A UNITED STATES
DEPARTMENT OF
COMMERCE
PUBLICATION

NBS TECHNICAL NOTE 763

A Set of Debugging and Monitoring Facilities to Improve the Diagnostic Capabilities of a Compiler

U.S.
DEPARTMENT
OF
COMMERCE
National
Bureau
of
Standards

NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau consists of the Institute for Basic Standards, the Institute for Materials Research, the Institute for Applied Technology, the Center for Computer Sciences and Technology, and the Office for Information Programs.

THE INSTITUTE FOR BASIC STANDARDS provides the central basis within the United States of a complete and consistent system of physical measurement; coordinates that system with measurement systems of other nations; and furnishes essential services leading to accurate and uniform physical measurements throughout the Nation's scientific community, industry, and commerce. The Institute consists of a Center for Radiation Research, an Office of Measurement Services and the following divisions:

> Applied Mathematics — Electricity — Mechanics — Heat — Optical Physics — Linac Radiation [2] — Nuclear Radiation [2] — Applied Radiation [2] — Quantum Electronics [3] — Electromagnetics [3] — Time and Frequency [3] — Laboratory Astrophysics [3] — Cryogenics [3].

THE INSTITUTE FOR MATERIALS RESEARCH conducts materials research leading to improved methods of measurement, standards, and data on the properties of well-characterized materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; and develops, produces, and distributes standard reference materials. The Institute consists of the Office of Standard Reference Materials and the following divisions:

> Analytical Chemistry—Polymers—Metallurgy—Inorganic Materials—Reactor Radiation—Physical Chemistry.

THE INSTITUTE FOR APPLIED TECHNOLOGY provides technical services to promote the use of available technology and to facilitate technological innovation in industry and Government; cooperates with public and private organizations leading to the development of technological standards (including mandatory safety standards), codes and methods of test; and provides technical advice and services to Government agencies upon request. The Institute also monitors NBS engineering standards activities and provides liaison between NBS and national and international engineering standards bodies. The Institute consists of a Center for Building Technology and the following divisions and offices:

> Engineering and Product Standards—Weights and Measures—Invention and Innovation—Product Evaluation Technology—Electronic Technology—Technical Analysis—Measurement Engineering—Building Standards and Code Services[4]—Housing Technology[4]—Federal Building Technology[4]—Structures, Materials and Life Safety[4]—Building Environment[4]—Technical Evaluation and Application[4]—Fire Technology.

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides technical services designed to aid Government agencies in improving cost effectiveness in the conduct of their programs through the selection, acquisition, and effective utilization of automatic data processing equipment; and serves as the principal focus within the executive branch for the development of Federal standards for automatic data processing equipment, techniques, and computer languages. The Center consists of the following offices and divisions:

> Information Processing Standards—Computer Information—Computer Services —Systems Development—Information Processing Technology.

THE OFFICE FOR INFORMATION PROGRAMS promotes optimum dissemination and accessibility of scientific information generated within NBS and other agencies of the Federal Government; promotes the development of the National Standard Reference Data System and a system of information analysis centers dealing with the broader aspects of the National Measurement System; provides appropriate services to ensure that the NBS staff has optimum accessibility to the scientific information of the world, and directs the public information activities of the Bureau. The Office consists of the following organizational units:

> Office of Standard Reference Data—Office of Technical Information and Publications—Library—Office of International Relations.

---

[1] Headquarters and Laboratories at Gaithersburg, Maryland, unless otherwise noted; mailing address Washington, D.C. 20234.
[2] Part of the Center for Radiation Research.
[3] Located at Boulder, Colorado 80302.
[4] Part of the Center for Building Technology.

A Set of Debugging and Monitoring Facilities to Improve the Diagnostic Capabilities of a Compiler

Elizabeth Fong

Systems Development Division
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, D.C. 20234

NBS Technical Notes are designed to supplement the Bureau's regular publications program. They provide a means for making availahle scientific data that are of transient for limited interest. Technical Notes may he listed or referred to in the open literature.

U.S. DEPARTMENT OF COMMERCE, Frederick B. Dent, *Secretary*
NATIONAL BUREAU OF STANDARDS, Richard W. Roberts, *Director*

Issued March 1973

National Bureau of Standards Technical Note 763

Nat. Bur. Stand. (U.S.), Tech. Note 763, 25 pages (Mar. 1973)
CODEN: NBTNAE

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
Price $0.50 domestic postpaid or $0.35 G.P.O. Bookstore.

