and Simon, H.A., "Human Problem Solving.", Prentice-Hall, New Jersey (1972).

[Orr 79] Orr, K.T., "Introducing Structured Systems Design.", in Tutorial: Software Design Strategies, Glenn D. Bergland and Ronald D. Gordon (eds.), Ieee Computer Society Press (1979) 72 - 82.

[Paig74a] Paige, M.R. and Benson, J.P., "The Use of Software Probes in Testing Fortran Programs.", Computer, Vol. 7, No. 7 (July,1974) 40 - 47.

[Paig74b] Paige, M.R., "Software Testing: An Overview.", IEEE, Fourth Annual International Symposium on Fault-Tolerant Computing (1974) 5-18 - 5-21.

[Paig75] Paige, M.R., "Program Graphs, an Algebra, and Their Implication for Programming.", Ieee Transactions on Software Engineering Vol. SE-1, No. 3 (September,1975) 286 - 291.

[Pete81] Peters, L.J., "Software Design: Methods and Techniques.", Yourdon Press, NewYork (1981).

[Prel84] Prell, E.M. and Sheng, A.P., "Building Quality and Productivity into a Large Software System", IEEE Software (July,1984).

[Prob80] Probert, R., "On the Optimal Placement of Software Monitors.", Technical Report, University of Ottawa, Ottawa, Ontario, Canada, TR80.05.

[Rama77] Ramamoorthy, C.V. and Ho, S.F., "Testing Large Software with Automated Software Evaluation Systems.", in Current Trends in Programming Methodology, Vol. II, Yeh, R.T. (ed.), Prentice-Hall, New Jersey (1977) 112 - 150.

[Sage85] Sager, N., "Natural Language Processing: A Computer Grammar of English and its Applications, Addison-Wesley (1985).

[Satt72] Satterthwaite, E., "Debugging Tools For High Level Languages.", Software Practice and Experience, Vol. 2 (1972) 197 - 217.

[Sche63] Scheerer, M., "Problem Solving.", Scientific American, Vol. 204, No. 4 (April,1963) 118 - 128.

[Schn79] Schneidewind, N.F., "Software Metrics for Aiding Program Development and Debugging.", AFIPS Conference Proceedings, Vol. 48 (1979) 989 - 994.

[Shaw80] Shaw, M., "The Impact of Abstraction Concerns on Modern Programming Languages.", Proceedings of the IEEE, Vol. 68, No. 9 (September,1980) 1119 - 1130.

[Shei81] Sheil, B.A., "The Psychological Study of Programming.", ACM Computing Surveys, Vol. 13, No. 1 (March,1981) 101 - 120.

[Shne75] Shneiderman, B., "Cognitive Psychology and Programming Language Design.", SIGPLAN Notices Vol.10, No. 7 (July,1975) 46 - 47.

[Shne77] Shneiderman, B., "Measuring Computer Program Quality and Comprehension.", International Journal of Man - Machine Studies, Vol. 9 (1977) 465 - 478.

[Snee84] Sneed, H., "Software Renewal: A Case Study.", IEEE Software (July,1984) 56 - 63.

[Sowa85] Sowa, J., "Conceptual Structures: Information Processing in Mind and Machine.", Addison-Wesley (1985).

[Stuc77] Stucki, L., "New Directions in Automated Tools For Improving Software Quality.", in Current Trends in Programming Methodology, Vol. II, Yeh, R.T. (ed.), Prentice-Hall, New Jersey (1977) 80 - 111.

[Turn80] Turner, J., "The Structure of Modular Programs.", Communications of the ACM, Vol. 23, No. 5 (May,1980) 272 - 277.

[Wino85] Winograd, T., "Language as a Cognitive Process Vol 1: Syntax.", Addison-Wesley (1985).

[Wass78] Wasserman, A.I. and Freeman, P., "Software Engineering Education: Status and Prospects.", Proceedings of the IEEE, Vol. 66, No. 8 (August,1978) 445 - 451.

[Wass80] Wasserman, A.I., "A Strategy for Improving Software Development Practices.", in Tutorial on Software Design Techniques, Peter Freeman and Anthony Wasserman (eds.), IEEE Computer Society Press 3rd edition (1980) 440 - 444.

[Wein71] Weinberg, G.M., "The Psychology of Computer Programming."", Van Nostrand Reinhold Company, New York (1971).

[Wino77] Winograd, T., "Beyond Programming Languages.", Communications of the ACM, Vol. 22, No.7 (July,1977) 391 - 401.

[Yeh 77] Yeh, R.T., (ed.), "Current Trends in Programming Methodology, Vol. II.", Prentice-Hall, New Jersey (1977).

[Yode78] Yoder, C.M. and Schrag, M.L., "Nassi-Shneiderman Charts an Alternative to Flowcharts for Design.", Proceedings , ACM SIGSOFT/SIGMETRICS Software and Assurance Workshop (November,1978).

[Zweb79] Zweben, S.H. and Halstead, M.H., "The Frequency Distribution of Operators in PL/I Programs.", IEEE Transactions on Software Engineering, Vol. SE-5, No. 2 " (March,1979) 91 - 95.

A.1 HOW TO USE PROFILE

A.1 HOW TO USE THE PROFILE

PROFILE takes a Pascal program as input, and outputs an edited program which maintains frequency counts of statement execution. Profile then has this modified program compiled and executed employing the user supplied input and output files. An indented listing of the program showing the number of times each statement was executed and the logic level of each statement is written on a file specified by the user. This listing file is rewound before and after execution.

The time required for the program execution will be increased by a factor of approximately 1.5.

XXXFILE, XXXCOMM, XXXW, XXXAAA, XXXRR are local files used by PROFILE They are returned before the end of the procedure.

To obtain an execution profile of a Pascal program, the following commands must be used:

```
GET,PROCS/UN=KFFFI82
-PROFILE,PROCS,S,$F$,R
```

where:

S is the name of a local file containing the Pascal source code program (not a compilation listing). It is rewound before being read. It must have no

compilation errors. The default value of S is Prog.

F stands for the files used in the program. All input files must be local. The file names are seperated by commas.

R is the name of a local file to which profile will write the program listing which has been expanded to include execution counts. It is rewound before and after execution. The default value of R is PRESULT.

EXAMPLE

Assume that the source program is on a file called SAMPLE.

Assume that the program statement of SAMPLE is the following:

```
PROGRAM SAMPLE TEST (INPUT,OUTPUT,OLDFILE,NEWFILE)
```

To run this program the user might use the following command:

```
LGO,RA1,,F22,G7.
```

Assume that the user wants the output on MYPRFIL.

He would use the following commands :

```
GET,PROCS/UN=KFFFII82
-PROFILE,PROCS,SAMPLE,$RA1,,F22,G7$,MYPRFIL
```

RESTRICTION:

The period which ends the Pascal program must be on the same line as the last END.

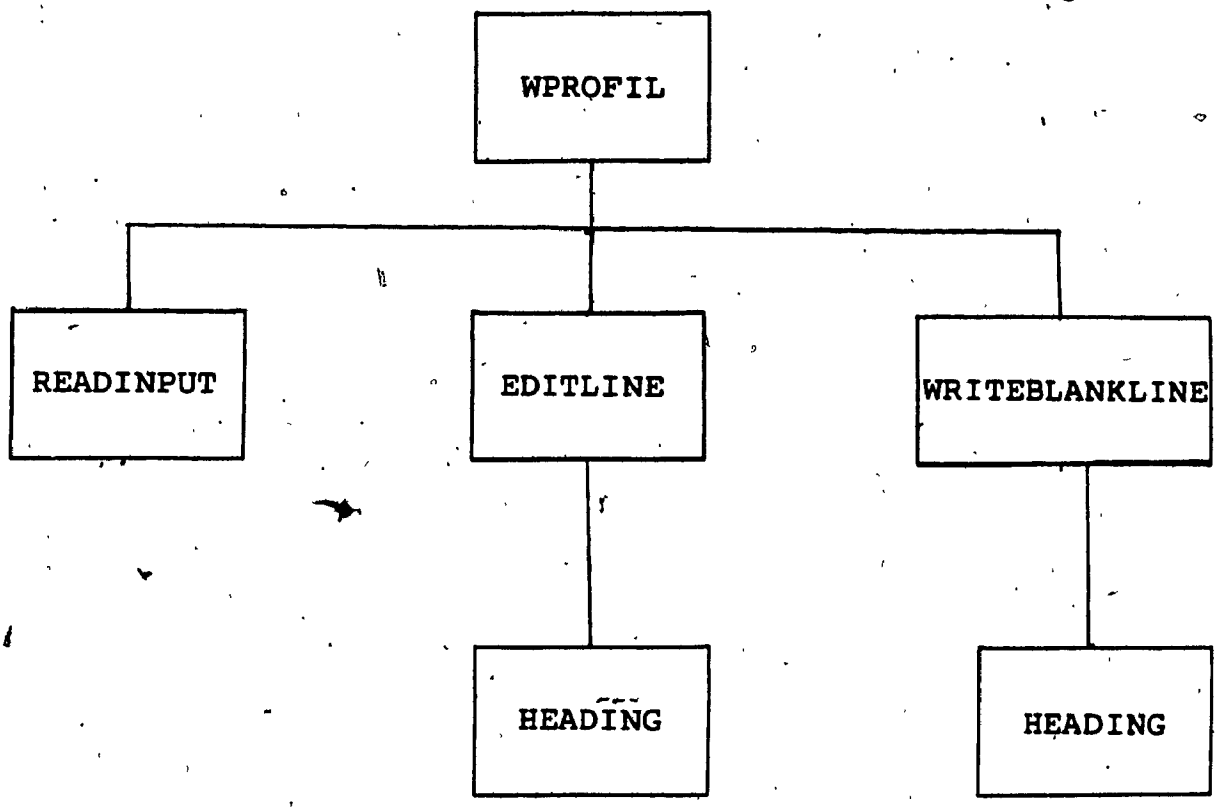
A.2 PROCEDURE FILE FOR PROFILE

A.2 PROCEDURE FILE FOR PROFILING A PROGRAM

```
.PROC,PROFILE,SOURCE,FILES,PRESULT=PRESULT.
GET,BBMAIN,BBPROC/UN=KFFFI82.
BBMAIN,SOURCE,XXXFILE,XXXW,XXXAAA,XXXCOMM.
REWIND,LGO,XXXFILE.
PASCAL,XXXFILE,XXXW.
SKIP,NOERRORS.
EXIT.  *** COMPILE ERRORS ***
REVERT,ABORT.  *** COMPILE ERRORS ***
ENDIF,NOERRORS.
COPY,BBPROC,LGO.
LGO,FILES.
SKIP,SUCCESS.
EXIT.
REVERT,ABORT.  *** RUNTIME ERRORS ***
ENDIF,SUCCESS.
REWIND,XXXRR,PRESULT.
COPY,XXXRR,PRESULT.
REWIND,PRESULT.
RETURN,XXXFILE,XXXAAA,XXXCOMM,XXXW,XXXRR.
REVERT.
```

B.1 WPROFIL: A HIERARCHIAL VIEW

B.2 WPROFIL: A HIERARCHIAL VIEW



WPROFIL

READINPUT

EDITLINE

HEADING

WRITEBLANKLINE

HEADING

B.2 WPROFIL: ABL/W4 REPRESENTATION

B.2.1 WPROFIL ABL/W4 ABSTRACT PROGRAM

```
STRATEGY

S1    X                                                    initialize
S2       X  X  X  X                                        line edit + overlay

P1       T  F  F  F     EOF (XXXCOMM)                      EOF on command file
P2          T  T  F     LinesIn > 0                        source code rather than control line
P3          T  F        CtrNo <> OldCtrNo ∧ CtrNo <> 0     change of state - not at root level
P4             T        Lev = 99                           indicates start of procedure or function call

A1    1                initialize files and lines, pageno, and ctrno    initialize files and set indices to zero
A2          1  1  1     READINPUT( ----- )                 reads command line and source statement
A3          2           OldCtrNo = CtrNo                   change of state
A4          3           CtrChange = TRUE                   records a new branch or decision point
A5          4  2        EDITLINE( ----- )                  edits source code line
A6                2     WRITEBLANKLINE(Pageno,Lines)       inserts two blank lines at the start of a subprogram

NEXT  2  0  2  2  2     next state vector

ALT   1  2  3  4  5     alternative numbers
```

Procedure Calls

ReadInput (LinesIn, LinesOut, Lev, PrLev, CtrNo, Line, LineLength)

EditLine (PageNo, Lines, LineesIn, LinesOut, Lev, Prlev, CtrNo, Line, LineLength, CtrChange)

WriteBlankLine(PageNo,Lines)

FILES

XXXCOMM Command file which "drives" the editor. Contains original line number profile line number, logic level, print level and the subscript (ctrno) which points to the correct value in the counter array.

XXXFILE Edited source code plus added lines

XXXRR Output file - contains the execution profile of the program.

STRATEGY NUMBER
!
!
! ALTERNATIVE NUMBER
! !
! !
S1 A1 initializes: number of lines, page number and old counter number to 0.
        Resets source code file and command file, rewrites output files. ==> 2

S2 A2 Detects end of the command file which drives the edit strategy ==> 0

S2 A3 Detects a branch or decision point in the source code and edits the
        output file with the counter number accordingly, iterates to S2. ==> 2

S2 A4 Finds the source code statement to be at the same logical level (i.e.
        within an alternative) as the previous statement, hence edits the
        output source code accordingly, iterates to S2. ==> 2

S2 A5 Detects the start of a procedure or function call and edits the output
        source code by inserting 2 blank lines, to enhance the format of the
        source code and iterates to S2. ==> 2

VARIABLES

CtrNo	Counter Number
CtrChange	Counter Change (i.e. new counter) - BOOLEAN
Lines	Line index on output page
LinesOut	Profile line number
LinesIn	Source code line number
LineLength	Number of characters in source line
Line	Array used to hold source line
Lev	Branch or logic level, equivalent to an alternative
PrLev	Print level (for output format)
PageNo	Page number (for output format)
OldCtrNo	Old counter number (previous branch or counter number)
Lev = 99	Level label - used to indicate start of sub-program call

B.2.2 EDITLINE ABL/W4 ABSTRACT PROGRAM

```
PROCEDURE EDITLINE (PageNo, Lines, LinesIn, LinesOut, Lev, PrLev, CtrNo, Line, LineLength, CtrChange)

                    STRATEGY

S1   X X                                                                    output for new page
S2       X                                                                  output control information
S3         X X X                                                            change of state or level zero
S4               X                                                          tab for source code; initialize m
S5                 X X X                                                    output non wrap around line
S6                       X X                                                output first part of wrap around
S7                             X X                                          format for heading
S8                                   X                                      set controls for wrap around line
S9                                     X X                                  output second part of wrap around

P1   T F                               T F       Lines mod PageSize = 0     check for new page
P2         T F F                                 CtrChange = TRUE           check for change of state
P3           T F                                 CtrNo <> 0                 check for root level of source
P4                   T T F                       Margin + LineLength <= 132 will source statement fit on line
P5                   T F                   T F   I < LineLength             check if end of statement
P6                             T F               I < Extra                  check if end of initial portion

A1   1                                 1         HEADING(PageCtr)           output profile header
A2       2                                       WRITE(LInesin:6,LinesOut:6,Lev:4,' ':(Lev-1)*Indent   write line statistics
A3         1                                     WRITE(XXXKounter[CtrNo]:8) write the branch or path frequenc
A4         2                                     CtrChange = FALSE          reset change of state flag
A5           1                                   WRITE(')':8)               write ')' to indicate no change o
A6             1                                 WRITE(' ':8)               write spaces to indicate root lev
A7               1                               WRITE(' ':2+(PrLev-Lev)*Indent  write tab for source statement
A8               2                               Margin = 26 + Indent*(PrLev-1) compute start of source statement
A9               3                               I = 1                      initialize counter
A10                    2     2           2       I = I + 1                  increment counter
A11                    1     1           1       WRITE(Line[I])             write character of source stateme
A12                      1       1           1   write line                 write new line
A13                        1                     Extra = 132 - Margin       available print positions
A14        1                             1       Lines = Lines + 1          increment line count
A15                                      2       WRITE(' ':Margin)tab for second part of line
A16                                      3       I = Extra + 1               initialize counter to remaning c

NEXT 2 2 3 4 4 4 5 5 0 6 6 7 8 8 9 9 0  next state vector

ALT  1 2 3 4 5 6 7 8 9 1 1 1 1 1 1 1 1  alternatives
                      0 1 2 3 4 5 6 7

Ident - global constant, value 3 PageSize - global constant, vlue 54
XXXKounter - global array of frequency counts - passed from host program

LOCAL VARIABLES Margin, Extra, I
```

STRATEGY NUMBER

ALTERNATIVE NUMBER

S1	A1	Output page header.	==> S2
S1	A2		==> S2
S2	A3	Increment line counter - output original line number and logic level of source code	==> S3
S3	A4	Change of state - write value of counter and set change of state flag to false	==> S4
S3	A5	No change of state, write ')' to indicate this.	==> S4
S3	A6	Root level of source code - write spaces	==> S4
S4	A7	Write tab for source code, initialize margin to reflect print positions used	==> S5
S5	A8	Write a character from the source line array and increment the pointer	==> S5
S5	A9	Write a line on output file XXXRR	==> 0
S5	A10	Comput positions left on print line - Extra	==> S6
S6	A11	Write a character from the source line array and increment the pointer	==> S6
S6	A12	Write a line on output file XXXRR	==> S7
S7	A13	Output page header	==> S8
S7	A14		==> S8
S8	A15	Increment lines, output tab for second part of wrap around line and initialize I to point to first character remaining to be printed	==> S9
S9	A16	Write a character from the source line array and increment the pointer	==> S9
S9	A17	Write a line on output file XXXRR	==> 0

B.2.3 READINPUT ABL/W4 ABSTRACT PROGRAM

PROCEDURE READINPUT (LINESIN, LINESOUT, LEV, PRLEV, LINE, LINELENGTH)

STRATEGY

```
S1   X                                  Initialize
S2     X X                              Read a line character by character
S3       X                              Read next line

P1     F T   EOLN(XXXFILE)               Test end of line on edited source file

A1     1     LineLength = 0              Initialize index to source line
A2     2     READLN(XXXCOMM,-----)       Read a line from the command file
A3       1   LineLength = LineLength + 1 Increment index to source line array
A4       2   READ (XXXRR, Line[LineLength]) Read a character into line indexed by line length
A5         1 READLN(XXXFILE)             Read past end of line marker

NEXT 2 2 3 0  Next strategy vector

ALT  1 2 3 4  Alternatives
```

PROCEDURE CALLS

READLN (XXXCOMM, LinesIn, LinesOut, Lev, PrLev, CtrNo)
Reads control information from the command file about the corresponding line on the edited source file.

READ (XXXFILE, Lines [LineLength])
Reads a character from the edited source code file into the array line at the position specified by linelength.

READLN (XXXFILE)
Reads past end of line mark on edited source file.

STRATEGY NUMBER

ALTERNATIVE NUMBER

S1	A1	Initializes the index to zero and reads a line from the command file (XXXCOMM)	==> S2
S2	A2	Iteratively strips the text from the source code line character by character until end of line is encountered.	==> S2
	A3	When end of line is encountered on the edited source file, control is transfered to S3	==> S3
S3	A4	Reads past end of line mark on edited source code file.	==> 0

8.2.4 WRITEBLANKLINE 'ABL/W4 ABSTRACT PROGRAM
PROCEDURE WRITEBLANKLINE (PageNo, Lines)

```
STRATEGY
S1   X X X                                  Insert blank lines to highlight procedures and functions

P1   F T T  Lines mod Pagesize <> 0         NOT End of page
P2     T F  (Lines + 1) mod PageSize = 0    One line short of end of page

A1     1    Lines = Lines + 1               Increment line counter by 1
A2       1  Lines = Lines + 2               Increment line counter by 2
A4       2  WRITELN(XXXRR),WRITELN(XXXRR)   Write two blank lines to input file

NEXT 0 0 0  Next state vector

ALT  1 2 3  Alternatives
```

B.3 WPROFIL: SOURCE LISTING

```
000004     1 (*$L50*)
000004     2 PROGRAM DUMMY(XXXFILE,XXXRR,XXXCOMM);
000004     3 (*$E+*)
000004     4
000004     5 TYPE
000004     6   XXXTYP = ARRAY [INTEGER] OF INTEGER;
000004     7
000004     8 VAR
000004     9   XXXRR : TEXT;       (* PROFILE *)
000040    10   XXXFILE : TEXT;     (* EDITED PROGRAM *)
000074    11   XXXCOMM : TEXT;     (* ORIGINAL LINE NUMBER, PROFILE LINE NUMBER,
000130    12                          LEVEL NUMBER, PRINT LEVEL NUMBER, COUNTER NUMBER *)
000130    13
000130    14 PROCEDURE WPROFIL (XXXKOUNTER : DYNAMIC XXXTYP;      (* ARRAY OF COUNTERS *)
000003    15                    XXXMAXCTR : INTEGER;              (* NUMBER OF COUNTERS *)
000004    16                    XXXPROGID : ALFA;                 (* PROGRAM NAME *)
000005    17                    VAR XXXFILE,
000006    18                        XXXRR,
000007    19                        XXXCOMM : TEXT);
000014    20
000014    21 CONST
000014    22   INDENT = 2;
000014    23   PAGESIZE = 48;
000014    24
000014    25 TYPE
000014    26   LINETYPE = ARRAY [ 1 .. 120 ] OF CHAR;
000014    27
000014    28 VAR
000014    29   LINE : LINETYPE;
000204    30   LINES,      (* LINE COUNTER FOR REPORT *)
000204    31
000204    32 (* CONTROL VALUES FOR EACH LINE IN EDITED PROGRAM *)
000204    33   LINESIN,    (* LINE NUMBER OF PROGRAM *)
000204    34   LINESOUT,   (* LINE NUMBER ON PROFILE *)
000204    35   CTRNO,      (* COUNTER NUMBER *)
000204    36   LEV,        (* LOGICAL LEVEL OF LINE *)
000204    37   PRLEV : INTEGER;  (* PRINT LEVEL FOR LINE *)
000212    38
000212    39   OLDCTRNO : INTEGER;     (* USED TO KEEP THE OLD COUNTER NUMBER *)
000213    40   CH : CHAR;
000214    41   LINELENGTH : INTEGER;     (* LENGTH OF SOURCE PROGRAM LINE *)
000215    42   CTRCHANGE : BOOLEAN;     (* USED TO INDICATE CHANGE OF COUNTER NUMBER *)
000216    43   PAGENO : INTEGER;
000217    44
000217    45 PROCEDURE XXXHEADING (VAR PAGENO : INTEGER);
000003    46
000003    47 BEGIN
```

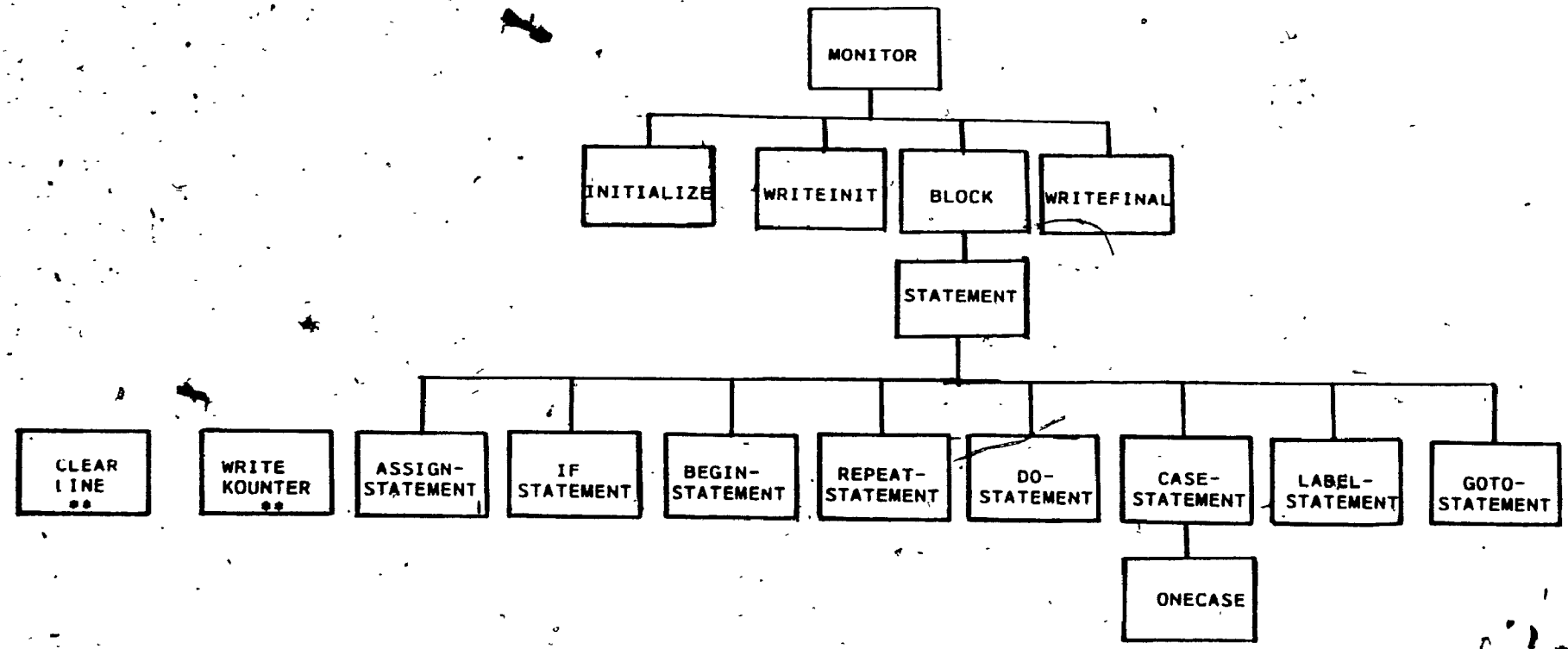
```
000003    48   PAGENO := PAGENO + 1;
000006    49   WRITELN (XXXRR,'1 * * *  EXECUTION  PROFILE  OF  ',XXXPROGID,
000022    50            ' * * *',' ':48,'PAGE',PAGENO:6);
000047    51   WRITELN(XXXRR);
000053    52   WRITELN(XXXRR,'  USER PROFILE  LEV NO OF    ',
000062    53           '*  *  *  PROGRAM  ',
000070    54           'STATEMENTS  *     *');
000100    55   WRITELN(XXXRR,'  LINE  LINE        TIMES');
000111    56   WRITELN(XXXRR,'  NO.   NO.       EXECUTED');
000122    57   WRITELN(XXXRR)
000125    58 END;
000155    59
000155    60 PROCEDURE XXXWRITEBLANKLINE(VAR PAGENO, LINES : INTEGER);
000004    61
000004    62 BEGIN
000004    63   IF LINES MOD PAGESIZE = 0
000013    64     THEN
000014    65       XXXHEADING(PAGENO)
000014    66     ELSE
000016    67       IF (LINES + 1) MOD PAGESIZE = 0
000024    68         THEN
000025    69           BEGIN
000025    70             LINES := LINES + 1;
000026    71           END
000026    72         ELSE
000027    73           BEGIN
000027    74             LINES := LINES + 2;
000031    75             WRITELN(XXXRR);
000034    76             WRITELN(XXXRR)
000037    77           END;
000040    78 END;
000046    79
000046    80 PROCEDURE XXXEDITLINE ( VAR PAGENO, LINES :  INTEGER;
000004    81                        LINESIN, LINESOUT, LEV, PRLEV, CTRNO : INTEGER;
000011    82                        LINE : LINETYPE;
000012    83                        LINELENGTH : INTEGER;
000013    84                        VAR CTRCHANGE : BOOLEAN);
000204    85
000204    86
000204    87 VAR
000204    88   MARGIN : INTEGER;
000205    89   I, EXTRA : INTEGER;
000207    90
000207    91 BEGIN
000207    92   IF LINES MOD PAGESIZE = 0
000017    93     THEN
000020    94       XXXHEADING(PAGENO);
```

```
000021     95    LINES := LINES + 1;
000023     96    WRITE (XXXRR,LINESIN : 6,LINESOUT: 6, LEV: 4);
000041     97    IF LEV > 1 THEN
000043     98      WRITE(XXXRR,' ':(LEV-1)*INDENT);
000051     99    IF CTRCHANGE
000051    100      THEN
000053    101        BEGIN
000053    102          WRITE (XXXRR,XXXKOUNTER[CTRNO]:8);
000064    103          CTRCHANGE := FALSE
000065    104        END
000066    105      ELSE
000067    106        IF CTRNO <> 0
000067    107          THEN
000071    108            WRITE (XXXRR,')':8)
000071    109          ELSE
000100    110            WRITE (XXXRR,' ':8);
000106    111    WRITE (XXXRR,' ': 2 + ((PRLEV-LEV) * INDENT));
000115    112    MARGIN := 26 + INDENT * (PRLEV - 1);
000121    113    IF MARGIN + LINELENGTH <= 132
000122    114      THEN
000124    115        FOR I := 1 TO LINELENGTH DO
000125    116          WRITE (XXXRR,LINE[I])
000142    117      ELSE
000146    118        BEGIN
000146    119          EXTRA := 132 - MARGIN;
000147    120          FOR I := 1 TO EXTRA DO
000151    121            WRITE (XXXRR, LINE[I]);
000172    122          WRITELN(XXXRR);
000175    123          IF LINES MOD PAGESIZE = 0
000204    124            THEN
000205    125              XXXHEADING (PAGENO);
000207    126          LINES := LINES + 1;
000211    127          WRITE (XXXRR,' ':MARGIN);
000217    128          FOR I := EXTRA + 1 TO LINELENGTH DO
000222    129            WRITE (XXXRR,LINE[I]);
000243    130        END;
000243    131    WRITELN(XXXRR);
000247    132  END;
000301    133
000301    134  PROCEDURE XXXREADINPUT (VAR LINESIN,LINESOUT,LEV,PRLEV,CTRNO : INTEGER;
000007    135                          VAR LINE : LINETYPE;
000010    136                          VAR LINELENGTH : INTEGER);
000011    137
000011    138  BEGIN
000011    139    LINELENGTH := 0;
000006    140    READLN (XXXCOMM,LINESIN,LINESOUT,LEV,PRLEV,CTRNO);
000027    141    WHILE NOT EOLN (XXXFILE) DO
```

```
000032   142      BEGIN
000032   143        LINELENGTH := LINELENGTH + 1;
000034   144        READ (XXXFILE, LINE[LINELENGTH])
000045   145      END;
000046   146    READLN(XXXFILE);
000051   147  END;
000067   148
000067   149  BEGIN  (* MAIN *)
000067   150    RESET (XXXFILE);
000020   151    RESET (XXXCOMM);
000022   152    REWRITE (XXXRR);
000024   153    LINES := 0;
000026   154    PAGENO := 0;
000027   155    OLDCTRNO := 0;
000030   156    CTRCHANGE := FALSE;
000031   157    WHILE NOT EOF (XXXCOMM) DO
000033   158      BEGIN
000033   159        XXXREADINPUT (LINESIN,LINESOUT,LEV,PRLEV,CTRNO,LINE,LINELENGTH);
000041   160        IF LINESIN > 0
000041   161          THEN
000043   162            BEGIN
000043   163              IF (CTRNO <> OLDCTRNO) AND (CTRNO <> 0)
000045   164                THEN
000047   165                  BEGIN
000047   166                    OLDCTRNO := CTRNO;
000050   167                    CTRCHANGE := TRUE;
000052   168                  END;
000052   169              XXXEDITLINE(PAGENO,LINES,LINESIN,LINESOUT,LEV,PRLEV,CTRNO,
000057   170                          LINE,LINELENGTH,CTRCHANGE);
000063   171            END
000063   172          ELSE
000064   173            IF LEV = 99
000064   174              THEN
000067   175                XXXWRITEBLANKLINE(PAGENO,LINES)
000070   176        END;
000072   177  END;
000132   178
000132   179  BEGIN
000132   180  END.

          COMPILER-ESTIMATED 'W' OPTION = 0042068.
```

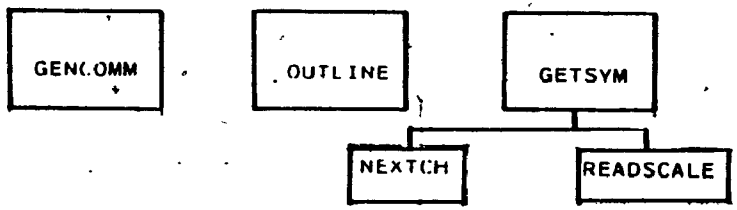
C.1 MONITOR: A HIERARCHIAL VIEW

C.2 PROGRAM MONITOR - A HIERARCHIAL VIEW



** ----- CALLED BY ALL PROCEDURES ON THE SAME LEVEL

UTILITY PROCEDURES CALLED BY ALL PROCEDURES ABOVE



C.2 MONITOR: ABL/W4 REPRESENTATION

C.2.0 MONITOR: ABL/W4 REPRESENTATION

C.2.1 Monitor: A Structural View

C.2.2 Initialize ABL/W4 Abstract Program

C.2.3 GenComm ABL/W4 Abstract Program

C.2.4 Outline ABL/W4 Abstract Program

C.2.5 GetSym ABL/W4 Abstract Program

C.2.51 NextCh ABL/W4 Abstract Program

C.2.52 ReadScale ABL/W4 Abstract Program

C.2.6 WriteInit ABL/W4 Abstract Program

C.2.7 WriteFinal ABL/W4 Abstract Program

C.2.8 Block ABL/W4 Abstract Program

C.2.81 Statement ABL/W4 Abstract Program

C.2.811 CLEARLINE ABL/W4 ABSTRACT PROGRAM

C.2.812 WRITEKOUNTER ABL/W4 ABSTRACT PROGRAM

C.2.82 ABL/W4 Representation of Symbol Master Machine

C.2.821 IfStatement Abstract Program

C.2.822 BeginStatement Abstract Program

C.2.823 RepeatStatement Abstract Program

C.2.824 DoStatement Abstract Program

C.2.825 CaseStatement Abstract Program

C.2.8251 OneCase Abstract Program

C.2.826 LabelStatement Abstract Program

C.2.827 GoToStatement Abstract Program

C.2.83 CHECKPAREN

C.2.9 Main ABL/W4 Abstract Program

C.2.0 MONITOR: ABL/W4 REPRESENTATION

```
                                PROGRAM MONITOR
                        (PROG, XXXFILE, OUTPUT, XXXAAA, XXXCOMM)


DECLARATIONS

INITIALIZE (StatBegSym, ConstBegSym, Key, KSym, LL, CC, LinesIn, LinesOut,
                        CtrNo, SavCtr, EndOfLin)

GENCOMM (LinesIn,LinesOut, Lev, PrLev, CtrNo)

OUTLINE (N, Lev, PrLev, CtrNo)

GETSYM (Lev, PrLev)

                        NEXTCH
                        READSCALE

                        CASE Ch OF
                                Letters ('A' .. 'Z')
                                Digits ('0' .. '9')
                                ' '
                                (
                                %, :
                                .
                                +
                                -

                                ;

WRITEINIT

WRITEFINAL

BLOCK (Level)

                        STATEMENT (CaseLev, Lev, PrLev)
                                CLEARLINE
                                     WRITEKOUNTER
                                ASSIGNMENT
                                IFSTATEMENT
                                BEGINSTATEMENT
                                REPEATSTATEMENT
                                DOSTATEMENT
                                CASESTATEMENT
                                .....ONECASE
                                LABELSTATEMENT
                                GOTOSTATEMENT

                        CHECKPAREN

MONITOR (main block)
```

C.2.2 INITIALIZE ABL/W4 ABSTRACT PROGRAM

```
PROCEDURE INITIALIZE(StatBegSym, ConstBegSym, Key, Ksym, LL, CC,
                     LinesIn, LinesOut, CtrNo, SavCtr, EndOfLin, Ch)
```

Initializes the variables and indices used in the program, specifies the elements of the set of statement begin symbols, the set of constant begin symbols, the array of function words [Key] and the array of function symbols.

All statements are assignment statements.

C.2.3 GENCOMM (LinesIn,LinesOut,Lev,PrLev,CtrNo)

This procedure generates a line on the command file XXXCOMM.

C.2.4 OUTLINE ABL/W4 ABSTRACT PROGRAM

```
OUTLINE(N,Lev,PrLev,CtrNo)


S1    X X
S2      X X
S3        X X
S4          X X
S5            X

P1    T F                   Suppress = FALSE
P2      T F                 Line[DD] = Blank AND DD <= N
P3        T T               N >= DD
P4        T F               I < N

A1          2               I = I + 1
A2                1         WRITELN(XXXFILE)
A3          1               WRITE(XXXFILE<Line[I]
A4        3                 I = DD
A5        2                 GENCOMM(LinesIn,LinesOut,Lev,PrLev,CtrNo)
A6    1                     DD = DD + 1
A7          1               LinesOut = LinesOut + 1
A8                2         DD = N + 1

NEXT 2 3 2 3 4 0 4 5 0      next state vector


PROCEDURE CALLS

GENCOMM(LineIn,LinesOut,Lev,PrLev,CtrNo)
```

C.2.5 GETSYM ABL/W4 ABSTRACT PROGRAM

```
S1    X X
S2      X X X
S3          X
S4            X X
S5                X X

P1    T F T T F             Ch = Blank
P2      T F                 NOT EOLN(PROG)
P3              T F         Ch IN Letters, Digits or Special Characters
P4                  T F     Sym = Other

A1      1           1       NEXTCH
A2        1                 OUTLINE(CC-1,Lev,PrLev,CtrNo)
A3        2                 EndOfLin = FALSE
A4              1           SavPos = CC
A5              2           Sym = Other
A6              1           CASE Ch OF

NEXT  2 3 2 2 3 4 5 5 0 0   next state vector
```

PROCEDURE CALLS

NEXTCH

C.2.51 NEXTCH ABL/W4 ABSTRACT PROGRAM

```
S1   X X
S2       X X
S3           X X
S4               X X
S5                   X X
S6                       X X
S7                           X
S8                             X X

P1   T F                     T F   T F   CC = LL
P2       T F                         |   EOF(PROG)
P3               T F                 |   Line[CC] = Blank AND CC <> LL
P4           F T                     |   EOLN(PROG)
P5                   T F             |   Suppress = TRUE

A1   1                               |   LinesIn = LinesIn + 1
A2   2                               |   EndOfLin = FALSE
A3       1                           |   GOTO 99
A4         1                         |   LL = 0
A5         2                         |   CC = 1
A6           1 1                     |   LL = LL + 1
A7           2 2                     |   READ(PROG,Line[LL])
A8               1           1       |   CC = CC + 1
A9                   1               |   DD = 1
A10                    1             |   DD = C
A11                          1       |   CC = CC - 1
A12                          2       |   Ch = Line[CC]
A13                            1     |   EndOfLin = TRUE

NEXT 2 1 0 3 3 4 4 5 6 6 7 7 8 0 0   next state vector
```

C.2.52 READSCALE ABL/W4 ABSTRACT PROGRAM

```
S1   X
S2     X X
S3         X X

P1         T F   Ch IN Digits
P2       T F     Ch IN ['+','-']

A1   1 1   1     NEXTCH

NEXT 2 3 3 3 0   next state vector
```

C.2.6 WRITEINIT ABL/W4 ABSTRACT PROGRAM

```
S1   X
S2       X X
S3          X X
S4              X X

P1       T F          Sym <> RParen
P2          T F       Sym <> ';'
P3              T F   Sym NOT IN [BeginSym, FuncSym, ProcSym]

A1   1                Suppress = FALSE
A2   2,3 1    1   1   GETSYM(1,1)
A3   4                XXXProgId = Id
A4           1        OUTLINE(CC-2,1,1,0)
A5           2   2    GENCOMM(0,0,0,0,0)
A6           3        WRITELN(XXXFILE,'XXXFILE,XXXRR,XXXCOMM')    *1
A7               1    OUTLINE(CC-1,1,1,0)
A8               3    WRITELN(XXXFILE,'(*$I''XXXTRA''?''XXXAAA''*)')    *2
A9                   1   --- described below                      *3
A10              4    Suppress = TRUE

NEXT 2   2 3 3 4 4 0  next state vector
```

*1 This write statement inserts the files used by profile into the program statement of the host program

*2 This write statement inserts a message to the compiler to include the source code XXXTRA from the file XXXAAA. This source code consists of declarations describing variables added to the program by profile.

*3 The declaration for the external procedure WPROFIL is inserted in the source code.

Then the "compromised" source code will execute according to its specifications while at the same time generating a record of its execution in the counter array.

PROCEDURES CALLED

GETSYM(1,1)

GENCOMM(LinesIn, LinesOut, Lev, PrLev,CtrNo)

C.2.7 PROCEDURE WRITEFINAL

WRITEFINAL writes the extra source code declarations on the file XXXAAA. When the edited file XXXFILE is compiled, these declarations will be included.

The array type XXXTYPE is defined with a specific size (0 .. CtrNo).
The constant XXXCTRNO is defined with the value of CtrNo.
Declarations are included for the added files XXXFILE, XXXRR, and XXXCOMM, as well as a declaration for the variable XXXPROGID needed to hold the name of the host program.
The array of counters XXXKOUNTER is defined and is initialized to zero.

WRITEFINAL also inserts a statement which assigns the host program's name into XXXPROGID and a procedure call to the external procedure WPROFIL. This procedure will then read the edited source program (XXXFILE) and the command file (XXXCOMM) and print the profile.

Example of an XXXAAA file created:

```
XXXTRA
  TYPE XXXTYPE= ARRAY[0..  320] OF INTEGER;
  CONST XXXCTRNO = 320;
  VAR XXXKOUNTER XXXTYPE;
  XXXFILE, XXXRR, XXXCOMM TEXT;
  XXXPROGID ALFA;
  VALUE XXXKOUNTER = ( 321 OF 0);
```

C.2.8 BLOCK ABL/W4 PROGRAM

```
S1   X X
S2       X X
S3           X X
S4               X X
S5                   X
S6                     X X
S7                         X X
S8                             X X
S9                                 X
S10                                    X X
S11                                        X
S12                                            X X
S13                                                X X
S14                                                    X

P1   T F                                                   Sym NOT IN [ProcSym,FuncSym,BeginSym,ExternSym,FORTRANSym,Forward
P2       T F                 T F                           Sym NOT IN [FORTRANSym,ExternSym,ForwardSym]
P3           T F                                           Sym IN [ProcSym,FuncSym
P4               T F                                       Sym = LParen
P5                       T F                               Sym IN [BeginSym,ExternSym,FORTRANSym,ForwardSym
P6                               T F                 T F   Sym = BeginSym
P7                                   T F                   DD < SavPos
P8                                       T F               Sym <> Period

A1   1           1,2             1                 1       GETSYM(1,1)
A2       1.                                           2   GENCOMM(0,0,99,0,0)
A3       2                                            3   WRITELN(XXXFILE)
A4               1                                       CHECKPAREN
A5                   1                                   BLOCK(Level+1)
A6                                   1                   Suppress = FALSE
A7                                       1           1   OUTLINE(SavPos-1,L,L,0)
A8                                           1           CtrNo = SavCtr
A9                                           2           STATEMENT(0,1,1)
A10                                          3           SavCtr = CtrNo
A11                                          4           CtrNo = 0
A12                                                  1   Suppress = TRUE
NEXT 1 2 3 7 4 4 5 5 6 7 7 8 1 9 0 1 1 1 1 1 1 1 1 0   next state vector
                                 0 1 1 2 3 3 4 4
```

PROCEDURE CALLS

```
GETSYM(1,1)

GENCOMM(0,0,99,0,0)

CHECKPAREN
```

```
BLOCK(Level+1)
OUTLINE(SavPos-1,1,1,0)
STATEMENT(0,1,1)
```

C.2.811 CLEARLINE ABL/W4 ABSTRACT PROGRAM

```
S1      X

A1      1   OUTLINE(SavPos-1,Lev,Prlev,CtrNo)
a2      2   DD = SavPos

next    0   next state vector
```

PROCEDURES CALLED

OUTLINE(N,Lev,Prlev,CtrNo)

C.2.812 WRITEKOUNTER

This procedure is used to write an imbedded command on the edited file (XXXFILE):

XXXKOUNTER[CtrNo] = XXXKOUNTER[CtrNo] + 1

The current value of CtrNo is used.

C.2.82 SYMBOL MASTER MACHINE

```
P1 ..... Sym C [semicolon] + StateBegSym
P2 ..... Sym C ConstBegSym
P3 ..... Sym C [ElseSym, Semicolon, UntilSym, EndSym]
P4 ..... Sym <> BeginSym
P5 ..... Sym <> ElseSym
P6 ..... Sym <> Period
P7 ..... Sym <> ThenSym
P8 ..... Sym <> Semicolon
P9 ..... Sym <> DoSym
P10 .... Sym <> Colon
P11 .... Sym <> OfSym
P12 .... Sym = EndSym
P13 .... Sym = OtherwiseSym
P14 .... Sym = UntilSym
P15 .... EndSwitch = TRUE
P16 .... DD < SavPos
P17 .... Lev <> CaseLev
P18 .... Lev <> 1

A1 ..... DD=CC
A2 ..... CC = SavPos
A3 ..... CtrNo = CtrNo + 1
A4 ..... CtrNo = 0
A5 ..... CtrNo = SavCtr
A6 ..... SavCtr = CtrNo
A7 ..... GenComm(0,0,0,0,0)
A8 ..... ClearLine
A9 ..... GetSym(Lev,PrLev)
A10 .... EndSwitch = TRUE
A11 .... EndSwitch = FALSE
A12 .... OneCase(Lev + 1, PrLev + 1)
A13 .... WriteKounter
A14 .... Statement(CaseLev,Lev+1, PrLev+1)
A15 .... Statement(CaseLev,Lev, PrLev+1)
A16 .... Statement(Lev,Lev,PrLev+1)
A17 .... Sym = Semicolon
A18 .... PrLev = PrLev - 1
A19 .... PrLev = PrLev + 1
A20 .... OutLine(CC-1, Lev, PrLev, CtrNo)
A21 .... OutLine(SavPos-1, Lev, PrLev, CtrNo)
A22 .... OutLine(CC-1, Lev+1, PrLev, CtrNo+1)
A23 .... OutLine(CC-1, Lev, PrLev, CtrNo+1)
A24 .... OutLine(SavPos-1, Lev, PrLev, CtrNo+1)
A25 .... WRITELN(XXXFILE,';')
A26 .... WRITELN(XXXFILE)
A27 .... WRITELN(XXXFILE,'END')
A28 .... WRITELN(XXXFILE,'BEGIN')
```

C.2.81 STATEMENT ABL/W4 ABSTRACT PROGRAM

```
S1   X X
S2       X X X X X X X X

P1    T F                    Sym IN StatBegSym + [Ident] + [IntSym]
P2        T                  Sym = Ident
P3          T                Sym = BeginSym
P4            T              Sym = IfSym
P5              T            Sym = CaseSym
P6                T          Sym = WhileSym, ForSym, DoSym
P7                  T        Sym = RepeatSym
P8                    T      Sym = GoToSym
P9                      T    Sym = IntSym

A1          1                ASSIGNMENT
A2            1              BEGINSTATEMENT
A3              1            IFSTATEMENT
A4                1          CASESTATEMENT
A5                  1        DOSTATEMENT
A6                    1      REPEATSTATEMENT
A7                      1    GOTOSTATEMENT
A8                        1  LABELSTATEMENT
NEXT 2 0 0 0 0 0 0 0 0   next state vector
```

PROCEDURE CALLS

ASSIGNMENT

BEGINSTATEMENT

IFSTATEMENT

CASESTATEMENT

DOSTATEMENT

REPEATSTATEMENT

GOTOSTATEMENT

LABELSTATEMENT

C.2.83 CHECKPAREN ABL/W4 ABSTRACT PROGRAM

```
S1    X
S2      X X X

P1      T T F     SYM <> PAREN
P2      T F       SYM = RPAREN

A1    1 2 1     GETSYM(1,1)
A2      1       CHECKPAREN
NEXT  2 2 2 0
```

PROCEDURES CALLED

GETSYM(Lev,PrLev)

CHECKPAREN

C.2.821 IFSTATEMENT ABSTRACT PROGRAM

```
S1   X
S2     X X
S3       X X
S4         X
S5           X X
S6             X
S7               X X
S8                 X X
S9                   X
S10                    X X
S11                      X
S12                        X X
S13                          X
S14                            X X
S15                              X X
S16                                X

P4             T F                   T F                    Sym <> BeginSym
P5                       T F   T F               T F        Sym <> ElseSym
P7     T F                                                  Sym <> ThenSym
P15                    T F                       T F        EndSwitch = TRUE
P16          T F                                            DD < SavPos

A1             3                     3                      DD=CC
A3               1         2           3         2          CtrNo = CtrNo + 1
A7               2       2               1       1          GenComm(0,0,0,0,0)
A8       1                                                  ClearLine
A9     1         4                     4                    GetSym(Lev,PrLev)
A10              5                       5                  EndSwitch = TRUE
A11    1                 1                                  EndSwitch = FALSE
A13              4           3           4         3        WriteKounter
A14                   1                       1            Statement(CaseLev,Lev+1, PrLev+1)
A18                             1                     1      PrLev = PrLev - 1
A19              1                     1                    PrLev = PrLev + 1
A22              2                     2                    OutLine(CC-1, Lev+1, PrLev, CtrNo+1)
A25                       1                       1          WRITELN(XXXFILE,';')
A26                         1                       1        WRITELN(XXXFILE)
A27                   3                       2            WRITELN(XXXFILE,'END')
A28              3                     2                    WRITELN(XXXFILE,'BEGIN')

NEXT 2 2 3 4 4 5 6 6 7 8 9 9 9 1 0 1 1 1 1 1 1 1 1 1 0     next state vector
                             0   1 2 3 3 4 5 6 6 6
```

C.2.822 BEGINSTATEMENT ABSTRACT PROGRAM

```
S1     X
S2       X X X
S3           X X
S4               X X
S5                   X X X
S6                         X X

P1     T T F                       Sym C [semicolon] + StateBegSym
P5           |       T F   T       Sym <> ElseSym
P6           |   T F               Sym <> Period
P8     T F   |       T T F         Sym <> Semicolon
P12          | T F                 Sym = EndSym
P17          |             T       Lev <> CaseLev
P18          |             T       Lev <> 1

A1   3                     2       DD=CC
A3   1                       1     CtrNo = CtrNo + 1
A7                       2         GenCmmm(0,0,0,0,0)
A8                       1 1       ClearLine
A9   5   1   1                     GetSym(Lev,PrLev)
A13  4                       2     WriteKounter
A15  6 2 2                         Statement(CaseLev,Lev, PrLev+1)
A20  2                     1       OutLine(CC-1, Lev, PrLev, CtrNo)
A25                    3           WRITELN(XXXFILE,';')

NEXT 2 2 2 3 4 0 5 0 6 6 6 0 0     next state vector
```

C.2.823 REPEATSTATEMENT ABSTRACT PROGRAM

```
S1   X X
S2     X
S3       X X X
S4           X X
S5               X X
S6                   X X
S7                       X X

P1       T T F                   Sym C [semicolon] + StateBegSym
P3           T F                 Sym C [ElseSym, Semicolon, UntilSym, EndSym]
P5                       T F     Sym <> ElseSym
P8       T F           T F       Sym <> Semicolon
P14          T F                 Sym = UntilSym
P16  T F                         DD < SavPos

A3       2                   1   CtrNo = CtrNo + 1
A7                       2       GenComm(0,0,0,0,0)
A8   1                           ClearLine
A9       4   1   1   1           GetSym(Lev,PrLev)
A13      3                   2   WriteKounter
A14      5 2 2                   Statement(CaseLev,Lev+1, PrLev+1)
A20      1                       OutLine(CC-1, Lev, PrLev, CtrNo)
A23                  1           OutLine(CC-1, Lev, PrLev, CtrNo+1)
A24                      1       OutLine(SavPos-1, Lev, PrLev, CtrNo+1)
A25                      3       WRITELN(XXXFILE,';')

NEXT 2 2 3 3 3 4 5 0 6 5 7 7 0 0   next state vector
```

C.2.824 DOSTATEMENT ABSTRACT PROGRAM

```
S1    X
S2      X X
S3        X
S4          X X
S5            X
S6              X X
S7                X X

P4          T F             Sym <> BeginSym
P5                T F       Sym <> ElseSym
P9      T F                 Sym <> DoSym
P15               T F       EndSwitch = TRUE

A1            2             DO=CC
A3            4       1     CtrNo = CtrNo + 1
A7            1   1         GenComm(0,0,0,0,0)
A9    1       3             GetSym(Lev,PrLev)
A10           3             EndSwitch = TRUE
A11   1                     EndSwitch = FALSE
A13           5       2     WriteKounter
A14               1         Statement(CaseLev,Lev+1, PrLev+1)
A20           1             OutLine(CC-1, Lev, PrLev, CtrNo)
A27               2         WRITELN(XXXFILE,'END')
A28           2             WRITELN(XXXFILE,'BEGIN')

NEXT 2 2 3 4 5 5 6 7 0 0 0    next state vector
```

C.2.825 CASESTATEMENT ABSTRACT PROGRAM

```
S1   X X
S2     X X
S3       X
S4         X X
S5           X X
S6             X X
S7               X
S8                 X X X
S9                     X X
S10                      X X
S11                        X X

P1                   T T F                 Sym C [semicolon] + StateBegSym
P8           T F     T F       T F         Sym <> Semicolon
P11    T F                                 Sym <> OfSym
P12                          T F           Sym = EndSym
P13              T F                       Sym = OtherwiseSym
P16  T F               T F                 DD < SavPos
P17                                T F     Lev <> CaseLev

A1                                 2       DD=CC
A3                   2                 1   CtrNo = CtrNo + 1
A4                             2           CtrNo = 0
A6                             1           SavCtr = CtrNo
A7                                 2       GenComm(0,0,0,0,0)
A8   1                 1           1       ClearLine
A9       1   2   1         4   1   3       GetSym(Lev,PrLev)
A12          3   2                         OneCase(Lev + 1, PrLev+ 1)
A13                  3                 2   WriteKounter
A14                  5 1 2                 Statement(CaseLev,Lev+1, PrLev+1)
A20          1       1                     OutLine(CC-1, Lev, PrLev, CtrNo)
A21                                1       OutLine(SavPos-1, Lev, PrLev, CtrNo)
A25                                3       WRITELN(XXXFILE,';')

NEXT 2 2 2 3 4 5 4 6 9 7 7 8 8 8 9 1 0 1 1 0 0   next state vector
                               0   1 1
```

C.2.8251 ONECASE ABSTRACT PROGRAM

```
    S1    X X
    S2        X X
    S3            X
    S4              X X
    S5                  X
    S6                    X X

    P2    T F                       Sym C ConstBegSym
    P4            T F               Sym <> BeginSym
    P10       T F                   Sym <> Colon
    P15                   T F       EndSwitch = TRUE

    A3                3             CtrNo = CtrNo + 1
    A7                1       1     GenComm(0,0,0,0,0)
    A9        1       3             GetSym(Lev,PrLev)
    A10               5             EndSwitch = TRUE
    A11             1               EndSwitch = FALSE
    A13               4             WriteKounter
    A16                   1         Statement(Lev,Lev,PrLev+1)
    A23             2               OutLine(CC-1, Lev, PrLev, CtrNo+1)
    A27                     2       WRITELN(XXXFILE,'END')
    A28               2             WRITELN(XXXFILE,'BEGIN')

    NEXT 2 0 2 3 4 5 5 6 0 0        next state vector
```

C.2.826 LABELSTATEMENT ABSTRACT PROGRAM

```
S1    X

A3    2    CtrNo = CtrNo + 1
A9    1    GetSym(Lev,PrLev)
A13   4    WriteKounter
A17   5    Sym = Semicolon
A20   3    OutLine(CC-1, Lev, PrLev, CtrNo)

NEXT  0    next state vector
```

C.2.827 GOTOSTATEMENT ABSTRACT PROGRAM

```
S1    X X
S2        X X X
S3              X X
S4                    X X

P3    T F                       Sym C [ElseSym, Semicolon, UntilSym, EndSym]
P5                        T F   Sym <> ElseSym
P8        T T     T F           Sym <> Semicolon
P16       T T F                 OD < SavPos

A3                        1     CtrNo = CtrNo + 1
A7                1             GenComm(0,0,0,0,0)
A8          1                   ClearLine
A9      1                       GetSym(Lev,PrLev)
A13                       2     WriteKounter
A20         1                   OutLine(CC-1, Lev, PrLev, CtrNo)
A25               2             WRITELN(XXXFILE,';')

NEXT  2 1 3 3 3 4 4 0 0         next state vector
```

C.2.9 MAIN ABL/W4 ABSTRACT PROGRAM

```
S1   X

A1   1     INITIALIZE
A2   2     WRITEINIT
A4   3     BLOCK(0)
A5   4     99
A6   5     WRITEFINAL
```

The main program consists of four procedure calls. The label 99 was used as a furthur test of the logic in profile program itself.