CONTENTS

A SET OF DEBUGGING AND MONITORING FACILITIES TO IMPROVE THE DIAGNOSTIC CAPABILITIES OF A COMPILER

Elizabeth Fong

Increasing concern with the quality of computer software today makes it important to evaluate critically the debugging facilities available in high-level languages. This paper presents a collection of program debugging and monitoring facilities to improve the diagnostic capabilities of a compiler. A distinction is made between debugging and monitoring facilities performed at compile time, at link/load time and at execution time. These facilities are described in terms of this breakdown with a conscious attempt to move the detection of errors from execution time to compile or link/load time, and to collect information when the information is available during the compilation process.

Key words: Compiler; debugging; error diagnostic; high-level programming languages; monitoring; procedural-oriented languages.

1. INTRODUCTION

Software production requires large amounts of both human and machine resources. The recent trend has been towards a more systematic development of software and in particular towards producing "quality" software that is effective and maintainable. During the program development stage, most of the programmer's effort is spent in debugging. Program debugging is still a very uncertain process. Professionals do not agree on how to get the bugs out of a program systematically, or how to construct the most reliable program. Tools, both automated and manual, exist but surprisingly, much of the work in this field has been described only in unpublished reports or passed on through oral tradition. Interest in debugging techniques has been aroused recently, and a collection of papers devoted to the topics of debugging and maintenance of large software systems can be found in Reference [1].[1]

The programming activity is rooted in the traditional environment, that is, writing the program in a high-level programming language, such as FORTRAN, ALGOL, COBOL, or PL/1 with the usual compile-link/load-go situation. To aid the programmers in the stage of program development, it is advantageous to build debugging aids and monitoring features into a high-level programming language system.

This paper presents a collection of program debugging and monitoring facilities to improve the diagnostic capabilities of a compiler. A distinction is made between debugging and monitoring facilities performed

[1] Figures in brackets indicate the literature reference at the end of this paper.

at compile time, at link/load time and at execution time. These facilities are described in terms of this breakdown with a conscious attempt:

- to move, as much as possible, the detection of errors from execution time to compile or link/load time in order to avoid overhead in an executing program, and
- to collect information when the information is available during the compilation process. At the user's option, this information could be output at a later time.

2. SCOPE AND SOURCES

Many programmers think of debugging aids as "dumps" (the display of the contents of storage cells) and "traces" (the display of the control flow during execution). However, these basic tools have become considerably more sophisticated over the years. In particular, very powerful debugging facilities have been built for the on-line interactive environment [2,3] giving the user a high-level language with which to control the setting of breakpoints and to allow interrogation of individual storage cells.

The universities in teaching programming to students, have found that they need an extensive set of debugging facilities. "Forgiving" compilers have emerged. These include not only error detection but error correction features as well. Examples include Cornell Computing Language (CORC) [4], Waterloo FORTRAN (WATFOR and WATFIV) [5], and Purdue Fast FORTRAN Translator (PUFFT) [6].

Program monitors are another programming aid. These can be viewed as a logical extension of the debugging aids described above. A program monitor is a program built to measure the behavior of another program. In general, the program monitors that are available commercially operate in two phases: data gathering and data presentation.

The idea of combining debugging and monitoring facilities as an augmentation to high-level languages has been attempted in the EXDAMS Project [7]. Both static and dynamic displays may be exhibited at the request of the programmer.

Dijkstra's notion of structured programming [8], and Wirth's concept of program development as the process of successive refinement [9] are attempts to move programming from a private art to a new level of precision. The essential ingredient in achieving a correct final program quickly seems to lie in the area called "good" programming practices. Examples of "good" programming practice include the exhibition of structured readable source statement as output, planting special checks and messages for the use of undeclared variables, and elimination of wild GOTO transfers, etc. Some of these ingredients could be implemented on a compiler.

3. DESCRIPTIONS OF DEBUGGING AND MONITORING FACILITIES

In a compiler environment, the program text is coded, keypunched and fed into a compiler which produces object code. The object code and a system library are link-edited and loaded by a link/loader which produces absolute code ready to be executed. (see Figure 1). Three distinct times can be identified at which debugging and monitoring can occur: compile time, link/load time, and execution. Each individual type of check is described in terms of this breakdown.