C.3 MONITOR: SOURCE LISTING

```
000004     1 (*$L50*)
000004     2 PROGRAM MONITOR (PROG,XXXFILE,OUTPUT,XXXAAA,XXXCOMM);
000040     3 (*$P+*)
000040     4 LABEL
000040     5   99;
000040     6
000040     7 CONST
000040     8   ALNG = 10;  (* NO. OF SIGNIFICANT CHARS IN IDENTIFIER *)
000040     9   ASTERICK = '*';
000040    10   BLANK = ' ';
000040    11   COMMA = ',';
000040    12   EQUALSGN = '=';
000040    13   LLNG = 121; (* INPUT LINE LENGTH *)
000040    14   NKW = 21;   (* NO.OF KEY WORDS *)
000040    15
000040    16 TYPE
000040    17   SYMBOL = (IDENT,INTSYM,RPAREN,SEMICOLON,PROGSYM,FUNCSYM,PROCSYM,PERIOD,STRING,BECOMES,
000040    18             BEGINSYM,IFSYM,CASESYM,REPEATSYM,WHILESYM,FORSYM,ENDSYM,LPAREN,
000040    19             ELSESYM,UNTILSYM,DOSYM,THENSYM,GOTOSYM,WITHSYM,
000040    20             FORTRANSYM,EXTERNSYM,FORWARDSYM,OTHERWISESYM,OTHER,REALSYM,COLON,OFSYM,
000040    21             CHARCON, PLUS, MINUS);
000040    22   SYMSET = SET OF SYMBOL;
000040    23   KEYTYPE = ARRAY [ 1 .. NKW ] OF ALFA;
000040    24   KSYMTYPE = ARRAY [ 1 .. NKW ] OF SYMBOL;
000040    25
000040    26   LINETYPE = ARRAY [ 1 .. LLNG] OF CHAR;
000040    27   LETTTYPE = 'A'..'Z';
000040    28   DIGTYPE = '0'..'9';
000040    29
000040    30 VAR
000040    31   (* FILES *)
000040    32   XXXFILE,          (* EDITED SOURCE FILE *)
000040    33   XXXCOMM,          (* CONTROL FILE FOR XXXFILE - 1:1 CORRESPONDENCE *)
000040    34   XXXAAA,           (* ADDITIONAL SOURCE CODE TO ADD TO HOST PROGRAM *)
000040    35   PROG :            (* ORIGINAL PROGRAM *)
000040    36         TEXT;
000220    37
000220    38   (*  *)
000220    39   SAVPOS,    (* INDEX TO FIRST CHARACTER OF PRESENT IDENTIFIER *)
000220    40   DD,        (*  INDEX TO FIRST NON BLANK CHARACTER OF INPUT LINE TO BE PRINTED *)
000220    41   CC,        (* INDEX TO NEXT CHARACTER *)
000220    42   LL         (* INDEX TO LAST CHARACTER OF INPUT LINE *)
000220    43       : INTEGER;
000224    44   (*  *)
000224    45   SUPPRESS,              (* USED TO KEEP ORIGINAL SPACING OF DEFINITION AND DECLARATION PART OF PRO
000224    46   ENDOFLIN : BOOLEAN;  (* CC = LL *)
000226    47
```

```
000226     48   LETTERS : SET OF LETTTYPE;
000227     49   DIGITS : SET OF DIGTYPE;
000230     50   LINE : LINETYPE;
000421     51   ID, INTVAL, XXXPROGID : ALFA;
000424     52   CH : CHAR;
000425     53   SYM : SYMBOL;
000426     54
000426     55   KEY : KEYTYPE;
000453     56   KSYM : KSYMTYPE;
000500     57
000500     58   LINESIN, LINESOUT, CTRNO, SAVCTR : INTEGER;
000504     59   CONSTBEGSYM, STATBEGSYM : SYMSET;
000506     60
000506     61 PROCEDURE INITIALIZE (VAR STATBEGSYM, CONSTBEGSYM : SYMSET;
000004     62                        VAR KEY : KEYTYPE;
000005     63                        VAR KSYM : KSYMTYPE;
000006     64                        VAR LL, CC : INTEGER;
000010     65                        VAR LINESIN, LINESOUT, CTRNO, SAVCTR : INTEGER;
000014     66                        VAR ENDOFLIN:BOOLEAN;
000015     67                        VAR CH : CHAR);
000016     68
000016     69 BEGIN
000016     70   RESET (PROG);
000006     71   REWRITE (XXXFILE);
000010     72   REWRITE (XXXCOMM);
000012     73   STATBEGSYM := [BEGINSYM, IFSYM, CASESYM, WHILESYM, FORSYM, WITHSYM,
000013     74                  REPEATSYM, GOTOSYM];
000015     75   CONSTBEGSYM := [PLUS, MINUS, INTSYM, REALSYM, CHARCON, IDENT];
000017     76   KEY [  1] := 'BEGIN      ';
000020     77   KEY [  2] := 'CASE       ';
000022     78   KEY [  3] := 'DO         ';
000023     79   KEY [  4] := 'ELSE       ';
000025     80   KEY [  5] := 'END        ';
000026     81   KEY [  6] := 'EXTERN     ';
000030     82   KEY [  7] := 'FORTRAN    ';
000031     83   KEY [  8] := 'FORWARD    ';
000033     84   KEY [  9] := 'FOR        ';
000034     85   KEY [ 10] := 'FUNCTION   ';
000036     86   KEY [ 11] := 'GOTO       ';
000037     87   KEY [ 12] := 'IF         ';
000041     88   KEY [ 13] := 'OF         ';
000042     89   KEY [ 14] := 'OTHERWISE  ';
000044     90   KEY [ 15] := 'PROCEDURE  ';
000045     91   KEY [ 16] := 'PROGRAM    ';
000047     92   KEY [ 17] := 'REPEAT     ';
000050     93   KEY [ 18] := 'THEN       ';
000052     94   KEY [ 19] := 'UNTIL      ';
```

```
000053    95    KEY [ 20] := 'WHILE      ';
000055    96    KEY [ 21] := 'WITH       ';
000056    97    KSYM [  1] := BEGINSYM;
000060    98    KSYM [  2] := CASESYM;
000062    99    KSYM [  3] := DOSYM;
000063   100    KSYM [  4] := ELSESYM;
000065   101    KSYM [  5] := ENDSYM;
000066   102    KSYM [  6] := EXTERNSYM;
000070   103    KSYM [  7] := FORTRANSYM;
000071   104    KSYM [  8] := FORWARDSYM;
000073   105    KSYM [  9] := FORSYM;
000074   106    KSYM [ 10] := FUNCSYM;
000076   107    KSYM [ 11] := GOTOSYM;
000077   108    KSYM [ 12] := IFSYM;
000101   109    KSYM [ 13] := OFSYM;
000102   110    KSYM [ 14] := OTHERWISESYM;
000104   111    KSYM [ 15] := PROCSYM;
000105   112    KSYM [ 16] := PROGSYM;
000107   113    KSYM [ 17] := REPEATSYM;
000110   114    KSYM [ 18] := THENSYM;
000112   115    KSYM [ 19] := UNTILSYM;
000113   116    KSYM [ 20] := WHILESYM;
000115   117    KSYM [ 21] := WITHSYM;
000116   118
000116   119
000116   120    LL := 0;
000120   121    CC := 0;
000121   122    CH := BLANK;
000123   123    LINESIN := 0;
000124   124    LINESOUT := 0;
000125   125    CTRNO := 0;
000127   126    SAVCTR := 0;
000130   127    LETTERS := [ 'A' .. 'Z'];
000132   128    DIGITS := [ '0' .. '9'];
000133   129
000133   130
000133   131    ENDOFLIN := FALSE;
000135   132  END;
000206   133
000206   134
000206   135
000206   136
000206   137  PROCEDURE GENCOMM (LINESIN, LINESOUT, LEV, PRLEV, CTRNO : INTEGER);
000007   138
000007   139
000007   140  BEGIN
000007   141    WRITE (XXXCOMM,LINESIN:5,LINESOUT:6,LEV:3,PRLEV:3,CTRNO:5);
```

```
000030   142   WRITELN(XXXCOMM)
000030   143 END;
000046   144
000046   145 PROCEDURE OUTLINE (N, LEV, PRLEV, CTRNO : INTEGER);
000006   146
000006   147
000006   148   (*  GLOBAL VARIABLES USED :     LINE, LINESIN, SUPPRESS
000006   149                           MODIFIED  DD, LINESOUT  *)
000006   150
000006   151 VAR
000006   152   I : INTEGER;
000007   153
000007   154 BEGIN
000007   155   IF NOT SUPPRESS THEN
000006   156     WHILE (LINE[DD] = BLANK) AND (DD <= N) DO
000015   157       DD := DD + 1;
000017   158   IF N >= DD
000017   159     THEN
000022   160       BEGIN
000022   161         LINESOUT := LINESOUT + 1;
000024   162         GENCOMM (LINESIN, LINESOUT, LEV, PRLEV, CTRNO);
000027   163         FOR I := DD TO N DO
000032   164           WRITE (XXXFILE, LINE[I]);
000050   165         WRITELN (XXXFILE);
000051   166         DD := N + 1
000051   167       END
000053   168 END;    (* OUTLINE *)
000067   169
000067   170 PROCEDURE GETSYM (LEV, PRLEV : INTEGER ) ;
000004   171 (*
000004   172         GLOBAL VARIABLES ACCESSED  CTRNO, LINESOUT
000004   173                          MODIFIED  ID, INTVAL, SYM, SAVPOS   *)
000004   174
000004   175
000004   176 LABEL
000004   177   1;
000004   178
000004   179 VAR
000004   180   I, J, K : INTEGER;
000007   181   COMMFIN : BOOLEAN;
000010   182   STRFIN : BOOLEAN;
000011   183   STORE : INTEGER;
000012   184   COMMSWT : BOOLEAN;
000013   185
000013   186 PROCEDURE NEXTCH;
000002   187
000002   188 (*     GLOBAL VARIABLES MODIFIED CC, LL, DD, LINESIN, ENDOFLIN, LINE, CH  *)
```

```
000002   189
000002   190
000002   191 BEGIN
000002   192   IF CC = LL
000004   193     THEN
000006   194       BEGIN
000006   195         WHILE CC = LL DO
000010   196           BEGIN
000010   197             LINESIN := LINESIN + 1;
000012   198             ENDOFLIN := FALSE;
000014   199             IF EOF (PROG)
000014   200               THEN
000015   201                 BEGIN
000015   202                   GOTO 99
000017   203                 END;
000017   204             LL := 0;
000021   205             CC := 1;
000022   206             WHILE NOT EOLN(PROG) DO
000024   207               BEGIN
000024   208                 LL := LL + 1;
000026   209                 READ (PROG, LINE[LL])
000035   210               END;
000036   211             LL := LL + 1;
000040   212             READ  (PROG, LINE[LL]);
000050   213             WHILE (LINE [CC] = BLANK) AND (CC <> LL) DO
000057   214               CC := CC + 1;
000061   215             IF SUPPRESS THEN
000063   216               DD := 1
000063   217             ELSE
000064   218               DD := CC;
000065   219             IF CC <> LL
000065   220               THEN
000067   221                 CC := CC - 1;
000071   222         END
000071   223       END;
000072   224   CC := CC + 1;
000074   225   CH := LINE [CC];
000102   226   IF CC = LL THEN ENDOFLIN := TRUE;
000104   227 END;   (* NEXTCH *)
000106   228
000106   229 PROCEDURE READSCALE;
000002   230
000002   231 BEGIN
000002   232   NEXTCH;
000006   233   IF CH IN [ '+', '-' ] THEN
000011   234     NEXTCH;
000013   235   WHILE CH IN DIGITS DO
```

```
000016   236      NEXTCH;
000021   237  END;
000023   238
000023   239
000023   240  BEGIN (* GETSYM *)
000023   241    1 : WHILE CH = BLANK DO
000007   242          BEGIN
000007   243            WHILE  ( CH = BLANK) AND (NOT ENDOFLIN ) DO
000014   244              NEXTCH;
000016   245            IF ENDOFLIN THEN
000017   246              BEGIN
000017   247                OUTLINE (CC-1,LEV,PRLEV,CTRNO);
000023   248                ENDOFLIN := FALSE;
000025   249              END;
000025   250          END;
000026   251        SAVPOS := CC;
000030   252        SYM := OTHER;
000032   253        IF ( CH IN LETTERS) OR (CH IN DIGITS) OR (CH IN ['''','(',')','+','-','.',',',':','%'])
000036   254            OR (CH = ';') THEN
000041   255        CASE CH OF
000042   256        'A','B','C','D','E','F','G','H',
000042   257        'I','J','K','L','M','N','O','P',
000042   258        'Q','R','S','T','U','V','W','X',
000042   259        'Y','Z'
000042   260          BEGIN
000042   261            K := 0;
000044   262            ID := '          ';
000046   263            REPEAT
000046   264              IF K < ALNG THEN
000051   265                BEGIN
000051   266                  K := K + 1;
000052   267                  ID[K] := CH
000055   268                END;
000062   269              NEXTCH;
000064   270            UNTIL NOT ( CH IN ['A' .. 'Z', '0' .. '9'] );
000067   271            IF K < ALNG THEN
000072   272              BEGIN
000072   273                I :=1;
000073   274                J := NKW;
000075   275                REPEAT
000075   276                  K := ( I + J) DIV 2;
000100   277                  IF ID <= KEY [K] THEN
000106   278                    J := K - 1;
000107   279                  IF ID >= KEY [K] THEN
000115   280                    I := K + 1;
000117   281                UNTIL I > J;
000121   282                IF ( I - 1 ) > J
```

```
000122  283                 THEN
000123  284                   SYM := KSYM[K]
000126  285                 ELSE SYM := IDENT;
000133  286             END
000133  287           ELSE
000134  288             SYM := IDENT;
000135  289         END;
000136  290         '1','2','3','4','5',
000136  291         '6','7','8','9','0'  :
000136  292           BEGIN
000136  293             K := 0;
000140  294             SYM := INTSYM;
000141  295             REPEAT
000141  296               IF K < 5 THEN
000144  297                 BEGIN
000144  298                   K := K + 1;
000145  299                   INTVAL[K] := CH;
000156  300                 END;
000156  301               NEXTCH;
000160  302             UNTIL NOT (CH IN ['0' .. '9']);
000163  303             IF CH = '.' THEN
000165  304               BEGIN
000165  305                 NEXTCH;
000167  306                 IF CH = '.'
000167  307                   THEN
000172  308                     CH := ':'
000172  309                   ELSE
000173  310                     BEGIN
000173  311                       SYM := REALSYM;
000175  312                       WHILE CH IN ['0' ..'9'] DO
000200  313                         NEXTCH;
000203  314                       IF CH = 'E' THEN
000205  315                           READSCALE
000205  316                       END
000207  317             END
000207  318           ELSE
000210  319             IF CH = 'E  THEN
000212  320               BEGIN
000212  321                 SYM :=REALSYM;
000214  322                 READSCALE
000214  323               END
000215  324         END;
000216  325     '''' :
000216  326     BEGIN
000216  327       STRFIN := FALSE;
000220  328       SYM := CHARCON;
000221  329        K := 0;
```

```
000223   330        REPEAT
000223   331          NEXTCH;
000225   332          WHILE CH <> '''' DO
000230   333            BEGIN
000230   334              K := K + 1;
000232   335              NEXTCH;
000233   336            END;
000234   337          IF K > 1 THEN
000236   338            SYM := STRING;
000240   339          NEXTCH;
000242   340          IF CH <> '''' THEN
000245   341            STRFIN := TRUE
000245   342          ELSE
000247   343            BEGIN
000247   344              NEXTCH;
000251   345              IF CH = '''' THEN
000254   346                BEGIN
000254   347                  STRFIN := TRUE;
000255   348                  NEXTCH
000255   349                END;
000257   350            END
000257   351        UNTIL STRFIN;
000261   352      END;
000262   353    '(' :
000262   354      BEGIN
000262   355        NEXTCH;
000264   356        IF CH = ASTERICK THEN
000267   357          BEGIN
000267   358            COMMFIN := FALSE;
000270   359            IF NOT SUPPRESS THEN
000272   360              WHILE LINE[DD] = ' ' DO
000277   361                DD := DD + 1;
000301   362            IF DD = (CC - 1) THEN
000304   363              COMMSWT := TRUE
000304   364            ELSE
000306   365              COMMSWT := FALSE;
000310   366            NEXTCH;
000312   367            REPEAT
000312   368              WHILE (CH <> ASTERICK) AND NOT ENDOFLIN  DO
000317   369                NEXTCH;
000321   370              IF CH = ASTERICK THEN
000323   371                BEGIN
000323   372                  NEXTCH;
000324   373                  IF CH = ')' THEN
000327   374                    COMMFIN := TRUE;
000330   375                END;
000330   376            UNTIL COMMFIN OR ENDOFLIN;
```

```
000333   377            IF ENDOFLIN THEN
000334   378              IF COMMSWT THEN
000335   379                OUTLINE (CC-1, LEV, PRLEV, 0)
000340   380              ELSE
000342   381                OUTLINE (CC-1, LEV, PRLEV, CTRNO);
000346   382            IF NOT COMMFIN THEN
000350   383                REPEAT
000350   384                  NEXTCH;
000352   385                  DD := 1;
000354   386                  WHILE NOT ENDOFLIN AND NOT COMMFIN DO
000357   387                    BEGIN
000357   388                      WHILE NOT ENDOFLIN AND (CH <> ASTERICK) DO
000364   389                        NEXTCH;
000366   390                      IF CH = ASTERICK THEN
000370   391                        BEGIN
000370   392                          NEXTCH;
000371   393                          IF CH = ')' THEN
000374   394                            COMMFIN := TRUE;
000375   395                        END;
000375   396                      IF ENDOFLIN THEN
000377   397                        IF COMMSWT THEN
000400   398                          OUTLINE (CC-1, LEV, PRLEV, 0)
000403   399                        ELSE
000405   400                          OUTLINE (CC-1, LEV, PRLEV, CTRNO);
000411   401                    END;
000412   402                UNTIL COMMFIN;
000413   403              NEXTCH;
000415   404              WHILE (CH = ' ') AND (NOT ENDOFLIN) DO
000422   405                NEXTCH;
000424   406              IF ENDOFLIN THEN
000425   407                IF COMMSWT THEN
000427   408                  OUTLINE (CC-1, LEV, PRLEV, 0)
000432   409                ELSE
000433   410                  OUTLINE (CC-1, LEV, PRLEV, CTRNO);
000440   411              GOTO 1;
000441   412          END
000441   413        ELSE
000442   414          SYM := LPAREN
000442   415    END;
000445   416  ':' : BEGIN
000445   417            NEXTCH;
000447   418            IF CH = '=' THEN
000452   419              BEGIN
000452   420                SYM := BECOMES;
000453   421                NEXTCH
000453   422              END
000455   423            ELSE
```

```
000456   424                 SYM := COLON;
000460   425               END;
000461   426
000461   427     '.' : BEGIN
000461   428             NEXTCH;
000463   429             IF CH = ',' THEN
000466   430               BEGIN
000466   431                 SYM := COLON;
000467   432                 NEXTCH
000467   433               END
000471   434             ELSE
000472   435               SYM := PERIOD;
000474   436         END;
000475   437
000475   438     '+' : BEGIN SYM := PLUS; NEXTCH END;
000500   439     '-' : BEGIN SYM := MINUS; NEXTCH END;
000503   440     ';' : BEGIN SYM := SEMICOLON; NEXTCH END;
000506   441     ')' : BEGIN SYM := RPAREN; NEXTCH END;
000511   442   END;   (* CASE *)
000554   443 IF SYM = OTHER THEN NEXTCH
000557   444 END;   (* GETSYMBOL *)
000606   445
000606   446 PROCEDURE WRITEINIT ;
000002   447
000002   448 BEGIN
000002   449   SUPPRESS := FALSE;
000006   450   GETSYM(1,1);
000007   451   GETSYM(1,1);
000011   452   XXXPROGID := ID;
000013   453   WHILE SYM <> RPAREN DO
000015   454     GETSYM(1,1);
000020   455   OUTLINE(CC-2,1,1,0);
000023   456   GENCOMM(0,0,0,0,0);
000026   457   WRITELN(XXXFILE, ',XXXFILE,XXXRR,XXXCOMM');
000034   458   WHILE SYM <> SEMICOLON DO
000037   459     GETSYM(1,1);
000041   460   OUTLINE(CC-1,1,1,0);
000044   461   GENCOMM(0,0,0,0,0);
000047   462   WRITELN(XXXFILE,'(*$I''XXXTRA''/''XXXAAA''*)');
000055   463   SUPPRESS := TRUE;
000057   464   WHILE NOT (SYM IN [BEGINSYM, FUNCSYM, PROCSYM]) DO
000062   465     GETSYM(1,1);
000065   466   GENCOMM(0,0,0,0,0);
000070   467   WRITELN(XXXFILE,'(*$X4*)');
000076   468   GENCOMM(0,0,0,0,0);
000101   469   WRITELN(XXXFILE,'PROCEDURE WPROFIL(XXXKOUNTER:DYNAMIC XXXTYPE;XXXCTRNO:INTEGER; ');
000107   470   GENCOMM(0,0,0,0,0);
```

```
000112   471    WRITELN(XXXFILE,' XXXPROGID:ALFA;VAR XXXFILE, XXXRR, XXXCOMM : TEXT);');
000120   472    GENCOMM(0,0,0,0,0);
000123   473    WRITELN(XXXFILE,' EXTERN;');
000131   474    GENCOMM(0,0,0,0,0);
000134   475    WRITELN(XXXFILE,'(*$X+ *)');
000142   476  END;
000172   477
000172   478  PROCEDURE WRITEFINAL;
000002   479
000002   480  BEGIN
000002   481   CTRNO := SAVCTR;
000006   482    GENCOMM(0,0,0,0,0);
000011   483    WRITELN(XXXFILE,'XXXPROGID := ''', XXXPROGID, ''';');
000030   484    GENCOMM(0,0,0,0,0);
000033   485    WRITELN(XXXFILE,'WPROFIL(XXXKOUNTER,',CTRNO:4,',XXXPROGID,XXXFILE,XXXRR,XXXCOMM);');
000052   486    OUTLINE(CC-1,1,1,CTRNO);
000055   487    RESET(XXXFILE);
000057   488    REWRITE(XXXAAA);
000061   489    WRITELN(XXXAAA,'XXXTRA');
000067   490    WRITELN(XXXAAA,' TYPE XXXTYPE= ARRAY[0..',CTRNO:4,'] OF INTEGER;');
000106   491    WRITELN(XXXAAA,' CONST XXXCTRNO = ',CTRNO:4,';');
000124   492    WRITELN(XXXAAA,' VAR XXXKOUNTER : XXXTYPE;');
000132   493    WRITELN(XXXAAA,' XXXFILE, XXXRR, XXXCOMM : TEXT;');
000140   494    WRITELN(XXXAAA,' XXXPROGID : ALFA;');
000146   495    CTRNO := CTRNO + 1;
000150   496    WRITELN(XXXAAA,' VALUE XXXKOUNTER = (',CTRNO:4,' OF 0);');
000167   497    RESET(XXXAAA);
000171   498  END;
000231   499
000231   500  PROCEDURE BLOCK (LEVEL : INTEGER);
000003   501
000003   502  PROCEDURE STATEMENT(CASELEV,LEV,PRLEV : INTEGER);
000005   503
000005   504  VAR
000005   505    I : INTEGER;
000006   506
000006   507  PROCEDURE CLEARLINE;
000002   508
000002   509  BEGIN
000002   510    OUTLINE (SAVPOS - 1, LEV, PRLEV, CTRNO);
000011   511    DO := SAVPOS
000011   512  END;
000015   513
000015   514  PROCEDURE WRITEKOUNTER;
000002   515
000002   516  BEGIN
000002   517    GENCOMM (0,0,0,0,0);
```

```
000007   518    WRITE (XXXFILE, 'XXXKOUNTER[', CTRNO:1, '] := XXXKOUNTER[',
000023   519           CTRNO:1, '] + 1;');
000034   520    WRITELN (XXXFILE);
000036   521  END;
000045   522
000045   523  PROCEDURE ASSIGNMENT;
000002   524
000002   525  BEGIN
000002   526    WHILE NOT (SYM IN [SEMICOLON, ENDSYM, UNTILSYM, ELSESYM]) DO
000007   527      GETSYM(LEV,PRLEV);
000013   528    IF SYM = SEMICOLON THEN
000015   529    BEGIN
000015   530        OUTLINE(CC-1, LEV, PRLEV, CTRNO);
000022   531        DD := CC
000022   532      END
000024   533    ELSE
000025   534      BEGIN
000025   535        OUTLINE(SAVPOS-1, LEV, PRLEV, CTRNO);
000032   536        DD := SAVPOS;
000034   537        GENCOMM(0,0,0,0,0);
000037   538        WRITELN(XXXFILE,';');
000045   539      END;
000045   540  END;   (* ASSIGNMENT *)
000050   541
000050   542  PROCEDURE IFSTATEMENT;
000002   543
000002   544  VAR
000002   545    ENDSWITCH : BOOLEAN;
000003   546
000003   547  BEGIN
000003   548    ENDSWITCH := FALSE;
000006   549    WHILE SYM <> THENSYM DO
000011   550      GETSYM(LEV,PRLEV);
000014   551    IF DD < SAVPOS THEN
000016   552      CLEARLINE;
000020   553    PRLEV := PRLEV + 1;
000023   554    OUTLINE(CC-1,LEV+1,PRLEV,CTRNO+1);
000027   555    DD := CC;
000031   556    GETSYM(LEV,PRLEV);
000033   557    IF SYM <> BEGINSYM THEN
000036   558      BEGIN
000036   559        CTRNO := CTRNO + 1;
000040   560        GENCOMM(0,0,0,0,0);
000042   561        WRITELN(XXXFILE,'BEGIN');
000050   562        WRITEKOUNTER;
000052   563        ENDSWITCH := TRUE
000052   564      END;
```

```
000054   565    STATEMENT(CASELEV,LEV+1,PRLEV+1);
000060   566    IF ENDSWITCH THEN
000062   567      BEGIN
000062   568        ENDSWITCH := FALSE;
000063   569        GENCOMM(0,0,0,0,0);
000066   570        WRITE(XXXFILE,'END');
000073   571        IF SYM <> ELSESYM THEN
000076   572          BEGIN
000076   573            WRITELN(XXXFILE,';');
000103   574            CTRNO := CTRNO + 1;
000105   575            WRITEKOUNTER
000105   576          END
000107   577        ELSE
000110   578          WRITELN(XXXFILE);
000112   579      END;
000112   580    PRLEV := PRLEV -1;
000115   581    IF SYM = ELSESYM THEN
000117   582      BEGIN
000117   583        PRLEV := PRLEV + 1;
000120   584        OUTLINE(CC-1,LEV +1, PRLEV, CTRNO + 1);
000124   585        DD := CC;
000126   586        GETSYM (LEV,PRLEV);
000130   587        IF SYM <> BEGINSYM THEN
000133   588          BEGIN
000133   589            GENCOMM(0,0,0,0,0);
000136   590            WRITELN(XXXFILE,'BEGIN');
000144   591            CTRNO := CTRNO + 1;
000146   592            WRITEKOUNTER;
000150   593            ENDSWITCH := TRUE;
000152   594          END;
000152   595
000152   596    STATEMENT(CASELEV,LEV + 1,PRLEV + 1);
000156   597    IF ENDSWITCH THEN
000160   598      BEGIN
000160   599        GENCOMM(0,0,0,0,0);
000163   600        WRITE(XXXFILE,'END');
000170   601        IF SYM <> ELSESYM THEN
000173   602          BEGIN
000173   603            WRITELN(XXXFILE,';');
000200   604            CTRNO := CTRNO + 1;
000202   605            WRITEKOUNTER;
000204   606          END
000204   607        ELSE
000205   608          WRITELN(XXXFILE);
000207   609      END;
000207   610    PRLEV := PRLEV - 1;
000212   611  END
```

```
000212   612 END;  (*  IFSTATEMENT  *)
000220   613
000220   614 PROCEDURE BEGINSTATEMENT;
000002   615
000002   616 BEGIN
000002   617   CTRNO := CTRNO + 1;
000006   618   OUTLINE(CC-1,LEV,PRLEV,CTRNO);
000012   619   DD := CC;
000014   620   WRITEKOUNTER;
000015   621   GETSYM(LEV,PRLEV);
000020   622   STATEMENT(CASELEV,LEV,PRLEV+1);
000024   623   WHILE SYM IN ([SEMICOLON] + STATBEGSYMS) DO
000030   624     BEGIN
000030   625       IF SYM = SEMICOLON THEN
000032   626         GETSYM(LEV,PRLEV);
000035   627       STATEMENT(CASELEV,LEV,PRLEV +1);
000041   628     END;
000042   629   IF SYM = ENDSYM THEN
000044   630     BEGIN
000044   631       (* CTRNO := CTRNO + 1;
000044   632        WRITEKOUNTER; *)
000044   633       GETSYM(LEV,PRLEV);
000047   634       IF SYM <> PERIOD THEN
000052   635         BEGIN
000052   636           IF SYM = SEMICOLON THEN
000054   637             BEGIN
000054   638               OUTLINE(CC-1,LEV,PRLEV,CTRNO);
000060   639               DD := CC;
000062   640             END
000062   641           ELSE
000063   642             BEGIN
000063   643               CLEARLINE;
000065   644               IF SYM <> ELSESYM THEN
000070   645                 BEGIN
000070   646                   GENCOMM(0,0,0,0,0);
000073   647                   WRITELN(XXXFILE,';');
000101   648                 END;
000101   649             END;
000101   650           IF (SYM <> ELSESYM) AND(LEV <> 1) AND (LEV <> CASELEV)
000106   651             THEN BEGIN
000110   652               CTRNO := CTRNO + 1;
000111   653               WRITEKOUNTER;
000113   654             END;
000113   655       END
000113   656     END
000113   657   ELSE
000114   658
```

```
000114   659 END;   (*  BEGINSTATEMENT  *)
000116   660
000116   661
000116   662 PROCEDURE REPEATSTATEMENT;
000002   663
000002   664 BEGIN
000002   665   IF DD < SAVPOS THEN
000006   666     CLEARLINE;
000010   667   OUTLINE(CC-1,LEV,PRLEV,CTRNO);
000015   668   CTRNO := CTRNO + 1;
000017   669   WRITEKOUNTER;
000021   670   GETSYM(LEV,PRLEV);
000024   671   STATEMENT(CASELEV,LEV +1 ,PRLEV + 1);
000030   672   WHILE SYM IN [SEMICOLON] + STATBEGSYM DO
000034   673     BEGIN
000034   674       IF SYM = SEMICOLON THEN
000036   675         GETSYM(LEV,PRLEV);
000041   676       STATEMENT(CASELEV,LEV + 1, PRLEV + 1);
000045   677     END;
000046   678   IF SYM = UNTILSYM THEN
000050   679     BEGIN
000050   680       GETSYM(LEV,PRLEV);
000053   681       WHILE NOT (SYM IN [ENDSYM, SEMICOLON, ELSESYM, UNTILSYM]) DO
000056   682         GETSYM(LEV,PRLEV);
000062   683       IF SYM = SEMICOLON THEN
000064   684         OUTLINE(CC-1,LEV,PRLEV,CTRNO + 1)
000071   685       ELSE
000072   686         BEGIN
000072   687           OUTLINE(SAVPOS-1,LEV,PRLEV,CTRNO+1);
000077   688           GENCOMM(0,0,0,0,0);
000102   689           WRITELN(XXXFILE,';')
000107   690         END;
000110   691       IF SYM <> ELSESYM THEN
000113   692         BEGIN
000113   693           CTRNO := CTRNO + 1;
000115   694           WRITEKOUNTER
000115   695         END
000116   696     END
000116   697   ELSE
000117   698
000117   699 END;  (*  REPEATSYM  *)
000122   700
000122   701
000122   702 PROCEDURE DOSTATEMENT;
000002   703
000002   704 VAR
000002   705   ENDSWITCH : BOOLEAN;
```

```
000003   706
000003   707 BEGIN
000003   708   ENDSWITCH := FALSE;
000006   709   WHILE SYM <> DOSYM DO
000011   710     GETSYM(LEV,PRLEV);
000014   711   OUTLINE(CC-1,LEV,PRLEV,CTRNO);
000021   712   DD := CC;
000023   713   GETSYM(LEV,PRLEV);
000025   714   IF SYM <> BEGINSYM THEN
000030   715     BEGIN
000030   716       GENCOMM(0,0,0,0,0);
000033   717       WRITELN(XXXFILE,'BEGIN');
000041   718       ENDSWITCH := TRUE;
000043   719       CTRNO := CTRNO + 1;
000045   720       WRITEKOUNTER;
000046   721     END;
000046   722   STATEMENT(CASELEV,LEV +1,PRLEV + 1);
000052   723   IF ENDSWITCH THEN
000054   724     BEGIN
000054   725       GENCOMM(0,0,0,0,0);
000057   726       WRITELN(XXXFILE,'END;');
000065   727       IF SYM <> ELSESYM THEN
000070   728         BEGIN
000070   729           CTRNO := CTRNO + 1;
000072   730           WRITEKOUNTER;
000073   731         END
000073   732     END;
000073   733 END;  (*  DOSTATEMENT *)
000101   734
000101   735
000101   736 PROCEDURE CASESTATEMENT;
000002   737
000002   738
000002   739 PROCEDURE ONECASE(LEV,PRLEV:INTEGER);
000004   740 VAR
000004   741   ENDSWITCH : BOOLEAN;
000005   742
000005   743 BEGIN
000005   744   IF SYM IN CONSTBEGSYM THEN
000007   745     BEGIN
000007   746       WHILE SYM <> COLON DO
000012   747         GETSYM(LEV,PRLEV);
000015   748       ENDSWITCH := FALSE;
000017   749       OUTLINE(CC-1,LEV,PRLEV,CTRNO + 1);
000023   750       GETSYM(LEV,PRLEV);
000026   751       IF SYM <> BEGINSYM THEN
000031   752         BEGIN
```

```
000031   753            GENCOMM(0,0,0,0,0);
000034   754            WRITELN(XXXFILE,'BEGIN');
000042   755            CTRNO := CTRNO + 1;
000044   756            WRITEKOUNTER;
000046   757            ENDSWITCH := TRUE
000046   758          END;
000050   759        STATEMENT(LEV,LEV,PRLEV + 1);
000053   760        IF ENDSWITCH THEN
000055   761          BEGIN
000055   762            GENCOMM(0,0,0,0,0);
000060   763            WRITELN(XXXFILE,'END;');
000066   764          END;
000066   765    END
000066   766  END;   (*  ONECASE  *)
000100   767
000100   768
000100   769  BEGIN
000100   770    IF DD < SAVPOS THEN
000006   771      CLEARLINE;
000010   772    WHILE SYM <> OFSYM DO
000013   773      GETSYM(LEV,PRLEV);
000016   774    OUTLINE(CC-1,LEV,PRLEV,CTRNO);
000023   775    GETSYM(LEV,PRLEV);
000026   776    ONECASE(LEV + 1,PRLEV + 1);
000032   777    WHILE SYM = SEMICOLON DO
000035   778      BEGIN
000035   779        GETSYM(LEV,PRLEV);
000037   780        ONECASE(LEV +1,PRLEV +1);
000043   781      END;
000044   782    IF SYM = OTHERWISESYM THEN
000046   783      BEGIN
000046   784        IF DD < SAVPOS THEN
000050   785          CLEARLINE;
000052   786        OUTLINE(CC-1,LEV,PRLEV,CTRNO);
000057   787        CTRNO := CTRNO + 1;
000061   788        WRITEKOUNTER;
000063   789        GETSYM(LEV,PRLEV);
000066   790        STATEMENT(CASELEV,LEV+1,PRLEV+1);
000072   791        WHILE SYM IN [SEMICOLON] + STATBEGSYM DO
000076   792          BEGIN
000076   793            IF SYM = SEMICOLON THEN
000100   794              GETSYM(LEV,PRLEV);
000103   795            STATEMENT(CASELEV,LEV +1,PRLEV +1);
000107   796          END;
000110   797      END;
000110   798    IF SYM = ENDSYM THEN
000113   799      BEGIN
```

```
000113   800        SAVCTR := CTRNO;
000114   801        CTRNO := 0;
000115   802        GETSYM (LEV,PRLEV);
000120   803        IF SYM = SEMICOLON THEN
000123   804          BEGIN
000123   805            OUTLINE(CC-1,LEV,PRLEV,CTRNO);
000127   806            DD := CC
000127   807          END
000131   808        ELSE
000132   809          BEGIN
000132   810            CLEARLINE;
000134   811            GENCOMM(0,0,0,0,0);
000137   812            WRITELN(XXXFILE,';')
000144   813          END;
000145   814        CTRNO := SAVCTR;
000147   815        IF LEV <> CASELEV THEN
000151   816          BEGIN
000151   817        CTRNO := CTRNO + 1;
000153   818            WRITEKOUNTER;
000154   819          END
000154   820      END
000154   821    ELSE
000155   822
000155   823  END;    (*  CASESTATEMENT  *)
000157   824
000157   825
000157   826  PROCEDURE LABELSTATEMENT;
000002   827
000002   828  BEGIN
000002   829    GETSYM(LEV,PRLEV);
000007   830    CTRNO := CTRNO + 1;
000011   831    OUTLINE(CC-1,LEV,PRLEV,CTRNO);
000015   832    WRITEKOUNTER;
000017   833    SYM := SEMICOLON
000017   834  END;
000023   835
000023   836
000023   837  PROCEDURE GOTOSTATEMENT;
000002   838
000002   839  BEGIN
000002   840    WHILE NOT (SYM IN [ENDSYM, SEMICOLON, UNTILSYM, ELSESYM]) DO
000007   841      GETSYM(LEV,PRLEV);
000013   842    IF DD < SAVPOS THEN
000015   843      IF SYM <> SEMICOLON THEN
000017   844        CLEARLINE
000017   845      ELSE
000022   846        OUTLINE(CC-1,LEV,PRLEV,CTRNO);
```

```
000027   847    IF SYM <> SEMICOLON THEN
000032   848      BEGIN
000032   849        GENCOMM(0,00,0,0,0);
000035   850        WRITELN(XXXFILE,';')
000042   851      END;
000043   852    IF SYM <> ELSESYM THEN
000046   853      BEGIN
000046   854        CTRNO := CTRNO + 1;
000050   855        WRITEKOUNTER
000050   856      END
000051   857 END;    (*  GOTOSTATEMENT *)
000054   858
000054   859
000054   860 BEGIN  (* STATEMENT *)
000054   861   IF SYM IN STATBEGSYM + [IDENT] + [INTSYM]
000006   862     THEN
000010   863       CASE SYM OF
000011   864       IDENT : BEGIN
000011   865                 ASSIGNMENT;
000013   866               END;
000014   867       BEGINSYM : BEGINSTATEMENT;
000017   868       IFSYM : IFSTATEMENT;
000022   869       CASESYM : CASESTATEMENT;
000025   870       WHILESYM, FORSYM, WITHSYM : DOSTATEMENT;
000030   871       REPEATSYM : REPEATSTATEMENT;
000033   872       GOTOSYM : GOTOSTATEMENT;
000036   873       INTSYM : LABELSTATEMENT
000036   874       END; (* CASE *)
000060   875 END;    (*  STATEMENT *)
000072   876
000072   877 PROCEDURE CHECKPAREN;
000002   878
000002   879 BEGIN
000002   880   GETSYM(1,1);
000006   881   WHILE SYM <> RPAREN DO
000010   882     BEGIN
000010   883       IF SYM = LPAREN THEN
000012   884         CHECKPAREN;
000014   885       GETSYM(1,1);
000016   886     END
000016   887 END;      (* CHECKPAREN *)
000021   888
000021   889
000021   890 BEGIN  (* BLOCK *)
000021   891   REPEAT
000004   892     WHILE NOT(SYM IN [PROCSYM,FUNCSYM,BEGINSYM,EXTERNSYM,FORTRANSYM,FORWARDSYM]) DO
000007   893       GETSYM(1,1);
```

```
000012   894      IF NOT(SYM IN [FORTRANSYM,EXTERNSYM,FORWARDSYM]) THEN
000015   895        BEGIN
000015   896          GENCOMM(0,0,99,0,0);
000020   897          WRITELN(XXXFILE);
000022   898          IF SYM IN [PROCSYM,FUNCSYM] THEN
000025   899            BEGIN
000025   900              GETSYM (1,1);       (*  GET SUBPROGRAM IDENTIFIER  *)
000027   901              GETSYM(1,1);
000031   902              IF SYM = LPAREN THEN
000034   903                  CHECKPAREN;
000035   904              BLOCK(LEVEL + 1);
000037   905              IF SYM IN [EXTERNSYM, FORTRANSYM, FORWARDSYM] THEN
000042   906                GETSYM(1,1);
000044   907            END
000044   908        END;
000044   909    UNTIL SYM IN [BEGINSYM,EXTERNSYM,FORTRANSYM,FORWARDSYM];
000047   910    IF SYM = BEGINSYM THEN
000051   911      BEGIN
000051   912        SUPPRESS := FALSE;
000053   913        IF DD < SAVPOS THEN
000055   914          OUTLINE(SAVPOS-1,1,1,0);
000057   915        CTRNO := SAVCTR;
000061   916        STATEMENT(0,1,1);
000063   917        SAVCTR := CTRNO;
000065   918        CTRNO := 0;
000066   919        IF SYM <> PERIOD THEN
000070   920          BEGIN
000070   921            GETSYM(1,1);
000072   922            IF SYM = BEGINSYM THEN
000075   923              BEGIN
000075   924                GENCOMM(0,0,99,0,0);
000100   925                WRITELN(XXXFILE)
000100   926              END
000102   927          END;
000102   928        SUPPRESS := TRUE;
000104   929    END;
000104   930 END;  (*  BLOCK  *)
000112   931
000112   932
000112   933
000112   934 BEGIN
000112   935    INITIALIZE (STATBEGSYM, CONSTBEGSYM, KEY, KSYM, LL, CC, LINESIN, LINESOUT, CTRNO, SAVCTR,
000052   936                ENDOFLIN, CH);
000055   937    WRITEINIT;
000056   938    BLOCK(0);
000060   939 99 : WRITEFINAL;
000061   940 END.
```

COMPILER-ESTIMATED 'W' OPTION = 005241B.

D.1 PROFILE OF GROUP-COUNT PROGRAM

```
USER LINE NO.  PROFILE LINE NO.  LEV  NO OF TIMES EXECUTED   * * * PROGRAM STATEMENTS * * *
     1              1            1      PROGRAM GROUPCOUNT (DATA,RES,INPUT/,OUTPUT
     1              2            1      );
     2              3            1      TYPE
     3              4            1         STRING = RECORD
     4              5            1              CASE BOOLEAN OF
     5              6            1                 TRUE : (ST : PACKED ARRAY [1..30] OF CHAR);
     6              7            1                 FALSE : (A1, A2, A3 : ALFA);
     7              8            1              END;
     8              9            1      VAR
     9             10            1         RES, DATA : TEXT;
    10             11            1         SIZE, K, CLASS, WS, CV : INTEGER;
    11             12            1         WORD : STRING;


    13             13            1      PROCEDURE READNEXT (VAR W : STRING; VAR I : INTEGER);
    14             14            1      VAR
    15             15            1         C : CHAR;
    16             16            1         J : INTEGER;


    17             17            1   3711  BEGIN
    18             18            1     )     FOR J := 1 TO 30 DO
    18             19            2   111330    W.ST [J] := ' ';
    19             20            1   3711    WRITELN(RES);
    20             21            1     )     IF NOT EOF (DATA)
    20             22            2    3711     THEN
    21             23            2       )       REPEAT
    21             24            3      13021      READ (DATA,C);
    22             25            3          )      IF C IN ['A'..'Z','.']
    22             26            4        9263       THEN
    22             27            4          )          WRITE (RES,C)
    23             28            2    3711       UNTIL (EOF (DATA)) OR (C = '.');
    24             29            1   3711    IF NOT EOF (DATA)
    24             30            2     3710    THEN
    25             31            2       )       REPEAT
    25             32            3      27869      READ (DATA,C);
    26             33            3          )      IF C = ' '
    26             34            4        24159      THEN
    26             35            4          )          WRITE (RES,C)
    27             36            2     3710       UNTIL (EOF (DATA)) OR (C <> ' ');
    28             37            1   3711    I := 1;
    29             38            1     )     WHILE (NOT EOF (DATA)) AND (NOT EOLN (DATA)) DO
    30             39            2    36814    BEGIN
    31             40            2       )       IF C IN ['A'..'Z']
    31             41            3      18395      THEN
    32             42            3          )          BEGIN
    33             43            3          )            W.ST [I] := C;
    33             44            3          )            I := I + 1;
```

```
USER PROFILE  LEV NO OF    *  *  *  PROGRAM  STATEMENTS  *  *  *
LINE  LINE        TIMES
NO.   NO.         EXECUTED

  34    45   3            )        END;
  35    46   2     36814         READ (DATA,C);
  36    47   2          )     END;
  37    48   1     3711     IF C IN ['A'..'Z']
  37    49   2        105      THEN
  38    50   2          )        W.ST [I] := C
  39    51   2        3606     ELSE
  40    52   2          )        I := I - 1;
  41    53   1     3711   END;


  43    54   1            FUNCTION EXPON (X, Y : INTEGER) : INTEGER;
  44    55   1            VAR
  45    56   1               I, Z : INTEGER;


  46    57   1     22181  BEGIN
  47    58   1          )     Z := 1;
  48    59   1          )     FOR I := 1 TO Y DO
  48    60   2       30255       Z := Z * X;
  49    61   1     22181     IF Y = 0
  49    62   2        6940      THEN
  49    63   2          )        EXPON := 1
  49    64   2        6241     ELSE
  49    65   2          )        EXPON := Z;
  50    66   1     22181  END;


  52    67   1            FUNCTION CVALUE (W : STRING; S : INTEGER) : INTEGER;
  53    68   1            VAR
  54    69   1               I, K : INTEGER;
  55    70   1               J : ARRAY [1..30] OF INTEGER;


  56    71   1     3661   BEGIN
  57    72   1          )     FOR K := 1 TO 30 DO
  57    73   2     109830        J [K] := 0;
  58    74   1     3661      WRITE (RES,W.A1,W.A2,W.A3);
  59    75   1          )     FOR I := 1 TO S DO
  60    76   2      18305       BEGIN
  61    77   2          )        IF (W.ST [I] IN ['A','E','I','O','U','Y'])
  61    78   3        6954          THEN
  62    79   3             )           BEGIN
  63    80   3             )              WRITE (RES,'V');
  64    81   3             )              FOR K := S DOWNTO (S - I + 1) DO
  65    82   4          22138                J [K] := J [K] + EXPON (2, (S - I));
  66    83   3          6954           END
  67    84   3         11351        ELSE
```

```
USER PROFILE  LEV NO OF    *  *  *  PROGRAM  STATEMENTS  *  *  *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

  68    85   3          )        WRITE (RES,'C');
  69    86   2      18305      END;
  70    87   1       3661    FOR K := (S + 1) TO 30 DO
  70    88   2      91525      WRITE (RES,' ');
  71    89   1       3661    WRITELN (RES,J [S]);
  72    90   1          )    FOR K := (S - 1) DOWNTO 1 DO
  72    91   2      14644      WRITE (RES,J [K] :8);
  73    92   1       3661    WRITELN (RES);
  74    93   1          )    CVALUE := J [S];
  75    94   1          )  END;


  77    95   1          1  BEGIN
  78    96   1          )    LINELIMIT (OUTPUT,-1);
  79    97   1          )    LINELIMIT (RES,-1);
  80    98   1          )    REWRITE(RES);
  81    99   1          )    REWRITE(OUTPUT);
  82   100   1          )    RESET (DATA);
  83   101   1          )    WRITELN (' ENTER WORD SIZE');
  84   102   1          )    READ (SIZE);
  85   103   1          )    WRITELN (RES,'1    SIZE = ',SIZE:2);
  86   104   1          )    REPEAT
  86   105   2            1    IF NOT EOF (DATA)
  86   106   3              1    THEN
  86   107   3              )      READNEXT (WORD,WS)
  86   108   1          1    UNTIL (WS = SIZE) OR (EOF (DATA));
  87   109   1          )    CV := CVALUE (WORD, SIZE);
  88   110   1          )    REPEAT
  89   111   2            1    CLASS := CV;
  90   112   2            )    WHILE ((CLASS < EXPON (2, SIZE)) AND (NOT EOF (DATA))) DO
  91   113   3             42    BEGIN
  92   114   3              )      K := 0;
  93   115   3              )      WHILE (NOT EOF (DATA)) AND (CV = CLASS) DO
  94   116   4             3660      BEGIN
  95   117   4                )        K := K + 1;
  96   118   4                )        REPEAT
  96   119   5                  3710     READNEXT (WORD,WS)
  96   120   4             3660         UNTIL (WS = SIZE) OR (EOF (DATA));
  97   121   4                )        CV := CVALUE (WORD, SIZE);
  98   122   4                )      END;
  99   123   3             42      IF K > 0
  99   124   4               42      THEN
 100   125   4                )        BEGIN
 101   126   4                )          WRITELN (' CLASS = ',CLASS:8,' COUNT = ',K:6);
 102   127   4                )        END;
 103   128   3             42      CLASS := CV (* CLASS + 1 *);
 104   129   3              )    END;
 105   130   1          1    UNTIL EOF (DATA);
```

```
USER PROFILE  LEV NO OF    *  *  *  PROGRAM  STATEMENTS  *  *  *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 106   131   1        )    WRITELN (' RESULTS IN LOCAL FILE RES');
 107   132   1        )  END.
```

D.2 PROFILE OF PROFILE

```
USER PROFILE  LEV NO OF    *  *  *  PROGRAM  STATEMENTS  *  *  *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

   1     1   1          PROGRAM MONITOR (PROG,XXXFILE,OUTPUT,XXXAAA,XXXCOMM
   1     2   1          );
   1     3   1
   2     4   1          (*$P+*)
   3     5   1          LABEL
   4     6   1            99;
   6     7   1          CONST
   7     8   1            ALNG = 10;  (* NO. OF SIGNIFICANT CHARS IN IDENTIFIER *)
   8     9   1            ASTERICK = '*';
   9    10   1            BLANK = ' ';
  10    11   1            COMMA = ',';
  11    12   1            EQUALSGN = '=';
  12    13   1            LLNG = 121; (* INPUT LINE LENGTH *)
  13    14   1            NKW = 21;   (* NO OF KEY WORDS *)
  15    15   1          TYPE
  16    16   1            SYMBOL = (IDENT,INTSYM,RPAREN,SEMICOLON,PROGSYM,FUNCSYM,PROCSYM,PERIOD,STRING,BECOMES,
  17    17   1                      BEGINSYM,IFSYM,CASESYM,REPEATSYM,WHILESYM,FORSYM,ENDSYM,LPAREN,
  18    18   1                      ELSESYM,UNTILSYM,DOSYM,THENSYM,GOTOSYM,WITHSYM,
  19    19   1                      FORTRANSYM,EXTERNSYM,FORWARDSYM,OTHERWISESYM,OTHER,REALSYM,COLON,OFSYM,
  20    20   1                      CHARCON, PLUS, MINUS);
  21    21   1            SYMSET = SET OF SYMBOL;
  22    22   1            KEYTYPE = ARRAY [ 1 .. NKW ] OF ALFA;
  23    23   1            KSYMTYPE = ARRAY [ 1 .. NKW ] OF SYMBOL;
  25    24   1            LINETYPE = ARRAY [ 1 .. LLNG] OF CHAR;
  26    25   1            LETTTYPE = 'A'..'Z';
  27    26   1            DIGTYPE = '0'..'9';
  29    27   1          VAR
  30    28   1            (* FILES *)
  31    29   1            XXXFILE,          (* EDITED SOURCE FILE *)
  32    30   1            XXXCOMM,          (* CONTROL FILE FOR XXXFILE - 1:1 CORRESPONDENCE *)
  33    31   1            XXXAAA,           (* ADDITIONAL SOURCE CODE TO ADD TO HOST PROGRAM *)
  34    32   1            PROG :            (* ORIGINAL PROGRAM *)
  35    33   1                   TEXT;
  37    34   1            (*  *)
  38    35   1            SAVPOS,    (* INDEX TO FIRST CHARACTER OF PRESENT IDENTIFIER *)
  39    36   1            DD,        (*  INDEX TO FIRST NON BLANK CHARACTER OF INPUT LINE TO BE PRINTED *)
  40    37   1            CC,        (* INDEX TO NEXT CHARACTER *)
  41    38   1            LL         (* INDEX TO LAST CHARACTER OF INPUT LINE *)
  42    39   1                : INTEGER;
  43    40   1            (*  *)
  44    41   1            SUPPRESS,                (* USED TO KEEP ORIGINAL SPACING OF DEFINITION AND DECLARATION PART OF PROGRAM *)
  45    42   1            ENDOFLIN : BOOLEAN;  (* CC = LL *)
  47    43   1            LETTERS : SET OF LETTTYPE;
  48    44   1            DIGITS : SET OF DIGTYPE;
  49    45   1            LINE : LINETYPE;
  50    46   1            ID, INTVAL, XXXPROGID : ALFA;
  51    47   1            CH : CHAR;
  52    48   1            SYM : SYMBOL;
```

```
USER LINE NO.  PROFILE LINE NO.  LEV  NO OF TIMES EXECUTED   * * * PROGRAM STATEMENTS * * *
    54             49            1                    KEY : KEYTYPE;
    55             50            1                    KSYM : KSYMTYPE;
    57             51            1                    LINESIN, LINESOUT, CTRNO, SAVCTR : INTEGER;
    58             52            1                    CONSTBEGSYM, STATBEGSYM : SYMSET;


    60             53            1                  PROCEDURE INITIALIZE (VAR STATBEGSYM, CONSTBEGSYM : SYMSET;
    61             54            1                                        VAR KEY : KEYTYPE;
    62             55            1                                        VAR KSYM : KSYMTYPE;
    63             56            1                                        VAR LL, CC : INTEGER;
    64             57            1                                        VAR LINESIN, LINESOUT, CTRNO, SAVCTR : INTEGER;
    65             58            1                                        VAR ENDOFLIN:BOOLEAN;
    66             59            1                                        VAR CH : CHAR);


    68             60            1          1  BEGIN
    69             61            1          )    RESET (PROG);
    70             62            1          )    REWRITE (XXXFILE);
    71             63            1          )    REWRITE (XXXCOMM);
    72             64            1          )    STATBEGSYM := [BEGINSYM, IFSYM, CASESYM, WHILESYM, FORSYM, WITHSYM,
    73             65            1          )    REPEATSYM, GOTOSYM];
    74             66            1          )    CONSTBEGSYM := [PLUS, MINUS, INTSYM, REALSYM, CHARCON, IDENT];
    75             67            1          )    KEY [  1] := 'BEGIN     ';
    76             68            1          )    KEY [  2] := 'CASE      ';
    77             69            1          )    KEY [  3] := 'DO        ';
    78             70            1          )    KEY [  4] := 'ELSE      ';
    79             71            1          )    KEY [  5] := 'END       ';
    80             72            1          )    KEY [  6] := 'EXTERN    ';
    81             73            1          )    KEY [  7] := 'FORTRAN   ';
    82             74            1          )    KEY [  8] := 'FORWARD   ';
    83             75            1          )    KEY [  9] := 'FOR       ';
    84             76            1          )    KEY [ 10] := 'FUNCTION  ';
    85             77            1          )    KEY [ 11] := 'GOTO      ';
    86             78            1          )    KEY [ 12] := 'IF        ';
    87             79            1          )    KEY [ 13] := 'OF        ';
    88             80            1          )    KEY [ 14] := 'OTHERWISE ';
    89             81            1          )    KEY [ 15] := 'PROCEDURE ';
    90             82            1          )    KEY [ 16] := 'PROGRAM   ';
    91             83            1          )    KEY [ 17] := 'REPEAT    ';
    92             84            1          )    KEY [ 18] := 'THEN      ';
    93             85            1          )    KEY [ 19] := 'UNTIL     ';
    94             86            1          )    KEY [ 20] := 'WHILE     ';
    95             87            1          )    KEY [ 21] := 'WITH      ';
    96             88            1          )    KSYM [  1] := BEGINSYM;
    97             89            1          )    KSYM [  2] := CASESYM;
    98             90            1          )    KSYM [  3] := DOSYM;
    99             91            1          )    KSYM [  4] := ELSESYM;
   100             92            1          )    KSYM [  5] := ENDSYM;
```

```
USER PROFILE      NO OF    * * *  PROGRAM  STATEMENTS  * * *
LINE  LINE        TIMES
NO.   NO.         ECUTED

 101    93   1          )    KSYM [  6] := EXTERNSYM;
 102    94   1          )    KSYM [  7] := FORTRANSYM;
 103    95   1          )    KSYM [  8] := FORWARDSYM;
 104    96   1          )    KSYM [  9] := FORSYM;
 105    97   1          )    KSYM [ 10] := FUNCSYM;
 106    98   1          )    KSYM [ 11] := GOTOSYM;
 107    99   1          )    KSYM [ 12] := IFSYM;
 108   100   1          )    KSYM [ 13] := OFSYM;
 109   101   1          )    KSYM [ 14] := OTHERWISESYM;
 110   102   1          )    KSYM [ 15] := PROCSYM;
 111   103   1          )    KSYM [ 16] := PROGSYM;
 112   104   1          )    KSYM [ 17] := REPEATSYM;
 113   105   1          )    KSYM [ 18] := THENSYM;
 114   106   1          )    KSYM [ 19] := UNTILSYM;
 115   107   1          )    KSYM [ 20] := WHILESYM;
 116   108   1          )    KSYM [ 21] := WITHSYM;
 119   109   1          )    LL := 0;
 120   110   1          )    CC := 0;
 121   111   1          )    CH := BLANK;
 122   112   1          )    LINESIN := 0;
 123   113   1          )    LINESOUT := 0;
 124   114   1          )    CTRNO := 0;
 125   115   1          )    SAVCTR := 0;
 126   116   1          )    LETTERS := [ 'A' .. 'Z'];
 127   117   1          )    DIGITS := [ '0' .. '9'];
 130   118   1          )    ENDOFLIN := FALSE;
 131   119   1          ) END;

 136   120   1              PROCEDURE GENCOMM (LINESIN, LINESOUT, LEV, PRLEV, CTRNO : INTEGER);

 139   121   1     1581 BEGIN
 140   122   1          )   WRITE (XXXCOMM,LINESIN:5,LINESOUT:6,LEV:3,PRLEV:3,CTRNO:5);
 141   123   1          )   WRITELN(XXXCOMM)
 142   124   1          ) END;

 144   125   1              PROCEDURE OUTLINE (N, LEV, PRLEV, CTRNO : INTEGER);
 147   126   1                (*  GLOBAL VARIABLES USED      LINE, LINESIN, SUPPRESS
 148   127   1                                       MODIFIED  OD, LINESOUT  *)
 150   128   1              VAR
 151   129   1                I : INTEGER;

 153   130   1     1730 BEGIN
 154   131   1          )   IF NOT SUPPRESS
 154   132   2       1607     THEN
```

```
USER PROFILE  LEV NO OF    *  *  *  PROGRAM  STATEMENTS  *  *  *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 155   133   2          )      WHILE (LINE[DD] = BLANK) AND (DD <= N) DO
 156   134   3        385        DD := DD + 1;
 157   135   1       1730    IF N >= DD
 158   136   2        982      THEN
 159   137   2          )        BEGIN
 160   138   2          )          LINESOUT := LINESOUT + 1;
 161   139   2          )          GENCOMM (LINESIN, LINESOUT, LEV, PRLEV, CTRNO);
 162   140   2          )          FOR I := DD TO N DO
 163   141   3      15777            WRITE (XXXFILE, LINE[I]);
 164   142   2        982          WRITELN (XXXFILE);
 165   143   2          )          DD := N + 1
 166   144   2          )        END
 167   145   1       1730  END;
 167   146   1             (* OUTLINE *)


 169   147   1             PROCEDURE GETSYM (LEV, PRLEV : INTEGER ) ;
 170   148   1             (*
 171   149   1                    GLOBAL VARIABLES ACCESSED  CTRNO, LINESOUT
 172   150   1                                       MODIFIED  ID, INTVAL, SYM, SAVPOS   *)
 175   151   1             LABEL
 176   152   1               1;
 178   153   1             VAR
 179   154   1               I, J, K : INTEGER;
 180   155   1               COMMFIN : BOOLEAN;
 181   156   1               STRFIN : BOOLEAN;
 182   157   1               STORE : INTEGER;
 183   158   1               COMMSWT : BOOLEAN;


 185   159   1             PROCEDURE NEXTCH;
 187   160   1             (*     GLOBAL VARIABLES MODIFIED CC, LL, DD, LINESIN, ENDOFLIN, LINE, CH  *)


 190   161   1           16709  BEGIN
 191   162   1               )    IF CC = LL
 192   163   2              849     THEN
 193   164   2                )       BEGIN
 194   165   2                )         WHILE CC = LL DO
 195   166   3              939           BEGIN
 196   167   3                 )             LINESIN := LINESIN + 1;
 197   168   3                 )             ENDOFLIN := FALSE;
 198   169   3                 )             IF EOF (PROG)
 199   170   4                    0           THEN
 200   171   4                     )            BEGIN
 201   172   4                     )              GOTO 99
 202   173   4                    0             END;
 203   174   3              939              LL := 0;
```

```
USER PROFILE  LEV NO OF   *  *  *  PROGRAM  STATEMENTS  *  *  *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 204   175   3          )          CC := 1;
 205   176   3          )          WHILE NOT EOLN(PROG) DO
 206   177   4      21808            BEGIN
 207   178   4          )              LL := LL + 1;
 208   179   4          )              READ (PROG, LINE[LL])
 209   180   4          )            END;
 210   181   3        939          LL := LL + 1;
 211   182   3          )          READ  (PROG, LINE[LL]);
 212   183   3          )          WHILE (LINE [CC] = BLANK) AND (CC <> LL) DO
 213   184   4       5948            CC := CC + 1;
 214   185   3        939          IF SUPPRESS
 214   186   4        146            THEN
 215   187   4          )              DD := 1
 216   188   4        793            ELSE
 217   189   4          )              DD := CC;
 218   190   3        939          IF CC <> LL
 219   191   4        849            THEN
 220   192   4          )              CC := CC - 1;
 221   193   3        939        END
 222   194   2      849        END;
 223   195   1    16709    CC := CC + 1;
 224   196   1        )    CH := LINE [CC];
 225   197   1        )    IF CC = LL
 225   198   2      849      THEN
 225   199   2        )        ENDOFLIN := TRUE;
 226   200   1    16709  END;
 226   201   1          (* NEXTCH *)


 228   202   1          PROCEDURE READSCALE;


 230   203   1        0  BEGIN
 231   204   1        )    NEXTCH;
 232   205   1        )    IF CH IN [ '+','-' ]
 232   206   2          0    THEN
 233   207   2          )      NEXTCH;
 234   208   1        0    WHILE CH IN DIGITS DO
 235   209   2          0    NEXTCH;
 236   210   1        0  END;


 239   211   1     4319  BEGIN
 239   212   1           (* GETSYM *)
 240   213   1     4360    1 :
 240   214   1        )    WHILE CH = BLANK DO
 241   215   2       3132    BEGIN
 242   216   2          )      WHILE  ( CH = BLANK) AND (NOT ENDOFLIN ) DO
```

USER LINE NO.	PROFILE LINE NO.	LEV	NO OF TIMES EXECUTED	* * * PROGRAM STATEMENTS * * *
243	217	3	2887	NEXTCH;
244	218	2	3132	IF ENDOFLIN
244	219	3	844	THEN
245	220	3	)	BEGIN
246	221	3	)	OUTLINE (CC-1,LEV,RRLEV,CTRNO);
247	222	3	)	ENDOFLIN := FALSE;
248	223	3	)	END;
249	224	2	3132	END;
250	225	1	4360	SAVPOS := CC;
251	226	1	)	SYM := OTHER;
252	227	1	)	IF (CH IN LETTERS) OR (CH IN DIGITS) OR (CH IN ['''','(',')','+','-','.',':','%'])
253	228	1	)	OR (CH = ';')
253	229	2	3600	THEN
254	230	2	)	CASE CH OF
255	231	3	)	'A','B','C','D','E','F','G','H',
256	232	3	)	'I','J','K','L','M','N','O','P',
257	233	3	)	'Q','R','S','T','U','V','W','X',
258	234	3	1836	'Y','Z'
259	235	3	)	BEGIN
260	236	3	)	K := 0;
261	237	3	)	ID := ' ';
262	238	3	)	REPEAT
263	239	4	8982	IF K < ALNG
263	240	5	8896	THEN
264	241	5	)	BEGIN
265	242	5	)	K := K + 1;
266	243	5	)	ID[K] := CH
267	244	5	)	END;
268	245	4	8982	NEXTCH;
269	246	3	1836	UNTIL NOT (CH IN ['A' .. 'Z', '0' .. '9']);
270	247	3	)	IF K < ALNG
270	248	4	1769	THEN
271	249	4	)	BEGIN
272	250	4	)	I :=1;
273	251	4	)	J := NKW;
274	252	4	)	REPEAT
275	253	5	7536	K := (I + J) DIV 2;
276	254	5	)	IF ID <= KEY [K]
276	255	6	3926	THEN
277	256	6	)	J := K - 1;
278	257	5	7536	IF ID >= KEY [K]
278	258	6	4114	THEN
279	259	6	)	I := K + 1;
280	260	4	1769	UNTIL I > J;
281	261	4	)	IF (I - 1) > J
282	262	5	504	THEN
283	263	5	)	SYM := KSYM[K]
284	264	5	1265	ELSE

USER LINE NO.	PROFILE LINE NO.	LEV	NO OF TIMES EXECUTED	* * * PROGRAM STATEMENTS * * *
284	265	5	)	SYM := IDENT;
285	266	4	1769	END
286	267	4	67	ELSE
287	268	4	)	SYM := IDENT;
288	269	3	1836	END;
289	270	3	)	'1','2','3','4','5',
290	271	3	346	'6','7','8','9','0' :
291	272	3	)	BEGIN
292	273	3	)	K := 0;
293	274	3	)	SYM := INTSYM;
294	275	3	)	REPEAT
295	276	4	380	IF K < 5
295	277	5	380	THEN
296	278	5	)	BEGIN
297	279	5	)	K := K + 1;
298	280	5	)	INTVAL[K] := CH;
299	281	5	)	END;
300	282	4	380	NEXTCH;
301	283	3	346	UNTIL NOT (CH IN ['0' .. '9']);
302	284	3	)	IF CH = '.'
302	285	4	0	THEN
303	286	4	)	BEGIN
304	287	4	)	NEXTCH;
305	288	4	)	IF CH = '.'
306	289	5	0	THEN
307	290	5	)	CH := ':'
308	291	5	0	ELSE
309	292	5	)	BEGIN
310	293	5	)	SYM := REALSYM;
311	294	5	)	WHILE CH IN ['0' ..'9'] DO
312	295	6	0	NEXTCH;
313	296	5	0	IF CH = 'E'
313	297	6	0	THEN
314	298	6	)	READSCALE
315	299	5	0	END
316	300	4	0	END
317	301	4	346	ELSE
318	302	4	)	IF CH = 'E'
318	303	5	0	THEN
319	304	5	)	BEGIN
320	305	5	)	SYM :=REALSYM;
321	306	5	)	READSCALE
322	307	5	)	END
323	308	3	346	END;
324	309	3	151	'''' :
325	310	3	)	BEGIN
326	311	3	)	STRFIN := FALSE;
327	312	3	)	SYM := CHARCON;

```
USER PROFILE  LEV NO OF   * * *  PROGRAM  STATEMENTS  * * *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 328   313   3          )          K := 0;
 329   314   3          )          REPEAT
 330   315   4        156            NEXTCH;
 331   316   4          )            WHILE CH <> '''' DO
 332   317   5        786              BEGIN
 333   318   5          )                K := K + 1;
 334   319   5          )                NEXTCH;
 335   320   5          )              END;
 336   321   4        156            IF K > 1
 336   322   5         56              THEN
 337   323   5          )                SYM := STRING;
 338   324   4        156            NEXTCH;
 339   325   4          )            IF CH <> ''''
 339   326   5        145              THEN
 340   327   5          )                STRFIN := TRUE
 341   328   5         11              ELSE
 342   329   5          )                BEGIN
 343   330   5          )                  NEXTCH;
 344   331   5          )                  IF CH = ''''
 344   332   6            6                  THEN
 345   333   6          )                      BEGIN
 346   334   6          )                        STRFIN := TRUE;
 347   335   6          )                        NEXTCH
 348   336   6          )                      END;
 349   337   5         11                END
 350   338   3      151            UNTIL STRFIN;
 351   339   3        )          END;
 352   340   3      235        (* :
 353   341   3        )          BEGIN
 354   342   3        )            NEXTCH;
 355   343   3        )            IF CH = ASTERICK
 355   344   4         41            THEN
 356   345   4          )              BEGIN
 357   346   4          )                COMMFIN := FALSE;
 358   347   4          )                IF NOT SUPPRESS
 358   348   5         21                  THEN
 359   349   5          )                    WHILE LINE[DD] = ' ' DO
 360   350   6           56                     DD := DD + 1;
 361   351   4         41                 IF DD = (CC - 1)
 361   352   5          24                  THEN
 362   353   5          )                    COMMSWT := TRUE
 363   354   5         17                  ELSE
 364   355   5 *        )                    COMMSWT := FALSE;
 365   356   4         41                 NEXTCH;
 366   357   4          )                 REPEAT
 367   358   5          41                 WHILE (CH <> ASTERICK) AND NOT ENDOFLIN  DO
 368   359   6         874                   NEXTCH;
 369   360   5          41                 IF CH = ASTERICK
```

```
USER PROFILE  LEV NO OF   * * * * PROGRAM STATEMENTS * * * *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 369   361   6          38          THEN
 370   362   6           )            BEGIN
 371   363   6           )              NEXTCH;
 372   364   6           )              IF CH = ')'
 372   365   7            38              THEN
 373   366   7             )                COMMFIN := TRUE;
 374   367   6          38              END;
 375   368   4          41          UNTIL COMMFIN OR ENDOFLIN;
 376   369   4           )          IF ENDOFLIN
 376   370   5            3           THEN
 377   371   5             )           IE COMMSWT
 377   372   6              2            THEN
 378   373   6               )             OUTLINE (CC-1, LEV, PRLEV, 0)
 379   374   6              1            ELSE
 380   375   6               )             OUTLINE (CC-1, LEV, PRLEV, CTRNO);
 381   376   4          41          IF NOT COMMFIN
 381   377   5            3           THEN
 382   378   5             )            REPEAT
 383   379   6              4             NEXTCH;
 384   380   6               )            DD := 1;
 385   381   6               )            WHILE NOT ENDOFLIN AND NOT COMMFIN DO
 386   382   7                4             BEGIN
 387   383   7                )              WHILE NOT ENDOFLIN AND.(CH <> ASTERICK) DO
 388   384   8                116              NEXTCH;
 389   385   7                4              IF CH = ASTERICK
 389   386   8                  3              THEN
 390   387   8                  )                BEGIN
 391   388   8                  )                  NEXTCH;
 392   389   8                  )                  IF CH = ')'
 392   390   9                    3                  THEN
 393   391   9                    )                    COMMFIN := TRUE;
 394   392   8                  3                  END;
 395   393   7                4              IF ENDOFLIN
 395   394   8                  1              THEN
 396   395   8                  )                IF COMMSWT
 396   396   9                    1                THEN
 397   397   9                    )                  OUTLINE (CC-1, LEV, PRLEV, 0)
 398   398   9                    0                ELSE
 399   399   9                    )                  OUTLINE (CC-1, LEV, PRLEV, CTRNO);
 400   400   7                4              END;
 401   401   5            3           UNTIL COMMFIN;
 402   402   4          41          NEXTCH;
 403   403   4           )          WHILE (CH = ' ') AND (NOT ENDOFLIN) DO
 404   404   5            24          NEXTCH;
 405   405   4          41          IF ENDOFLIN
 405   406   5            41          THEN
 406   407   5             )            IF COMMSWT
 406   408   6              24            THEN
```

```
USER PROFILE LEV NO OF   * * *  PROGRAM  STATEMENTS  * * *
LINE  LINE       TIMES
NO.   NO.      EXECUTED

 407   409   6                )                    OUTLINE (CC-1, LEV, PRLEV, 0)
 408   410   6               17                  ELSE
 409   411   6                )                    OUTLINE (CC-1, LEV, PRLEV, CTRNO);
 410   412   4               41                  GOTO 1;
 411   413   4                0                END,
 412   414   4              194              ELSE
 413   415   4                )                SYM := LPAREN
 414   416   3              194            END;
 415   417   3              233          '%',':' :
 415   418   3                )            BEGIN
 416   419   3                )              NEXTCH;
 417   420   3                )              IF CH = '='
 417   421   4              166                THEN
 418   422   4                )                  BEGIN
 419   423   4                )                    SYM := BECOMES;
 420   424   4                )                    NEXTCH
 421   425   4                )                  END
 422   426   4               67                ELSE
 423   427   4                )                  SYM := COLON;
 424   428   3              233            END;
 426   429   3               12          '.' :
 426   430   3                )            BEGIN
 427   431   3                )              NEXTCH;
 428   432   3                )              IF CH = '.'
 428   433   4               11                THEN
 429   434   4                )                  BEGIN
 430   435   4                )                    SYM := COLON;
 431   436   4                )                    NEXTCH
 432   437   4                )                  END
 433   438   4                1                ELSE
 434   439   4                )                  SYM := PERIOD;
 435   440   3               12            END;
 437   441   3               66          '+' :
 437   442   3                )            BEGIN
 437   443   3                )              SYM := PLUS;
 437   444   3                )              NEXTCH
 437   445   3                )            END;
 438   446   3               34          '-' :
 438   447   3                )            BEGIN
 438   448   3                )              SYM := MINUS;
 438   449   3                )              NEXTCH
 438   450   3                )            END;
 439   451   3              493          ';' :
 439   452   3                )            BEGIN
 439   453   3                )              SYM := SEMICOLON;
 439   454   3                )              NEXTCH
 439   455   3                )            END;
 440   456   3              194          ) :
```

```
USER PROFILE. LEV NO OF    *  *  *  PROGRAM - STATEMENTS  *  *  *
LINE  LINE       TIMES
NO.   NO.        EXECUTED

 440   457   3          )        BEGIN
 440   458   3          )          SYM := RPAREN;
 440   459   3          )          NEXTCH
 440   460   3          )        END;
 441   461   2              END;
 441   462   1          (* CASE *)
 442   463   1   4319     IF SYM = OTHER
 442   464   2      760     THEN
 442   465   2          )     NEXTCH
 443   466   1   4319   END;
 443   467   1          (* GETSYMBOL *)


 445   468   1          PROCEDURE WRITEINIT ;


 447   469   1      1   BEGIN
 448   470   1      )     SUPPRESS := FALSE;
 449   471   1      )     GETSYM(1,1);
 450   472   1      )     GETSYM(1,1);
 451   473   1      )     XXXPROGID := ID;
 452   474   1      )     WHILE SYM <> RPAREN DO
 453   475   2       11     GETSYM(1,1);
 454   476   1      1     OUTLINE(CC-2,1,1,0);
 455   477   1      )     GENCOMM(0,0,0,0,0);
 456   478   1      )     WRITELN(XXXFILE, ',XXXFILE,XXXRR,XXXCOMM');
 457   479   1      )     WHILE SYM <> SEMICOLON DO
 458   480   2        1     GETSYM(1,1);
 459   481   1      1     OUTLINE(CC-1,1,1,0);
 460   482   1      )     GENCOMM(0,0,0,0,0);
 461   483   1      )     WRITELN(XXXFILE,'(*$I''XXXTRA''/''XXXAAA''*)');
 462   484   1      )     SUPPRESS := TRUE;
 463   485   1      )     WHILE NOT (SYM IN [BEGINSYM, FUNCSYM, PROCSYM]) DO
 464   486   2      242     GETSYM(1,1);
 465   487   1      1     GENCOMM(0,0,0,0,0);
 466   488   1      )     WRITELN(XXXFILE,'(*$X4*)');
 467   489   1      )     GENCOMM(0,0,0,0,0);
 468   490   1      )     WRITELN(XXXFILE,'PROCEDURE WPROFIL(XXXKOUNTER:DYNAMIC XXXTYPE;XXXCTRNO:INTEGER;');
 469   491   1      )     GENCOMM(0,0,0,0,0);
 470   492   1      )     WRITELN(XXXFILE,' XXXPROGID:ALFA;VAR XXXFILE, XXXRR, XXXCOMM : TEXT);');
 471   493   1      )     GENCOMM(0,0,0,0,0);
 472   494   1      )     WRITELN(XXXFILE,' EXTERN;');
 473   495   1      )     GENCOMM(0,0,0,0,0);
 474   496   1      )     WRITELN(XXXFILE,'(*$X= *)');
 475   497   1      )   END;


 477   498   1          PROCEDURE WRITEFINAL;
```

```
USER PROFILE LEV NO OF   * * * PROGRAM STATEMENTS * * *
LINE  LINE       TIMES
NO.   NO.      EXECUTED

 479   499   1       1  BEGIN
 480   500   1       )    CTRNO := SAVCTR;
 481   501   1       )    GENCOMM(0,0,0,0,0);
 482   502   1       )    WRITELN(XXXFILE,'XXXPROGID := ''',XXXPROGID,''';');
 483   503   1       )    GENCOMM(0,0,0,0,0);
 484   504   1       )    WRITELN(XXXFILE,'WPROFIL(XXXKOUNTER,',CTRNO:4,',',XXXPROGID,XXXFILE,XXXRR,XXXCOMM);');
 485   505   1       )    OUTLINE(CC-1,1,1,CTRNO);
 486   506   1       )    RESET(XXXFILE);
 487   507   1       )    REWRITE(XXXAAA);
 488   508   1       )    WRITELN(XXXAAA,'XXXTRA');
 489   509   1       )    WRITELN(XXXAAA,' TYPE XXXTYPE= ARRAY[0..', CTRNO:4, '] OF INTEGER;');
 490   510   1       )    WRITELN(XXXAAA,' CONST XXXCTRNO = ',CTRNO:4,';');
 491   511   1       )    WRITELN(XXXAAA,' VAR XXXKOUNTER : XXXTYPE;');
 492   512   1       )    WRITELN(XXXAAA,' XXXFILE, XXXRR, XXXCOMM : TEXT;');
 493   513   1       )    WRITELN(XXXAAA,' XXXPROGID : ALFA;');
 494   514   1       )    CTRNO := CTRNO + 1;
 495   515   1       )    WRITELN(XXXAAA,' VALUE XXXKOUNTER = (',CTRNO:4,' OF 0);');
 496   516   1       )    RESET(XXXAAA);
 497   517   1       )  END;


 499   518   1          PROCEDURE BLOCK (LEVEL : INTEGER);


 501   519   1          PROCEDURE STATEMENT(CASELEV,LEV,PRLEV : INTEGER);
 503   520   1          VAR
 504   521   1            I : INTEGER;


 506   522   1          PROCEDURE CLEARLINE;


 508   523   1     107  BEGIN
 509   524   1       )    OUTLINE (SAVPOS - 1, LEV, PRLEV, CTRNO);
 510   525   1       )    DD := SAVPOS
 511   526   1       )  END;


 513   527   1          PROCEDURE WRITEKOUNTER;


 515   528   1     320  BEGIN
 516   529   1       )    GENCOMM (0,0,0,0,0);
 517   530   1       )    WRITE (XXXFILE, 'XXXKOUNTER[', CTRNO:1, '] := XXXKOUNTER[',
 518   531   1       )    CTRNO:1, '] + 1;');
 519   532   1       )    WRITELN (XXXFILE);
 520   533   1       )  END;
```

```
USER PROFILE  LEV NO OF    * * *  PROGRAM  STATEMENTS  * * *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 522   534   1          PROCEDURE ASSIGNMENT;


 524   535   1     386  BEGIN
 525   536   1          )    WHILE NOT (SYM IN [SEMICOLON, ENDSYM, UNTILSYM, ELSESYM]) DO
 526   537   2     2197      GETSYM(LEV,PRLEV);
 527   538   1     386    IF SYM = SEMICOLON
 527   539   2      347      THEN
 528   540   2          )      BEGIN
 529   541   2          )        OUTLINE(CC-1, LEV, PRLEV, CTRNO);
 530   542   2          )        DD := CC
 531   543   2          )      END
 532   544   2       39     ELSE
 533   545   2          )      BEGIN
 534   546   2          )        OUTLINE(SAVPOS-1, LEV, PRLEV, CTRNO);
 535   547   2          )        DD := SAVPOS;
 536   548   2          )        GENCOMM(0,0,0,0,0);
 537   549   2          )        WRITELN(XXXFILE,';');
 538   550   2          )      END;
 539   551   1     386  END;
 539   552   1          (* ASSIGNMENT *)


 541   553   1          PROCEDURE IFSTATEMENT;
 543   554   1          VAR
 544   555   1            ENDSWITCH : BOOLEAN;


 546   556   1      87  BEGIN
 547   557   1       )    ENDSWITCH := FALSE;
 548   558   1       )    WHILE SYM <> THENSYM DO
 549   559   2      440     GETSYM(LEV,PRLEV);
 550   560   1      87    IF DD < SAVPOS
 550   561   2       79     THEN
 551   562   2       )       CLEARLINE;
 552   563   1      87    PRLEV := PRLEV + 1;
 553   564   1       )    OUTLINE(CC-1,LEV+1,PRLEV,CTRNO+1);
 554   565   1       )    DD := CC;
 555   566   1       )    GETSYM(LEV,PRLEV);
 556   567   1       )    IF SYM <> BEGINSYM
 556   568   2       40     THEN
 557   569   2          )      BEGIN
 558   570   2          )        CTRNO := CTRNO + 1;
 559   571   2          )        GENCOMM(0,0,0,0,0);
 560   572   2          )        WRITELN(XXXFILE,'BEGIN');
 561   573   2          )        WRITEKOUNTER;
 562   574   2          )        ENDSWITCH := TRUE
 563   575   2          )      END;
```

```
USER PROFILE  LEV NO OF   *  *  *  PROGRAM  STATEMENTS  *  *  *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 564   576   1      87    STATEMENT(CASELEV,LEV+1,PRLEV+1);
 565   577   1       )    IF ENDSWITCH
 565   578   2        40    THEN
 566   579   2         )      BEGIN
 567   580   2         )        ENDSWITCH := FALSE;
 568   581   2         )        GENCOMM(0,0,0,0,0);
 569   582   2         )        WRITE(XXXFILE,'END');
 570   583   2         )        IF SYM <> ELSESYM
 570   584   3          30        THEN
 571   585   3           )          BEGIN
 572   586   3           )            WRITELN(XXXFILE,';');
 573   587   3           )            CTRNO := CTRNO + 1;
 574   588   3           )            WRITEKOUNTER
 575   589   3           )          END
 576   590   3          10        ELSE
 577   591   3           )          WRITELN(XXXFILE);
 578   592   2        40      END;
 579   593   1      87    PRLEV := PRLEV -1;
 580   594   1       )    IF SYM = ELSESYM
 580   595   2        23    THEN
 581   596   2         )      BEGIN
 582   597   2         )        PRLEV := PRLEV + 1;
 583   598   2         )        OUTLINE(CC-1,LEV +1, PRLEV, CTRNO + 1);
 584   599   2         )        DD := CC;
 585   600   2         )        GETSYM (LEV,PRLEV);
 586   601   2         )        IF SYM <> BEGINSYM
 586   602   3          17        THEN
 587   603   3           )          BEGIN
 588   604   3           )            GENCOMM(0,0,0,0,0);
 589   605   3           )            WRITELN(XXXFILE,'BEGIN');
 590   606   3           )            CTRNO := CTRNO + 1;
 591   607   3           )            WRITEKOUNTER;
 592   608   3           )            ENDSWITCH := TRUE;
 593   609   3           )          END;
 595   610   2        23      STATEMENT(CASELEV,LEV + 1,PRLEV + 1);
 596   611   2         )      IF ENDSWITCH
 596   612   3          17        THEN
 597   613   3           )          BEGIN
 598   614   3           )            GENCOMM(0,0,0,0,0);
 599   615   3           )            WRITE(XXXFILE,'END');
 600   616   3           )            IF SYM <> ELSESYM
 600   617   4            17            THEN
 601   618   4             )              BEGIN
 602   619   4             )                WRITELN(XXXFILE,';');
 603   620   4             )                CTRNO := CTRNO + 1;
 604   621   4             )                WRITEKOUNTER;
 605   622   4             )              END
 606   623   4              0          ELSE
```

```
USER PROFILE  LEV NO OF    * * *  PROGRAM  STATEMENTS  * * *
LINE  LINE        TIMES
NO.   NO.        EXECUTED

 607   624   4          )              WRITELN(XXXFILE);
 608   625   3        17          END;
 609   626   2        23        PRLEV := PRLEV - 1;
 610   627   2         )      END
 611   628   1       87  END;
 611   629   1          (*  IFSTATEMENT  *)


 613   630   1          PROCEDURE BEGINSTATEMENT;


 615   631   1       97  BEGIN
 616   632   1        )    CTRNO := CTRNO + 1;
 617   633   1        )    OUTLINE(CC-1,LEV,PRLEV,CTRNO);
 618   634   1        )    DD := CC;
 619   635   1        )    WRITEKOUNTER;
 620   636   1        )    GETSYM(LEV,PRLEV);
 621   637   1        )    STATEMENT(CASELEV,LEV,PRLEV+1);
 622   638   1        )    WHILE SYM IN ([SEMICOLON] + STATBEGSYMS) DO
 623   639   2       370     BEGIN
 624   640   2         )       IF SYM = SEMICOLON
 624   641   3       370         THEN
 625   642   3          )          GETSYM(LEV,PRLEV);
 626   643   2       370       STATEMENT(CASELEV,LEV,PRLEV +1);
 627   644   2         )    END;
 628   645   1       97    IF SYM = ENDSYM
 628   646   2        97     THEN
 629   647   2         )       BEGIN
 630   648   2                 (* CTRNO := CTRNO + 1;
 631   649   2                  WRITEKOUNTER; *)
 632   650   2        )          GETSYM(LEV,PRLEV);
 633   651   2        )          IF SYM <> PERIOD
 633   652   3         96          THEN
 634   653   3          )            BEGIN
 635   654   3          )              IF SYM = SEMICOLON
 635   655   4           68              THEN
 636   656   4            )                BEGIN
 637   657   4            )                  OUTLINE(CC-1,LEV,PRLEV,CTRNO);
 638   658   4            )                  DD := CC;
 639   659   4            )                END
 640   660   4           28              ELSE
 641   661   4                             BEGIN
 642   662   4                               CLEARLINE;
 643   663   4                               IF SYM <> ELSESYM
 643   664   5                                 THEN
 644   665   5                                   BEGIN
 645   666   5                                     GENCOMM(0,0,0,0,0);
 646   667   5                                     WRITELN(XXXFILE,';');
```

```
USER PROFILE  LEV NO OF    *  *  *  PROGRAM  STATEMENTS  *  *  *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 647   668   5          )                    END;
 648   669   4       28                   END;
 649   670   3      .96                IF (SYM <> ELSESYM) AND(LEV <> 1) AND (LEV <> CASELEV)
 650   671   4       50                  THEN
 650   672   4        )                    BEGIN
 651   673   4        )                      CTRNO := CTRNO + 1;
 652   674   4        )                      WRITEKOUNTER;
 653   675   4        )                    END;
 654   676   3      96               END
 655   677   2     97       END
 656   678   2      0     ELSE
 658   679   1     97   END;
 658   680   1          (*  BEGINSTATEMENT  *)


 661   681   1          PROCEDURE REPEATSTATEMENT;


 663   682   1        7  BEGIN
 664   683   1        )    IF DD < SAVPOS
 664   684   2          0    THEN
 665   685   2          )      CLEARLINE;
 666   686   1        7    OUTLINE(CC-1,LEV,PRLEV,CTRNO);
 667   687   1        )    CTRNO := CTRNO + 1;
 668   688   1        )    WRITEKOUNTER;
 669   689   1        )    GETSYM(LEV,PRLEV);
 670   690   1        )    STATEMENT(CASELEV,LEV +1 ,PRLEV + 1);
 671   691   1        )    WHILE SYM IN [SEMICOLON] + STATBEGSYM DO
 672   692   2         18    BEGIN
 673   693   2         )      IF SYM = SEMICOLON
 673   694   3          18      THEN
 674   695   3           )        GETSYM(LEV,PRLEV);
 675   696   2         18      STATEMENT(CASELEV,LEV + 1, PRLEV + 1);
 676   697   2         )    END;
 677   698   1        7    IF SYM = UNTILSYM
 677   699   2          7    THEN
 678   700   2          )      BEGIN
 679   701   2          )        GETSYM(LEV,PRLEV);
 680   702   2          )        WHILE NOT (SYM IN [ENDSYM, SEMICOLON, ELSESYM, UNTILSYM]) DO
 681   703   3           43        GETSYM(LEV,PRLEV);
 682   704   2          7        IF SYM = SEMICOLON
 682   705   3            7        THEN
 683   706   3            )          OUTLINE(CC-1,LEV,PRLEV,CTRNO + 1)
 684   707   3            0        ELSE
 685   708   3            )          BEGIN
 686   709   3            )            OUTLINE(SAVPOS-1,LEV,PRLEV,CTRNO+1);
 687   710   3            )            GENCOMM(0,0,0,0,0);
 688   711   3            )            WRITELN(XXXFILE,':')
```

```
USER PROFILE  LEV NO OF    *  *  *  PROGRAM  STATEMENTS  *  *  *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 689   712   3           )          END;
 690   713   2         7          IF SYM <> ELSESYM
 690   714   3           7          THEN
 691   715   3           )            BEGIN
 692   716   3           )              CTRNO := CTRNO + 1;
 693   717   3           )              WRITEKOUNTER
 694   718   3           )            END
 695   719   2         7      END
 696   720   2         0    ELSE
 698   721   1       7  END;
 698   722   1          (*  REPEATSYM  *)

 701   723   1          PROCEDURE DOSTATEMENT;
 703   724   1          VAR
 704   725   1            ENDSWITCH : BOOLEAN;

 706   726   1       31  BEGIN
 707   727   1        )    ENDSWITCH := FALSE;
 708   728   1        )    WHILE SYM <> DOSYM DO
 709   729   2       266     GETSYM(LEV,PRLEV);
 710   730   1       31    OUTLINE(CC-1,LEV,PRLEV,CTRNO);
 711   731   1        )    DD := CC;
 712   732   1        )    GETSYM(LEV,PRLEV);
 713   733   1        )    IF SYM <> BEGINSYM
 713   734   2         21    THEN
 714   735   2           )     BEGIN
 715   736   2           )       GENCOMM(0,0,0,0,0);
 716   737   2           )       WRITELN(XXXFILE,'BEGIN');
 717   738   2           )       ENDSWITCH := TRUE;
 718   739   2           )       CTRNO := CTRNO + 1;
 719   740   2           )       WRITEKOUNTER;
 720   741   2           )     END;
 721   742   1       31    STATEMENT(CASELEV,LEV+1,PRLEV+1);
 722   743   1        )    IF ENDSWITCH
 722   744   2         21    THEN
 723   745   2           )     BEGIN
 724   746   2           )       GENCOMM(0,0,0,0,0);
 725   747   2           )       WRITELN(XXXFILE,'END;');
 726   748   2           )       IF SYM <> ELSESYM
 726   749   3            21       THEN
 727   750   3              )       BEGIN
 728   751   3              )         CTRNO := CTRNO + 1;
 729   752   3              )         WRITEKOUNTER;
 730   753   3              )       END
 731   754   2         21    END;
 732   755   1       31  END;
```

```
USER PROFILE  LEV NO OF    * * *  PROGRAM  STATEMENTS  * * *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 732   756   1          (*  DOSTATEMENT *)


 735   757   1          PROCEDURE CASESTATEMENT;


 738   758   1          PROCEDURE ONECASE(LEV,PRLEV:INTEGER);
 739   759   1          VAR
 740   760   1            ENDSWITCH : BOOLEAN;


 742   761   1      19  BEGIN
 743   762   1       )    IF SYM IN CONSTBEGSYM
 743   763   2        18    THEN
 744   764   2         )      BEGIN
 745   765   2         )        WHILE SYM <> COLON DO
 746   766   3          92        GETSYM(LEV,PRLEV);
 747   767   2        18          ENDSWITCH := FALSE;
 748   768   2         )          OUTLINE(CC-1,LEV,PRLEV,CTRNO + 1);
 749   769   2         )          GETSYM(LEV,PRLEV);
 750   770   2         )          IF SYM <> BEGINSYM
 750   771   3             7        THEN
 751   772   3                      BEGIN
 752   773   3                        GENCOMM(0,0,0,0,0);
 753   774   3                        WRITELN(XXXFILE,'BEGIN');
 754   775   3                        CTRNO := CTRNO + 1;
 755   776   3                        WRITEKOUNTER;
 756   777   3                        ENDSWITCH := TRUE
 757   778   3                      END;
 758   779   2        18          STATEMENT(LEV,LEV,PRLEV + 1);
 759   780   2         )          IF ENDSWITCH
 759   781   3             7        THEN
 760   782   3                      BEGIN
 761   783   3                        GENCOMM(0,0,0,0,0);
 762   784   3                        WRITELN(XXXFILE,'END;');
 763   785   3                      END;
 764   786   2        18        END
 765   787   1      19  END;
 765   788   1          (*  ONECASE  *)


 768   789   1       2  BEGIN
 769   790   1       )    IF DD < SAVPOS
 769   791   2         0    THEN
 770   792   2         )      CLEARLINE;
 771   793   1       2    WHILE SYM <> OFSYM DO
 772   794   2         4      GETSYM(LEV,PRLEV);
 773   795   1       2    OUTLINE(CC-1,LEV,PRLEV,CTRNO);
```

```
USER PROFILE  LEV NO OF - * * * PROGRAM STATEMENTS * * *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 774   796   1       )    GETSYM(LEV,PRLEV);
 775   797   1       )    ONECASE(LEV + 1,PRLEV + 1);
 776   798   1       )    WHILE SYM = SEMICOLON DO
 777   799   2        17    BEGIN
 778   800   2         )      GETSYM(LEV,PRLEV);
 779   801   2         )      ONECASE(LEV +1,PRLEV +1);
 780   802   2         )    END;
 781   803   1       2    IF SYM = OTHERWISESYM
 781   804   2         0    THEN
 782   805   2         )      BEGIN
 783   806   2         )        IF DD < SAVPOS
 783   807   3           0        THEN
 784   808   3           )          CLEARLINE;
 785   809   2         0        OUTLINE(CC-1,LEV,PRLEV,CTRNO);
 786   810   2         )        CTRNO := CTRNO + 1;
 787   811   2         )        WRITEKOUNTER;
 788   812   2         )        GETSYM(LEV,PRLEV);
 789   813   2         )        STATEMENT(CASELEV,LEV+1,PRLEV+1);
 790   814   2         )        WHILE SYM IN [SEMICOLON] + STATBEGSYM DO
 791   815   3           0        BEGIN
 792   816   3           )          IF SYM = SEMICOLON
 792   817   4             0          THEN
 793   818   4             )            GETSYM(LEV,PRLEV);
 794   819   3           0          STATEMENT(CASELEV,LEV +1,PRLEV +1);
 795   820   3           )        END;
 796   821   2         0      END;
 797   822   1       2    IF SYM = ENDSYM
 797   823   2         2    THEN
 798   824   2         )      BEGIN
 799   825   2         )        SAVCTR := CTRNO;
 800   826   2         )        CTRNO := 0;
 801   827   2         )        GETSYM (LEV,PRLEV);
 802   828   2         )        IF SYM = SEMICOLON
 802   829   3           2        THEN
 803   830   3           )          BEGIN
 804   831   3           )            OUTLINE(CC-1,LEV,PRLEV,CTRNO);
 805   832   3           )            DD := CC
 806   833   3           )          END
 807   834   3           0        ELSE
 808   835   3           )          BEGIN
 809   836   3           )            CLEARLINE;
 810   837   3           )            GENCOMM(0,0,0,0,0);
 811   838   3           )            WRITELN(XXXFILE,';')
 812   839   3           )          END;
 813   840   2         2        CTRNO := SAVCTR;
 814   841   2         )        IF LEV <> CASELEV
 814   842   3           2        THEN
 815   843   3           )          BEGIN
```

```
USER PROFILE  LEV NO OF   * * *  PROGRAM  STATEMENTS  * * *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 816   844   3          )           CTRNO := CTRNO + 1;
 817   845   3          )          WRITEKOUNTER;
 818   846   3          )        END;
 819   847   2        2      END
 820   848   2        0    ELSE
 822   849   1      2  END;
 822   850   1         (*  CASESTATEMENT *)


 825   851   1         PROCEDURE LABELSTATEMENT;


 827   852   1      2  BEGIN
 828   853   1      )    GETSYM(LEV,PRLEV);
 829   854   1      )    CTRNO := CTRNO + 1;
 830   855   1      )    OUTLINE(CC-1,LEV,PRLEV,CTRNO);
 831   856   1      )    WRITEKOUNTER;
 832   857   1      )    SYM := SEMICOLON
 833   858   1      )  END;


 836   859   1         PROCEDURE GOTOSTATEMENT;


 838   860   1      2  BEGIN
 839   861   1      )    WHILE NOT (SYM IN [ENDSYM, SEMICOLON, UNTILSYM, ELSESYM]) DO
 840   862   2        4    GETSYM(LEV,PRLEV);
 841   863   1      2    IF DD < SAVPOS
 841   864   2        1    THEN
 842   865   2        )      IF SYM <> SEMICOLON
 842   866   3          0      THEN
 843   867   3          )        CLEARLINE
 844   868   3          1      ELSE
 845   869   3          )        OUTLINE(CC-1,LEV,PRLEV,CTRNO);
 846   870   1      2    IF SYM <> SEMICOLON
 846   871   2        1    THEN
 847   872   2        )      BEGIN
 848   873   2        )        GENCOMM(0,DD,0,0,0);
 849   874   2        )        WRITELN(XXXFILE,';')
 850   875   2        )      END;
 851   876   1      2    IF SYM <> ELSESYM
 851   877   2        2    THEN
 852   878   2        )      BEGIN
 853   879   2        )        CTRNO := CTRNO + 1;
 854   880   2        )        WRITEKOUNTER
 855   881   2        )      END
 856   882   1      2  END;
 856   883   1         (*  GOTOSTATEMENT *)
```

```
USER PROFILE  LEV NO OF   * * *  PROGRAM  STATEMENTS  * * *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 859   884   1     674  BEGIN
 859   885   1          (* STATEMENT *)
 860   886   1       )    IF SYM IN STATBEGSYM + [IDENT] + [INTSYM]
 861   887   2      614     THEN
 862   888   2       )        CASE SYM OF
 863   889   3      386         IDENT :
 863   890   3       )            BEGIN
 864   891   3       )              ASSIGNMENT;
 865   892   3       )            END;
 866   893   3       97          BEGINSYM :
 866   894   3       )            BEGINSTATEMENT;
 867   895   3       87          IFSYM :
 867   896   3       )            IFSTATEMENT;
 868   897   3        2          CASESYM :
 868   898   3       )            CASESTATEMENT;
 869   899   3       31          WHILESYM, FORSYM, WITHSYM :
 869   900   3       )            DOSTATEMENT;
 870   901   3        7          REPEATSYM :
 870   902   3       )            REPEATSTATEMENT;
 871   903   3        2          GOTOSYM :
 871   904   3       )            GOTOSTATEMENT;
 872   905   3        2          INTSYM :
 872   906   3       )            LABELSTATEMENT
 873   907   2                 END;
 873   908   1          (* CASE *)
 874   909   1      674  END;
 874   910   1          (*  STATEMENT *)


 876   911   1          PROCEDURE CHECKPAREN;


 878   912   1        7  BEGIN
 879   913   1       )     GETSYM(1,1);
 880   914   1       )     WHILE SYM <> RPAREN DO
 881   915   2        84     BEGIN
 882   916   2         )       IF SYM = LPAREN
 882   917   3           0       THEN
 883   918   3           )         CHECKPAREN;
 884   919   2        84       GETSYM(1,1);
 885   920   2         )     END
 886   921   1        7  END;
 886   922   1           (* CHECKPAREN *)


 889   923   1       23  BEGIN
 889   924   1           (* BLOCK *)
 890   925   1        )    REPEAT
```

```
USER PROFILE LEV NO OF    * * *  PROGRAM  STATEMENTS  * * *
LINE  LINE       TIMES
NO.   NO.        EXECUTED

 891   926   2        40    WHILE NOT(SYM IN [PROCSYM,FUNCSYM,BEGINSYM,EXTERNSYM,FORTRANSYM,FORWARDSYM]) DO
 892   927   3        82      GETSYM(1,1);
 893   928   2        40    IF NOT(SYM IN [FORTRANSYM,EXTERNSYM,FORWARDSYM])
 893   929   3          40    THEN
 894   930   3           )      BEGIN
 895   931   3           )        GENCOMM(0,0,99,0,0);
 896   932   3           )        WRITELN(XXXFILE);
 897   933   3           )        IF SYM IN [PROCSYM,FUNCSYM]
 897   934   4            22        THEN
 898   935   4             )          BEGIN
 899   936   4             )            GETSYM (1,1);
 899   937   4                          (*  GET SUBPROGRAM IDENTIFIER  *)
 900   938   4             )            GETSYM(1,1);
 901   939   4             )            IF SYM = LPAREN
 901   940   5                7           THEN
 902   941   5                )             CHECKPAREN;
 903   942   4            22              BLOCK(LEVEL + 1);
 904   943   4             )              IF SYM IN [EXTERNSYM, FORTRANSYM, FORWARDSYM]
 904   944   5                0             THEN
 905   945   5                )               GETSYM(1,1);
 906   946   4              22            END
 907   947   3          40      END;
 908   948   1       23    UNTIL SYM IN [BEGINSYM,EXTERNSYM,FORTRANSYM,FORWARDSYM];
 909   949   1        )    IF SYM = BEGINSYM
 909   950   2         23    THEN
 910   951   2          )    BEGIN
 911   952   2          )      SUPPRESS := FALSE;
 912   953   2          )      IF DD < SAVPOS
 912   954   3            0      THEN
 913   955   3             )       OUTLINE(SAVPOS-1,1,1,0);
 914   956   2         23       CTRNO := SAVCTR;
 915   957   2          )       STATEMENT(0,1,1);
 916   958   2          )       SAVCTR := CTRNO;
 917   959   2          )       CTRNO := 0;
 918   960   2          )       IF SYM <> PERIOD
 918   961   3            22      THEN
 919   962   3             )        BEGIN
 920   963   3             )          GETSYM(1,1);
 921   964   3             )          IF SYM = BEGINSYM
 921   965   4                5          THEN
 922   966   4                )            BEGIN
 923   967   4                )              GENCOMM(0,0,99,0,0);
 924   968   4                )              WRITELN(XXXFILE)
 925   969   4                )            END
 926   970   3            22        END;
 927   971   2         23       SUPPRESS := TRUE;
 928   972   2          )     END;
 929   973   1       23  END;
```

```
USER PROFILE  LEV NO OF    * * *  PROGRAM STATEMENTS  * * *
LINE  LINE        TIMES
NO.   NO.       EXECUTED

 929   974   1          (*  BLOCK  *)
                          *
 933   975   1       1  BEGIN
 934   976   1       )    INITIALIZE (STATBEGSYM, CONSTBEGSYM, KEY, KSYM, LL, CC, LINESIN, LINESOUT, CTRNO, SAVCTR,
 935   977   1       )    ENDOFLIN, CH);
 936   978   1       )    WRITEINIT;
 937   979   1       )    BLOCK(0);
 938   980   1       1    99 :
 938   981   1       )    WRITEFINAL;
 939   982   1       )  END.
```

E FROM EXECUTABLE SPECIFICATION TO SOURCE CODE

PROCESS 9 Procedure for rectangular layout; PASCAL; optimized; no user interface

MACHINE 6 Environment for rectangular layout; (PASCAL)

PROGRAM 6 Strategy for rectangular layout; optimized search strategy

```
1   Initialize process
    1   Read parameters

            THEN 1
                2  READLN(NSHT,SHTL,SHTW,NEXT)
                3  I:=1
                4  J:=1
                5  NPC:=0

                                        THEN CLUSTER 2

2   Find longest remaining unplaced piece
    2   This piece is rejected; Examine the next piece

            When  (PCP[I] OR PCR[I])                    IS TRUE
            AND   I<NPC                                 IS TRUE

            THEN 1  I:=I+1

                                        THEN CLUSTER 2

    3   No unplaced pieces remain on the list

            When  (PCP[I] OR PCR[I])                    IS TRUE
            AND   I<NPC                                 IS FALSE

            THEN 1  APL:=TRUE

                                        THEN CLUSTER 3

    4   Piece found

            When  (PCP[I] OR PCR[I])                    IS FALSE

            THEN 1  LONGST:=I

                                        THEN CLUSTER 3

3   Place a piece on a sheet if possible
    5   Placing complete

            When  (APL OR NWSTCK)                       IS TRUE

            THEN 1
                2  FOR I:1 TO NPC DO WRITELN('#';I:6;' LEN ';PCL[I]:6;' WID ';PCW[I]:6;' SHT # ';SH
                3  I:=1
                4  MATCUR:=0.

                                        THEN CLUSTER 10

    6   Piece is to be placed in lower left corner

            When  CPC                                   IS TRUE

            THEN 1  NEXT:=LONGST
                2  MAXL:=SHTL-XLLEFT
                3  MAXW:=SHTW-YLLEFT

                                        THEN CLUSTER 4

    11  Current rectangle full; sheet must be reduced

            When  (APL OR NWSTCK)                       IS FALSE
            AND   CPC                                   IS FALSE
            AND   FUP                                   IS TRUE
            AND   FACRS                                 IS TRUE

            THEN 1  I:=1
```

```
                                        THEN CLUSTER  5

17  Piece is to be placed across (horizontally)

        When  (APL OR NMSTCK)                              IS FALSE
        AND   CPC                                          IS FALSE
        AND   FUP                                          IS TRUE
        AND   FACRS                                        IS FALSE

        THEN 1  NEXT!=LONGST
             2  MAXL!=SHTL-XACRS
             3  MAXW!=SHTW-YACRS

                                        THEN CLUSTER  6

18  Piece is to be placed across (horizontally)

        When  (APL OR NMSTCK)                              IS FALSE
        AND   CPC                                          IS FALSE
        AND   FUP                                          IS FALSE
        AND   FACRS                                        IS FALSE
        AND   NACRS                                        IS TRUE

        THEN 1  NEXT!=LONGST
             2  MAXL!=SHTL-XACRS
             3  MAXW!=SHTW-YACRS

                                        THEN CLUSTER  6

23  Piece is to be placed up (vertically)

        When  (APL OR NMSTCK)                              IS FALSE
        AND   CPC                                          IS FALSE
        AND   FUP                                          IS FALSE
        AND   FACRS                                        IS TRUE

        THEN 1  NEXT!=LONGST
             2  MAXL!=SHTL-XUP
             3  MAXW!=SHTW-YUP

                                        THEN CLUSTER  7

24  Piece is to be placed up (vertically)

        When  (APL OR NMSTCK)                              IS FALSE
        AND   CPC                                          IS FALSE
        AND   FUP                                          IS FALSE
        AND   FACRS                                        IS FALSE
        AND   NACRS                                        IS FALSE

        THEN 1  NEXT!=LONGST
             2  MAXL!=SHTL-XUP
             3  MAXW!=SHTW-YUP

                                        THEN CLUSTER  7

4   Place a corner piece if possible

    7   Not this piece! look at next piece

            When  (PCW[NEXT]>MAXW  OR  PCL[NEXT]>MAXL  OR IS TRUE
                  PCR[NEXT] OR PCP[NEXT])
            AND   NEXT < NPC                               IS TRUE

            THEN 1  NEXT!=NEXT+1

                                        THEN CLUSTER  4

    8   Last sheet has been filled

            When  (PCW[NEXT]>MAXW  OR  PCL[NEXT]>MAXL  OR IS TRUE
                  PCR[NEXT] OR PCP[NEXT])
            AND   NEXT < NPC                               IS FALSE
            AND   CSHT = NSHT                              IS TRUE

            THEN 1  NMSTCK!=TRUE

                                        THEN CLUSTER  3

    9   This sheet is full! New sheet must be started

            When  (PCW[NEXT]>MAXW  OR  PCL[NEXT]>MAXL  OR IS TRUE
                  PCR[NEXT] OR PCP[NEXT])
            AND   NEXT < NPC                               IS FALSE
            AND   CSHT = NSHT                              IS FALSE
```

```
        THEN 1  CSHT!:=CSHT!+1
             2  XLLEFT!:=0
             3  YLLEFT!:=0
             4  MATUS[CSHT]!:=0

                                        THEN CLUSTER  3

10  Piece found! Place it in position

        When  (PCW[NEXT]>MAXW   OR    PCL[NEXT]>MAXL    OR IS FALSE
              PCR[NEXT] OR PCP[NEXT])

        THEN 1  PCP[NEXT]!=TRUE
             2  SHEETN[NEXT]!=CSHT
             3  PCX[NEXT]!=XLLEFT
             4  PCY[NEXT]!=YLLEFT
             5  XUP!=XLLEFT
             6  YUP!=YLLEFT+PCW[NEXT]
             7  XACRS!=XLLEFT+PCL[NEXT]
             8  YACRS!=YLLEFT
             9  YMAX!=YUP
            10  FUP!=FALSE
            11  FACRS!=FALSE
            12  CPC!=FALSE
            13  HACRS!=TRUE
            14  I!=LONGST
            15  MATUS[CSHT]!=MATUS[CSHT]+PCW[NEXT]*PCL[NEXT]
            16  NCOR!=NEXT
            17  NUP !=  0
            18  NAC!=0

                                        THEN CLUSTER  2

5   Reduce the size of the sheet

    13  No more pieces on this sheet! use last position

        When  NUP = 0                                       IS TRUE

        THEN 1  YLLEFT!=YMAX
             2  CPC!=TRUE
             3  I!=1

                                        THEN CLUSTER  3

    14  Piece marking left border found! use its position

        When  NUP = 0                                       IS FALSE
        AND   (PCY[PCUP[I]] + PCW[PCUP[I]] > YMAX)          IS TRUE

        THEN 1  XLLEFT=XLLEFT+PCL[PCUP[I]]
             2  YLLEFT!=YMAX
             3  CPC!=TRUE
             4  I!=1

                                        THEN CLUSTER  3

    15  Top piece placed vertically is below maximum Y position

        When  I<NUP                                         IS FALSE
        AND   NUP = 0                                       IS FALSE
        AND   (PCY[PCUP[I]] + PCW[PCUP[I]] > YMAX)          IS FALSE

        THEN 1  YLLEFT!=YMAX
             2  CPC!=TRUE
             3  I!=1

                                        THEN CLUSTER  3

    16  This piece is not at left edge! continue

        When  I<NUP                                         IS TRUE
        AND   NUP = 0                                       IS FALSE
        AND   (PCY[PCUP[I]] + PCW[PCUP[I]] > YMAX)          IS FALSE

        THEN 1  I!=I+1

                                        THEN CLUSTER  5

6   Place a piece across (horizontally) if possible

    19  Not this piece! look at next piece

        When  (PCW[NEXT]>MAXW   OR    PCL[NEXT]>MAXL    OR IS TRUE
              PCR[NEXT] OR PCP[NEXT])
        AND   NEXT < NPC                                    IS TRUE
```

```
        THEN 1  NEXT:=NEXT+1

                                        THEN CLUSTER  6

20  No more pieces may be placed across (horizontally)

        When  (PCW[NEXT]>MAXW   OR    PCL[NEXT]>MAXL    OR IS TRUE
              PCR[NEXT] OR PCP[NEXT])
        AND   NEXT < NPC                                IS FALSE

        THEN 1  FACRS:=TRUE

                                        THEN CLUSTER  3

21  Place piece and reset Y maximum (YMAX) value

        When  (PCW[NEXT]+YACRS) > YMAX                  IS TRUE
        AND   (PCW[NEXT]>MAXW   OR    PCL[NEXT]>MAXL    OR IS FALSE
              PCR[NFXT] OR PCP[NEXT])

        THEN 1  PCP[NEXT]:=TRUE
             2  SHEETN[NEXT]:=CSHT
             3  PCX[NEXT]:=XACRS
             4  PCY[NEXT]:=YACRS
             5  XACRS:=XACRS+PCL[NEXT]
             6  NACRS:=FALSE
             7  YMAX:=PCW[NEXT]+YACRS
             8  I:=LONGST
             9  MATUS[CSHT]:=MATUS[CSHT]+PCW[NFXT]*PCL[NEXT]
             9  NAC:=NAC+1
             10 PCACR[NAC]:=NEXT

                                        THEN CLUSTER  2

22  Place piece in position

        When  (PCW[NEXT]+YACRS) < YMAX                  IS FALSE
        AND   (PCW[NEXT]>MAXW   OR    PCL[NEXT]>MAXL    OR IS FALSE
              PCR[NEXT] OR PCP[NFXT])

        THEN 1  PCP[NEXT]:=TRUE
             2  SHEETN[NEXT]:=CSHT
             3  PCX[NEXT]:=XACRS
             4  PCY[NFXT]:=YACRS
             5  XACRS:=XACRS+PCL[NEXT]
             6  NACRS:=FALSE
             7  I:=LONGST
             8  MATUS[CSHT]:=MATUS[CSHT]+PCW[NEXT]*PCL[NEXT]
             8  NAC:=NAC+1
             9  PCACR[NAC]:=NEXT

                                        THEN CLUSTER  2

7   Place a piece up (vertically) if possible

25  Not this piece! look at next one

        When  (PCW[NEXT]>MAXW   OR    PCL[NEXT]>MAXL    OR IS TRUE
              PCR[NEXT] OR PCP[NEXT])
        AND   NEXT < NPC                                IS TRUE

        THEN 1  NEXT:=NEXT+1

                                        THEN CLUSTER  7

26  No more pieces may be placed up (vertically)

        When  (PCW[NEXT]>MAXW   OR    PCL[NEXT]>MAXL    OR IS TRUE
              PCR[NEXT] OR PCP[NEXT])
        AND   NEXT < NPC                                IS FALSE

        THEN 1  FUP:=TRUE

                                        THEN CLUSTER  3

27  Piece found! place it in position

        When  (PCW[NEXT]>MAXW   OR    PCL[NEXT]>MAXL    OR IS FALSE
              PCR[NEXT] OR PCP[NEXT])

        THEN 1  PCP[NEXT]:=TRUE
             2  SHEETN[NEXT]:=CSHT
             3  PCX[NEXT]:=XUP
             4  PCY[NEXT]:=YUP
             5  YUP:=YUP+PCW[NEXT]
             6  NACRS:=TRUE
```

```
        /  I1=LONGST
        7  MATUS[CSHT]1=MATUS[CSHT]1+PCW[NEXT]*PCL[NEXT]
        8  NUP1=NUP1+1
        9  PCUP[NUP]1=NEXT

                                        THEN CLUSTER  2
```

8 Read data

28 Read in bill of materials

```
When  J <= NENT                                   IS TRUE
AND   NSHT <= 0                                   IS FALSE
AND   NENT<=0                                     IS FALSE

THEN 1  READLN(PCLO,PCWO,NPCS)

                                        THEN CLUSTER  9
```

29 No stock available! terminate procedure

```
When  NSHT <= 0                                   IS TRUE
AND   NENT<=0                                     IS FALSE

THEN 1  NMSTCK1=TRUE
     2

                                        THEN CLUSTER  0
```

30 All materials read! initialize flags and objects

```
When  J <= NENT                                   IS FALSE
AND   NSHT <= 0                                   IS FALSE
AND   NFNT<=0                                     IS FALSE

THEN 1  NMSTCK1=FALSE
     2  APL1=FALSE
     3  FUP1=FALSE
     4  FACRS1=FALSE
     5  CPC1=TRUE
     6  XLLEFT1=0
     7  YLLEFT1=0
     9  LONGST1=1
     10 I1=LONGST
     11 CSHT1=1
     12 MATUS[CSHT]1=0

                                        THEN CLUSTER  2
```

33 No pieces to generate! exit

```
When  NENT<=0                                     IS TRUE


                                        THEN CLUSTER  0
```

9 Initialize objects

31 Piece within limits of stock material! store and initialize.

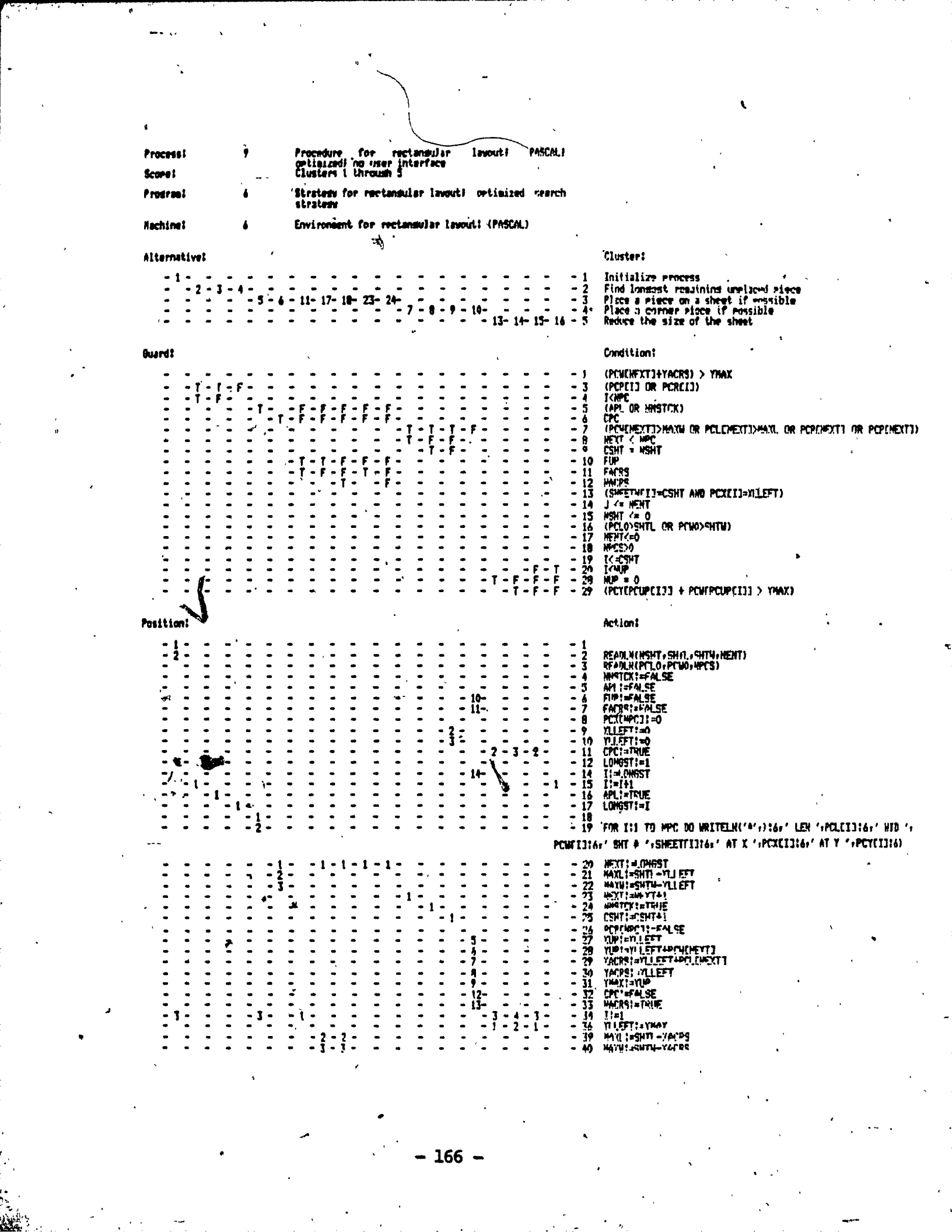
```
When  (PCLO>SHTL OR PCWO>SHTW)                     IS FALSE
AND   NPCS>0                                      IS TRUE

THEN 1  NPC1=NPC1+1
     2  PCY[NPC]1=0
     3  PCR[NPC]1=FALSE
     4  PCP[NPC]1=FALSE
     5  SHEETN[NPC]1=0
     6  PCX[NPC]1=0
     7  PCL[NPC]1=PCLO
     8  PCW[NPC]1=PCWO
     9  NPCS1=NPCS-1

                                        THEN CLUSTER  9
```

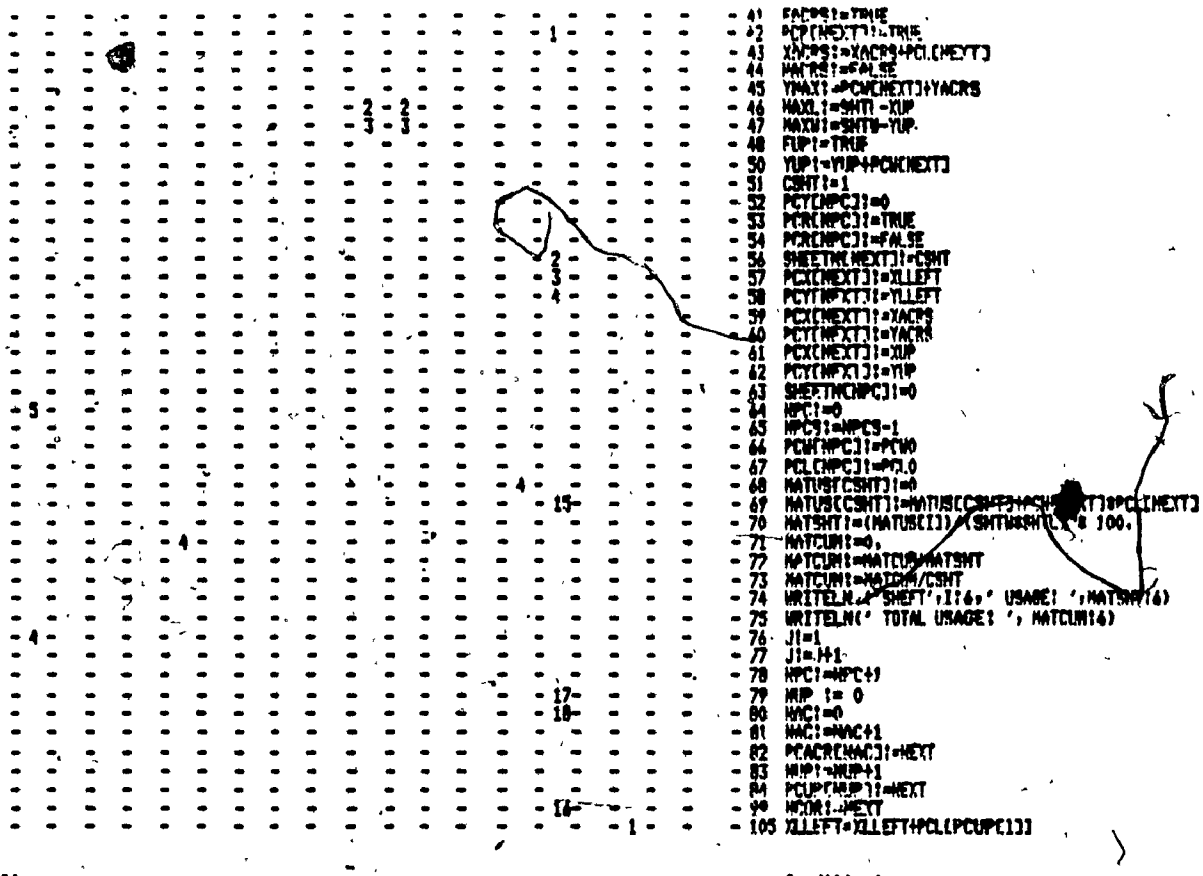
32 Piece larger than stock material! reject piece

```
When  (PCLO>SHTL OR PCWO>SHTW)                     IS TRUE
AND   NPCS>0                                      IS TRUE

THEN 1  NPC1=NPC1+1
     2  PCY[NPC]1=0
     3  PCR[NPC]1=TRUE
     4  PCX[NPC]1=0
     5  PCW[NPC]1=PCWO
     6  PCL[NPC]1=PCLO
```

```
            9  NPCS:=NPCS-1

                                              THEN CLUSTER  9

    34  Pieces recorded and checked

            When  NPCS>0                                    IS FALSE

            THEN 1  J:=J+1

                                              THEN CLUSTER  8

10  Generate output statistics

    35  Generate material wasted for used for each sheet

            When  I<=CSHT                                   IS TRUE

            THEN 1  MATSHT:=(MATUS[I])/(SHTW*SHTL) * 100.
                 2  WRITELN ('SHEET';I:4;' USAGE: ';MATSHT:6)
                 3  MATCUM:=MATCUM+MATSHT
                 4  I:=I+1

                                              THEN CLUSTER  10

    36  Generate average material used

            When  I<=CSHT                                   IS FALSE

            THEN 1  MATCUM:=MATCUM/CSHT
                 2  WRITELN(' TOTAL USAGE: '; MATCUM:6)

                                              THEN CLUSTER  0
```

Process: 9 Procedure for rectangular layout: PASCAL: optimized: no user interface
Score: Clusters 1 through 5
Program: 6 Strategy for rectangular layout: optimized search strategy
Machine: 6 Environment for rectangular layout: (PASCAL)





```
                                                                    41 FACRS:=TRUE
                                                                    42 PCR[NEXT]:=TRUE
                                                                    43 XACRS:=X[ACRS]+PCL[NEXT]
                                                                    44 MATRS:=FALSE
                                                                    45 YMAX:=PCW[NEXT]+YACRS
                                                                    46 MAXL:=SHTL-XUP
                                                                    47 MAXW:=SHTW-YUP.
                                                                    48 FUP:=TRUE
                                                                    50 YUP:=YUP+PCW[NEXT]
                                                                    51 CSHT:=1
                                                                    52 PCT[NPC]:=0
                                                                    53 PCR[NPC]:=TRUE
                                                                    54 PCR[NPC]:=FALSE
                                                                    56 SHEETN[NEXT]:=CSHT
                                                                    57 PCX[NEXT]:=XLLEFT
                                                                    58 PCY[NPCY]:=YLLEFT
                                                                    59 PCX[NEXT]:=XACRS
                                                                    60 PCY[NPCY]:=YACRS
                                                                    61 PCX[NEXT]:=XUP
                                                                    62 PCY[NPCY]:=YUP
                                                                    63 SHEETN[NPC]:=0
                                                                    64 NPC:=0
                                                                    65 NPCS:=NPCS-1
                                                                    66 PCW[NPC]:=PCW0
                                                                    67 PCL[NPC]:=PCL0
                                                                    68 MATUS[CSHT]:=0
                                                                    69 MATUS[CSHT]:=MATUS[CSHT]+PCW[NEXT]*PCL[NEXT]
                                                                    70 MATSHT:=(MATUS[I])/(SHTW*SHTL)*100.
                                                                    71 MATCUM:=0.
                                                                    72 MATCUM:=MATCUM+MATSHT
                                                                    73 MATCUM:=MATCUM/CSHT
                                                                    74 WRITELN,' SHEET',I:6,' USAGE: ',MATSHT:6)
                                                                    75 WRITELN(' TOTAL USAGE: ', MATCUM:6)
                                                                    76 J1:=1
                                                                    77 J1:=J+1
                                                                    78 NPC:=NPC+1
                                                                    79 NUP := 0
                                                                    80 NAC:=0
                                                                    81 NAC:=NAC+1
                                                                    82 PCACR[NAC]:=NEXT
                                                                    83 NUP:=NUP+1
                                                                    84 PCUP[NUP]:=NEXT
                                                                    90 NPCW:=NEXT
                                                                   105 XLLEFT:=XLLEFT+PCL[PCUPC[1]]
```

Goal:

Condition:

NEXT:

EXC:

SOS tabular display report (Courtesy GS Ltd.) 10/02/84

Process: 9 Procedure for rectangular layout: PASCAL: optimized no user interface
Scope: Clusters 4 through 10
Process: 6 Strategy for rectangular layout: optimized search strategy
Machine: 6 Environment for rectangular layout: (PASCAL)

Alternative:																						Cluster:
-	19	20	21	22	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	Place a piece across (horizontally) if possible
-	-	-	-	-	25	26	27	-	-	-	-	-	-	-	-	-	-	-	-	-	7	Place a piece up (vertically) if possible
-	-	-	-	-	-	-	-	28	29	30	33	-	-	-	-	-	-	-	-	-	8	Read data
-	-	-	-	-	-	-	-	-	-	-	-	31	32	34	-	-	-	-	-	-	9	Initialize objects
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	35	36	-	-	-	-	10	Generate output statistics

Guard:																						Condition:
-	-	-	T	-	F	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	(PCW[NEXT]+YACRS) > YMAX
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	3	(PCP[I] OR PCR[I])
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	I<NPC
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	5	(APL OR WHSTCK)
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	CPC
-	T	-	T	-	F	-	F	-	T	-	T	-	F	-	-	-	-	-	-	-	7	(PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCR[NEXT] OR PCP[NEXT])
-	T	-	F	-	-	-	-	T	-	F	-	-	-	-	-	-	-	-	-	-	8	NEXT < NPC
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	9	CSHT = NSHT
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	10	FUP
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	11	FACRS
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	12	NACRS
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	13	(SHEETNE[I]=CSHT AND PCY[I]=YLLEFT)
-	-	-	-	-	-	-	-	-	T	-	-	F	-	-	-	-	-	-	-	-	14	I <= NENT
-	-	-	-	-	-	-	-	-	F	-	T	-	F	-	-	-	-	-	-	-	15	NSHT <= 0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	F	-	T	-	-	-	-	16	(PCLO>SHTL OR PCW0>SHTW)
-	-	-	-	-	-	-	-	-	F	-	F	-	T	-	-	-	-	-	-	-	17	NENT<=0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	T	-	T	-	F	-	-	18	NPCS>0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	T	-	F	-	19	I<=CSHT
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	20	I<NUP
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	28	NUP = 0
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	29	(PCY[PCUP[I]] + PCW[PCUP[I]] > YMAX)

Position:																						Action:
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	2	READLN(NSHT,SHTL,SHTW,NENT)
-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	3	READLN(NPCL0,PCW0,NPCS)
-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	4	WHSTCK:=FALSE
-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	5	APL:=FALSE
-	-	-	-	-	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-	-	6	FUP:=FALSE
-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	-	-	-	7	FACRS:=FALSE
-	-	-	-	-	-	-	-	-	-	-	-	-	-	6	4	-	-	-	-	-	8	PEX[NPC]:=0
-	-	-	-	-	-	-	-	-	-	-	6	-	-	-	-	-	-	-	-	-	9	YLLEFT:=0
-	-	-	-	-	-	-	-	-	-	-	7	-	-	-	-	-	-	-	-	-	10	YLLEFT:=0
-	-	-	-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	11	CPC:=TRUE
-	-	-	-	-	-	-	-	-	-	-	9	-	-	-	-	-	-	-	-	-	12	LONGST:=1
-	-	-	8	-	7	-	-	7	-	-	10	-	-	-	-	-	-	-	-	-	14	I:=LONGST
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	15	I:=I+1
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	16	APL:=TRUE
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	17	LONGST:=I
-	-	-	-	-	-	-	-	-	-	-	2	-	-	-	-	-	-	-	-	-	18	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	19	FOR I:1 TO NPC DO WRITELN('P',I:6,' LEN ',PCL[I]:6,' WID ', PCW[I]:6,' SHT # ',SHEETE[I]:6,' AT X ',PCX[I]:6,' AT Y ',PCY[I]:6)
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	20	NEXT:=LONGST
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	21	MAXL:=SHTL-YLLEFT
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	22	MAXW:=SHTW-YLLEFT
-	1	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	23	NEXT:=NEXT+1
-	-	-	-	-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	24	WHSTCK:=TRUE
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	25	CSHT:=CSHT+1
-	-	-	-	-	-	-	-	-	-	-	-	-	-	4	-	-	-	-	-	-	26	PCP[NPC]:=FALSE
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	27	YUP:=YLLEFT
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	28	YUP:=YLLEFT+PCW[NEXT]
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	29	YACRS:=YLLEFT+PCL[NEXT]
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	30	YACRS:=YLLEFT
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	31	YMAX:=YUP
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	32	CPC:=FALSE
-	-	-	-	-	-	-	-	6	-	-	-	-	-	-	-	-	-	-	-	-	33	NACRS:=TRUE
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	34	I:=1
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	36	YLLEFT:=YMAX
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	39	MAXL:=SHTL-YACRS
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	40	MAXW:=SHTW-YACRS
-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	41	FACRS:=TRUE
-	-	-	1	-	1	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	42	PCP[NEXT]:=TRUE
-	-	-	3	-	3	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	43	XACRS:=XACRS+PCL[NEXT]
-	-	-	6	-	6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	44	NACRS:=[illegible] SF
-	-	-	7	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	45	YMAX:=PCW[NEXT]+YACRS
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	46	MAXL:=SHTL-XUP
-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	47	MAXW:=SHTW-YUP
-	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-	-	48	FUP:=TRUE
-	-	-	-	-	-	-	-	5	-	-	-	-	-	-	-	-	-	-	-	-	50	YUP:=YUP+PCW[NEXT]

```
-    -    -    -    -    -    -    -    -    - 11 -    -    -    -    -    -    - 51  CSHT:=1
-    -    -    -    -    -    -    -    -    -    -  2 -  2 -    -    -    -    - 52  PCY[NPC]:=0
-    -    -    -    -    -    -    -    -    -    -    -  3 -    -    -    -    - 53  PCR[NPC]:=TRUE
-    -    -    -    -    -    -    -    -    -    -  1 -    -    -    -    -    - 54  PCP[NPC]:=FALSE
-    -    -  2 -  2 -    -    -  2 -    -    -    -    -    -    -    -    -    - 56  SHEETN[NEXT]:=CSHT
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    - 57  PCX[NEXT]:=XLLEFT
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    - 58  PCY[NEXT]:=YLLEFT
-    -    -  3 -  3 -    -    -    -    -    -    -    -    -    -    -    -    - 59  PCX[NEXT]:=XACRS
-    -    -  4 -  4 -    -    -    -    -    -    -    -    -    -    -    -    - 60  PCY[NEXT]:=YACRS
-    -    -    -    -    -    -  3 -    -    -    -    -    -    -    -    -    - 61  PCX[NEXT]:=XUP
-    -    -    -    -    -    -  4 -    -    -    -    -    -    -    -    -    - 62  PCY[NEXT]:=YUP
-    -    -    -    -    -    -    -    -    -    -  5 -    -    -    -    -    - 63  SHEETN[NPC]:=0
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    - 64  NPC:=0
-    -    -    -    -    -    -    -    -    -    -  9 -  9 -    -    -    -    - 65  NPCS:=NPCS-1
-    -    -    -    -    -    -    -    -    -    -  8 -  5 -    -    -    -    - 66  PCW[NPC]:=PCW0
-    -    -    -    -    -    -    -    -    -    -  7 -  6 -    -    -    -    - 67  PCL[NPC]:=PCL0
-    -    -    -    -    -    -    -    -    - 12 -    -    -    -    -    -    - 68  MATUSE[CSHT]:=0
-    -    -  9 -  8 -    -    -  7 -    -    -    -    -    -    -    -    -    - 69  MATUSE[CSHT]:=MATUSE[CSHT]+PCW[NEXT]*PCL[NEXT]
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -  1 -    - 70  MATSHT:=(MATUSE[I])/(SHTW*SHTL) * 100.
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    - 71  MATCUM:=0.
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -  3 -    - 72  MATCUM:=MATCUM+MATSHT
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -  1 - 73  MATCUM:=MATCUM/CSHT
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -  2 -    - 74  WRITELN ('SHEET':I:6,' USAGE: ':MATSHT:6)
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -  2 - 75  WRITELN(' TOTAL USAGE: ', MATCUM:4)
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    - 76  J:=1
-    -    -    -    -    -    -    -    -    -    -    -    -    -  1 -    -    - 77  J:=J+1
-    -    -    -    -    -    -    -    -    -    -    -  1 -  1 -    -    -    - 78  NPC:=NPC+1
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    - 79  NUP := 0
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    - 80  NAC:=0
-    -    -  9 -  8 -    -    -    -    -    -    -    -    -    -    -    -    - 81  NAC:=NAC+1
-    -    - 10 -  9 -    -    -    -    -    -    -    -    -    -    -    -    - 82  PCACR[NAC]:=NEXT
-    -    -    -    -    -    -  8 -    -    -    -    -    -    -    -    -    - 83  NUP:=NUP+1
-    -    -    -    -    -    -  9 -    -    -    -    -    -    -    -    -    - 84  PCUP[NUP]:=NEXT
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    - 99  NCOR:=NEXT
-    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    - 105 XLLEFT=XLLEFT+PCL[PCUP[I]]

Goal:                                                                                 Condition:

NEXT:

     -  6 -  3 -  2 -  2 -  7 -  3 -  2 -  9 -  0 -  2 -  0 -  9 -  9 -  8 - 10 -  0 -
EXC:
     -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -    -
```

```
PROGRAM ALBAHD (INPUT,OUTPUT)
LABEL 10001 ;
LABEL 10002 ;
LABEL 10003 ;
LABEL 10004 ;
LABEL 10005 ;
LABEL 10006 ;
LABEL 10007 ;
LABEL 10008 ;
LABEL 10009 ;
LABEL 10010 ;
LABEL 50001 ;
LABEL 50002 ;
LABEL 50003 ;
LABEL 50004 ;
LABEL 50005 ;
LABEL 50006 ;
LABEL 50007 ;
LABEL 50008 ;
LABEL 50009 ;
LABEL 50010 ;
LABEL 50011 ;
LABEL 50012 ;
LABEL 50014 ;
LABEL 50015 ;
LABEL 50016 ;
LABEL 50017 ;
LABEL 50018 ;
LABEL 50019 ;
LABEL 50020 ;
LABEL 50021 ;
LABEL 50022 ;
LABEL 50023 ;
LABEL 50024 ;
LABEL 50025 ;
LABEL 50026 ;
LABEL 50027 ;
LABEL 50028 ;
LABEL 50029 ;
LABEL 50030 ;
LABEL 50031 ;
LABEL 50032 ;
LABEL 50033 ;
LABEL 50034 ;
LABEL 50035 ;
LABEL 50036 ;
LABEL 10000;
VAR
PCW0 : INTEGER;
PCX[1..500] : INTEGER;
PCL[1..500] : INTEGER;
PCWL[1..500] : INTEGER;
PCPL[1..500] : BOOLEAN;
PCDL[1..500] : BOOLEAN;
SHEETNC[1..500] : INTEGER;
PCYL[1..500] : INTEGER;
NPC : INTEGER;
NEXT: INTEGER;
LONGRT : INTEGER;
SHTL: INTEGER;
XLLEFT: INTEGER;
MAXL: INTEGER;
XUP: INTEGER;
XACRS: INTEGER;
XMIN: INTEGER;
I: INTEGER;
SHTW: INTEGER;
YLLEFT: INTEGER;
MAXW : INTEGER;
YACRS: INTEGER;
YMAX: INTEGER;
NSHT: INTEGER;
CSHT: INTEGER;
APL: BOOLEAN;
MASTCK: BOOLEAN;
CPC: BOOLEAN;
FUP: BOOLEAN;
FACRS: BOOLEAN;
MACRS: BOOLEAN;
PCL0 : INTEGER;
NPCS: INTEGER;
MATERC[1..500]: INTEGER;
MATCM: REAL;
MATSHT: REAL;
NEWT: INTEGER;
```

```
      J: INTEGER;
      YUP: INTEGER;
      K: INTEGER;
      NCOR: INTEGER;
      NUP: INTEGER;
      NAC:INTEGER;
      NNUM: INTEGER;
      NX: INTEGER;
      NY: INTEGER;
      PCUP[1..100]: INTEGER;
      PCACR[1..300] : INTEGER;
      ACRDIS: BOOLEAN;
      UPDTS: BOOLEAN;
      BEGIN
10001:
50001:
      ;
      READLN(NSHT,SHTL,SHTW,KENT) ;
      I:=1 ;
      J:=1 ;
      NPC:=0 ;
      GOTO 10008 ;
10002:
      IF( ( (PCP[I] OR PCR[I]) )
      AND ( I<NPC )
      ) THEN GOTO 50002 ;
      IF( ( (PCP[I] OR PCR[I]) )
      AND NOT ( I<NPC )
      ) THEN GOTO 50003 ;
      IF( NOT ( (PCP[I] OR PCR[I]) )
      ) THEN GOTO 50004 ;
50002:
      I:=I+1 ;
      GOTO 10002 ;
50003:
      APL:=TRUE ;
      GOTO 10003 ;
50004:
      LONGST:=I ;
      GOTO 10003 ;
10003:
      IF( ( (APL OR MMSTCK) )
      ) THEN GOTO 50005 ;
      IF( ( CPC )
      ) THEN GOTO 50006 ;
      IF( NOT ( (APL OR MMSTCK) )
      AND NOT ( CPC )
      AND ( FUP )
      AND ( FACRS )
      ) THEN GOTO 50011 ;
      IF( NOT ( (APL OR MMSTCK) )
      AND NOT ( CPC )
      AND ( FUP )
      AND NOT ( FACRS )
      ) THEN GOTO 50017 ;
      IF( NOT ( (APL OR MMSTCK) )
      AND NOT ( CPC )
      AND NOT ( FUP )
      AND NOT ( FACRS )
      AND ( MACRS )
      ) THEN GOTO 50018 ;
      IF( NOT ( (APL OR MMSTCK) )
      AND NOT ( CPC )
      AND NOT ( FUP )
      AND ( FACRS )
      ) THEN GOTO 50023 ;
      IF( NOT ( (APL OR MMSTCK) )
      AND NOT ( CPC )
      AND NOT ( FUP )
      AND NOT ( FACRS )
      AND NOT ( MACRS )
      ) THEN GOTO 50024 ;
50005:
      ;
      FOR I:1 TO NPC DO WRITELN('#',I:6,' LEN ',PCL[I]:6,' WID ',PCW[I]:6,' SHT # ',SHT[I]:6,' AT X ',PCX[I]:6,' AT Y ',PCY[I]:6) ;
      I:=1 ;
      MATCUM:=0. ;
      GOTO 10010 ;
50006:
      NEXT:=LONGST ;
      MAXL:=SHTL-YLLEFT ;
      MAXW:=SHTW-YLLEFT ;
      GOTO 10004 ;
50011:
      I:=1 ;
      GOTO 10005 ;
50017:
      NEXT:=LONGST ;
```

```
        MAXL!=SHTL-XACRS !
        MAXW!=SHTW-YACRS !
        GOTO 10006 !
50018!
        NEXT!=LONGST !
        MAXL!=SHTL-XACRS !
        MAXW!=SHTW-YACRS !
        GOTO 10006 !
50023!
        NEXT!=LONGST !
        MAXL!=SHTL-XUP !
        MAXW!=SHTW-YUP !
        GOTO 10007 !
50024!
        NEXT!=LONGST !
        MAXL!=SHTL-XUP !
        MAXW!=SHTW-YUP !
        GOTO 10007 !
10004!
        IF( ( (PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCR[NEXT] OR PCP[NEXT]) )
        AND ( NEXT < NPC )
        ) THEN GOTO 50007 !
        IF( ( (PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCR[NEXT] OR PCP[NEXT]) )
        AND NOT ( NEXT < NPC )
        AND ( CSHT = NSHT )
        ) THEN GOTO 50008 !
        IF( ( (PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCR[NEXT] OR PCP[NEXT]) )
        AND NOT ( CSHT = NSHT )
        AND NOT ( NEXT < NPC )
        ) THEN GOTO 50009 !
        IF( NOT ( (PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCR[NEXT] OR PCP[NEXT]) )
        ) THEN GOTO 50010 !
50007!
        NEXT!=NEXT+1 !
        GOTO 10004 !
50008!
        NHSTK!=TRUE !
        GOTO 10003 !
50009!
        CSHT!=CSHT+1 !
        XLLEFT!=0 !
        YLLEFT!=0 !
        NATUS[CSHT]!=0 !
        GOTO 10003 !
50010!
        PCP[NEXT]!=TRUE !
        SHEETNO[NEXT]!=CSHT !
        PCX[NEXT]!=XLLEFT !
        PCY[NEXT]!=YLLEFT !
        XUP!=XLLEFT !
        YUP!=YLLEFT+PCW[NEXT] !
        XACRS!=XLLEFT+PCL[NEXT] !
        YACRS!=YLLEFT !
        YMAX!=YUP !
        FUP!=FALSE !
        FACRS!=FALSE !
        CPC!=FALSE !
        MACRS!=TRUE !
        I!=LONGST !
        NATUS[CSHT]!=NATUS[CSHT]+PCW[NEXT]*PCL[NEXT] !
        NCOR!=NEXT !
        NUP != 0 !
        NAC!=0 !
        GOTO 10002 !
10005!
        IF( ( NUP = 0 )
        ) THEN GOTO 50013 !
        IF( NOT ( NUP = 0 )
        AND ( (PCY[PCUP[I]] + PCW[PCUP[I]] > YMAX) )
        ) THEN GOTO 50014 !
        IF( NOT ( NUP = 0 )
        AND NOT ( (PCY[PCUP[I]] + PCW[PCUP[I]] > YMAX) )
        AND NOT ( I<NUP )
        ) THEN GOTO 50015 !
        IF( NOT ( NUP = 0 )
        AND NOT ( (PCY[PCUP[I]] + PCW[PCUP[I]] > YMAX) )
        AND ( I<NUP )
        ) THEN GOTO 50016 !
50013!
        YLLEFT!=YMAX !
        CPC!=TRUE !
        I!=1 !
        GOTO 10003 !
50014!
        XLLEFT=XLLEFT+PCL[PCUP[I]] !
        YLLEFT!=YMAX !
        CPC!=TRUE !
        I!=1 !
```

```
      GOTO 10003 ;
50015:
      YLLEFT:=YMAX ;
      CPC:=TRUE ;
      I:=1 ;
      GOTO 10003 ;
50016:
      I:=I+1 ;
      GOTO 10005 ;
10006:
      IF( ( (PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCR[NEXT] OR PCP[NEXT]) )
      AND ( NEXT < NPC )
      ) THEN GOTO 50019 ;
      IF( ( (PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCP[NEXT] OR PCP[NEXT]) )
      AND NOT ( NEXT < NPC )
      ) THEN GOTO 50020 ;
      IF( ( (PCW[NEXT]+YACRS) > YMAX )
      AND NOT ( (PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCR[NEXT] OR PCP[NEXT]) )
      ) THEN GOTO 50021 ;
      IF( NOT ( (PCW[NEXT]+YACRS) > YMAX )
      AND NOT ( (PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCR[NEXT] OR PCP[NEXT]) )
      ) THEN GOTO 50022 ;
50019:
      NEXT:=NEXT+1 ;
      GOTO 10006 ;
50020:
      FACRS:=TRUE ;
      GOTO 10003 ;
50021:
      PCP[NEXT]:=TRUE ;
      SHEETH[NEXT]:=CSHT ;
      PCX[NEXT]:=XACRS ;
      PCY[NEXT]:=YACRS ;
      XACRS:=XACRS+PCL[NEXT] ;
      HACRS:=FALSE ;
      YMAX:=PCW[NEXT]+YACRS ;
      I:=LONGST ;
      MATUS[CSHT]:=MATUS[CSHT]+PCW[NEXT]*PCL[NEXT] ;
      NAC:=NAC+1 ;
      PCACR[NAC]:=NEXT ;
      GOTO 10002 ;
50022:
      PCP[NEXT]:=TRUE ;
      SHEETH[NEXT]:=CSHT ;
      PCX[NEXT]:=XACRS ;
      PCY[NEXT]:=YACRS ;
      XACRS:=XACRS+PCL[NEXT] ;
      HACRS:=FALSE ;
      I:=LONGST ;
      MATUS[CSHT]:=MATUS[CSHT]+PCW[NEXT]*PCL[NEXT] ;
      NAC:=NAC+1 ;
      PCACR[NAC]:=NEXT ;
      GOTO 10002 ;
10007:
      IF( ( (PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCR[NEXT] OR PCP[NEXT]) )
      AND ( NEXT < NPC )
      ) THEN GOTO 50025 ;
      IF( ( (PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCR[NEXT] OR PCP[NEXT]) )
      AND NOT ( NEXT < NPC )
      ) THEN GOTO 50026 ;
      IF( NOT ( (PCW[NEXT]>MAXW OR PCL[NEXT]>MAXL OR PCR[NEXT] OR PCP[NEXT]) )
      ) THEN GOTO 50027 ;
50025:
      NEXT:=NEXT+1 ;
      GOTO 10007 ;
50026:
      FUP:=TRUE ;
      GOTO 10003 ;
50027:
      PCP[NEXT]:=TRUE ;
      SHEETH[NEXT]:=CSHT ;
      PCX[NEXT]:=XUP ;
      PCY[NEXT]:=YUP ;
      YUP:=YUP+PCW[NEXT] ;
      HACRS:=TRUE ;
      I:=LONGST ;
      MATUS[CSHT]:=MATUS[CSHT]+PCW[NEXT]*PCL[NEXT] ;
      NUP:=NUP+1 ;
      PCUP[NUP]:=NEXT ;
      GOTO 10002 ;
10008:
      IF( I <= NENT )
      AND NOT ( NSHT <= 0 )
      AND NOT ( NENT<=0 )
      ) THEN GOTO 50028 ;
      IF( ( NSHT <= 0 )
      AND NOT ( NENT<=0 )
      ) THEN GOTO 50029 ;
```

```
      IF( NUI ( J <= NPCS )
      AND NOT ( NRHT <= 0 )
      AND NOT ( NRHT<=M) )
      ) THEN GOTO 50030 ;
      IF( ( NENT<=0 )
      ) THEN GOTO 50033 ;
50028:
      READLN(PCLO;PCWO;NPCS) ;
      GOTO 10009 ;
50029:
      WWSTCK:=TRUE ;
      ;
      GOTO 10000 ;
50030:
      WWSTCK:=FALSE ;
      APL:=FALSE ;
      FUP:=FALSE ;
      FACRS:=FALSE ;
      CPC:=TRUE ;
      XLLEFT:=0 ;
      YLLEFT:=0 ;
      LONGST:=0 ;
      I:=LONGST ;
      CSHT:=1 ;
      MATUSE[CSHT]:=0 ;
      GOTO 10002 ;
50033:
      GOTO 10000 ;
10009:
      IF( NOT ( (PCLO>SHTL OR PCWO>SHTW) )
      AND ( NPCS>0 )
      ) THEN GOTO 50031 ;
      IF( ( (PCLO>SHTL OR PCWO>SHTW) )
      AND ( NPCS>0 )
      ) THEN GOTO 50032 ;
      IF( NOT ( NPCS>0 )
      ) THEN GOTO 50034 ;
50031:
      NPC:=NPC+1 ;
      PCY[NPC]:=0 ;
      PCR[NPC]:=FALSE ;
      PCP[NPC]:=FALSE ;
      SHEETN[NPC]:=0 ;
      PCX[NPC]:=0 ;
      PCL[NPC]:=PCLO ;
      PCW[NPC]:=PCWO ;
      NPCS:=NPCS-1 ;
      GOTO 10009 ;
50032:
      NPC:=NPC+1 ;
      PCY[NPC]:=0 ;
      PCR[NPC]:=TRUE ;
      PCX[NPC]:=0 ;
      PCW[NPC]:=PCWO ;
      PCL[NPC]:=PCLO ;
      NPCS:=NPCS-1 ;
      GOTO 10009 ;
50034:
      J:=J+1 ;
      GOTO 10008 ;
10010:
      IF( ( I<=CSHT )
      ) THEN GOTO 50035 ;
      IF( NOT ( I<=CSHT )
      ) THEN GOTO 50036 ;
50035:
      MATSHT:=(MATUSE[I])/(SHTW*SHTL) * 100. ;
      WRITELN ('SHEET';I14;' USAGE! ';MATSHT16) ;
      MATCUM:=MATCUM+MATSHT ;
      I:=I+1 ;
      GOTO 10010 ;
50036:
      MATCUM:=MATCUM/CSHT ;
      WRITELN(' TOTAL USAGE! ', MATCUM16) ;
      GOTO 10000 ;
10000:END.
```

EXECUTION STATISTICS (ALTERNATIVES)

ALT #	1	EX->	1	TIME(S)	1.7286682E-03 PERCENT
ALT #	2	EX->	342	TIME(S)	0.5912045 PERCENT
ALT #	3	EX->	1	TIME(S)	1.7286682E-03 PERCENT
ALT #	4	EX->	343	TIME(S)	0.5929332 PERCENT
ALT #	5	EX->	1	TIME(S)	1.7286682E-03 PERCENT
ALT #	6	EX->	83	TIME(S)	0.1434795 PERCENT
ALT #	7	EX->	10348	TIME(S)	17.88826 PERCENT
ALT #	8	EX->	0	TIME(S)	0.0000000 PERCENT
ALT #	9	EX->	37	TIME(S)	6.3960724E-02 PERCENT
ALT #	10	EX->	46	TIME(S)	7.9518735E-02 PERCENT
ALT #	11	EX->	45	TIME(S)	7.7790067E-02 PERCENT
ALT #	12	EX->	0	TIME(S)	0.0000000 PERCENT
ALT #	13	EX->	27	TIME(S)	4.6674043E-02 PERCENT
ALT #	14	EX->	14	TIME(S)	2.4201356E-02 PERCENT
ALT #	15	EX->	4	TIME(S)	6.9146729E-03 PERCENT
ALT #	16	EX->	5	TIME(S)	8.6433413E-03 PERCENT
ALT #	17	EX->	207	TIME(S)	0.3578343 PERCENT
ALT #	18	EX->	71	TIME(S)	0.1227354 PERCENT
ALT #	19	EX->	26095	TIME(S)	45.10960 PERCENT
ALT #	20	EX->	45	TIME(S)	7.7790067E-02 PERCENT
ALT #	21	EX->	21	TIME(S)	3.6302034E-02 PERCENT
ALT #	22	EX->	212	TIME(S)	0.3664777 PERCENT
ALT #	23	EX->	47	TIME(S)	8.1247404E-02 PERCENT
ALT #	24	EX->	63	TIME(S)	0.1089061 PERCENT
ALT #	25	EX->	19255	TIME(S)	33.28551 PERCENT
ALT #	26	EX->	46	TIME(S)	7.9518735E-02 PERCENT
ALT #	27	EX->	64	TIME(S)	0.1106348 PERCENT
ALT #	28	EX->	21	TIME(S)	3.6302034E-02 PERCENT
ALT #	29	EX->	0	TIME(S)	0.0000000 PERCENT
ALT #	30	EX->	1	TIME(S)	1.7286682E-03 PERCENT
ALT #	31	EX->	343	TIME(S)	0.5929332 PERCENT
ALT #	32	EX->	0	TIME(S)	0.0000000 PERCENT
ALT #	33	EX->	0	TIME(S)	0.0000000 PERCENT
ALT #	34	EX->	21	TIME(S)	3.6302034E-02 PERCENT
ALT #	35	EX->	38	TIME(S)	6.5689392E-02 PERCENT
ALT #	36	EX->	1	TIME(S)	1.7286682E-03 PERCENT

TOTAL NUMBER OF ALTERNATIVES EXECUTED --> 57848

	EXECUTION		STATISTICS			(ACTIONNR)
ACT #	1	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	2	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	3	EX --->	21	TIME(S)	3.2131217E-02	PERCENT
ACT #	4	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	5	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	6	EX --->	47	TIME(S)	7.1912728E-02	PERCENT
ACT #	7	EX --->	47	TIME(S)	7.1912728E-02	PERCENT
ACT #	8	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	9	EX --->	38	TIME(S)	5.8142208E-02	PERCENT
ACT #	10	EX --->	38	TIME(S)	5.8142208E-02	PERCENT
ACT #	11	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	12	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	13	EX --->	0	TIME(S)	0.0000000	PERCENT
ACT #	14	EX --->	344	TIME(S)	0.5263399	PERCENT
ACT #	15	EX --->	385	TIME(S)	0.5890723	PERCENT
ACT #	16	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	17	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	18	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	19	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	20	EX --->	471	TIME(S)	0.7206573	PERCENT
ACT #	21	EX --->	83	TIME(S)	0.1269948	PERCENT
ACT #	22	EX --->	83	TIME(S)	0.1269948	PERCENT
ACT #	23	EX --->	55698	TIME(S)	85.22117	PERCENT
ACT #	24	EX --->	0	TIME(S)	0.0000000	PERCENT
ACT #	25	EX --->	37	TIME(S)	5.6612149E-02	PERCENT
ACT #	26	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	27	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	28	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	29	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	30	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	31	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	32	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	33	EX --->	110	TIME(S)	0.1683064	PERCENT
ACT #	34	EX --->	92	TIME(S)	0.1407653	PERCENT
ACT #	35	EX --->	0	TIME(S)	0.0000000	PERCENT
ACT #	36	EX --->	45	TIME(S)	6.8852611E-02	PERCENT
ACT #	37	EX --->	0	TIME(S)	0.0000000	PERCENT
ACT #	38	EX --->	0	TIME(S)	0.0000000	PERCENT
ACT #	39	EX --->	278	TIME(S)	0.4253561	PERCENT
ACT #	40	EX --->	278	TIME(S)	0.4253561	PERCENT
ACT #	41	EX --->	45	TIME(S)	6.8852611E-02	PERCENT
ACT #	42	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	43	EX --->	233	TIME(S)	0.3565035	PERCENT
ACT #	44	EX --->	233	TIME(S)	0.3565035	PERCENT
ACT #	45	EX --->	21	TIME(S)	3.2131217E-02	PERCENT
ACT #	46	EX --->	110	TIME(S)	0.1683064	PERCENT
ACT #	47	EX --->	110	TIME(S)	0.1683064	PERCENT
ACT #	48	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	49	EX --->	0	TIME(S)	0.0000000	PERCENT
ACT #	50	EX --->	64	TIME(S)	9.7923711E-02	PERCENT
ACT #	51	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	52	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	53	EX --->	0	TIME(S)	0.0000000	PERCENT
ACT #	54	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	55	EX --->	0	TIME(S)	0.0000000	PERCENT
ACT #	56	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	57	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	58	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	59	EX --->	233	TIME(S)	0.3565035	PERCENT
ACT #	60	EX --->	233	TIME(S)	0.3565035	PERCENT
ACT #	61	EX --->	64	TIME(S)	9.7923711E-02	PERCENT
ACT #	62	EX --->	64	TIME(S)	9.7923711E-02	PERCENT
ACT #	63	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	64	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	65	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	66	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	67	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	68	EX --->	38	TIME(S)	5.8142208E-02	PERCENT
ACT #	69	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	70	EX --->	38	TIME(S)	5.8142208E-02	PERCENT
ACT #	71	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	72	EX --->	38	TIME(S)	5.8142208E-02	PERCENT
ACT #	73	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	74	EX --->	38	TIME(S)	5.8142208E-02	PERCENT
ACT #	75	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	76	EX --->	1	TIME(S)	1.5300580E-03	PERCENT
ACT #	77	EX --->	21	TIME(S)	3.2131217E-02	PERCENT
ACT #	78	EX --->	343	TIME(S)	0.5248099	PERCENT
ACT #	79	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	80	EX --->	46	TIME(S)	7.0382670E-02	PERCENT
ACT #	81	EX --->	233	TIME(S)	0.3565035	PERCENT
ACT #	82	EX --->	233	TIME(S)	0.3565035	PERCENT
ACT #	83	EX --->	64	TIME(S)	9.7923711E-02	PERCENT
ACT #	84	EX --->	64	TIME(S)	9.7923711E-02	PERCENT
ACT #	85	EX --->	0	TIME(S)	0.0000000	PERCENT
ACT #	86	EX --->	0	TIME(S)	0.0000000	PERCENT

```
ACT #   87 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   88 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   89 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   90 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   91 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   92 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   93 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   94 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   95 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   96 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   97 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   98 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #   99 EX --->       46 TIME(S)  :  7.0182670E-02 PERCENT
ACT #  100 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #  101 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #  102 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #  103 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #  104 EX --->        0 TIME(S)  :  0.0000000   PERCENT
ACT #  105 EX --->       14 TIME(S)  :  2.1420812E-02 PERCENT
TOTAL NUMBER OF ACTIONS EXECUTED --->      65357
```