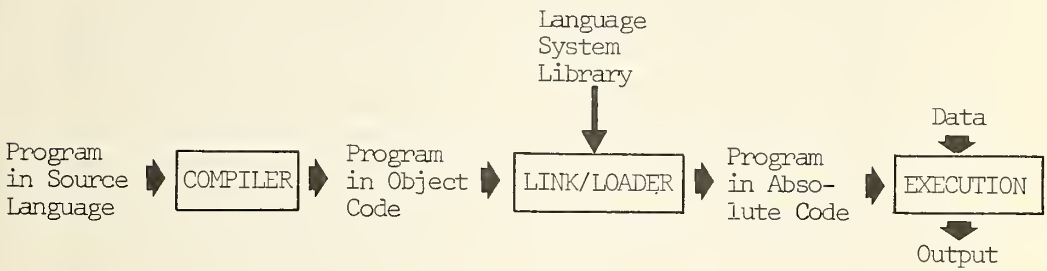

Figure 1

Debugging and monitoring features described below are categorized according to these stages. Some features may occur either at compile time or link-load time depending on the implementation. This three-part breakdown hypothesizes that the earlier an error is caught, the easier it is to deal with. Therefore, any tool or technique that moves detection of certain errors from execution time to compile time is valuable. This breakdown does not apply to the interpretive environment.

3.1 Compile Time Checks

The types of checks performed by the compiler are usually referred to as static checks because the problems they address are invariant with execution time and independent of the values of the input data. The following are descriptions of some program debugging and monitoring facilities that could be performed at compile time.

3.1.1 Syntax Checking. One of the first tasks done by a compiler is to scan the source program statements. Every high-level language has its own established set of syntax rules. Error situations occur when the compiler cannot translate a source statement, or the process of syntax-directed parsing is blocked.

Examples of syntax errors include misspelling of reserve words, punctuation errors such as unmatched parentheses, missing operators, illegal sequences of operators in expressions, or missing statement labels. In a class of university-developed compilers (e.g., CORC, PUFFT, WATFOR) with "forgiving" features built in, extensive syntax checking is done. These compilers also do some error-correction during source language scanning. For example, CORC attempts to define various categories of misspelling and always tries to repair the error and continue to compile executable code. All compilers have syntax checking facilities, however the extent of checking varies.

3.1.2 Static Concordance of Variables and Labels. At compile time, all the variables and labels are entered into a symbol table or an intermediate file. This information could be made available during compile time to produce one of the most beneficial aids to debugging. This facility is sometimes called a cross reference listing. The usual format consists of the symbolic name followed by the concordance of the line numbers where the variable is being defined and the line numbers where this variable is being referenced. Some compilers also print out data type and relocation information. An example appeared in Figure 2. In this case, the statement label concordance appeared as a separate listing. The advantages of such a facility would be to detect a variable or a label being defined but never referenced or vise versa. For a block-oriented language such as ALGOL 60 or PL/1, the variables and labels may be declared as local within a block. Exhibiting a static concordance of variables and labels plus their scope of definition would aid debugging. Most of the compilers provide this facility. There are also programming support packages available which can generate such a concordance.

3.1.3 Language Flagging Features. Due to historical development, most programming languages (e.g., FORTRAN) have undergone modifications and upgrading (e.g., FORTRAN II, Basic FORTRAN, FORTRAN IV, and extended FORTRAN). Moreover, each implementation of the language may contain differences. Flagging can be done during the source language scanning phase to detect language syntactic features which deviate from some "standard" definition such as ASA FORTRAN [10], or additional features from the previous versions. A message could be printed out by the compiler as a warning to the user that the particular feature is unique to this version of the compiler. Very few compilers have this flagging facility. An idea closely related to this can be found in a COBOL compiler which prints the message: "REDEF Name feature not the same as previously defined version". (See Figure 3). It is possible to determine the language feature differences by compiling the previously compiled program with the new compiler, however, the point here is to detect deviations specifically against a "standard" language definition, or to be able to identify a subset of the language that existed earlier. To make a full semantic check of a program is not possible.

3.1.4 Logical Segmentation of Programs. A large program may be considered as consisting of small logical segments. This permits a modular organization. Modularity aids debugging because it breaks up the program into manageable sized pieces. In ALGOL, there are BEGIN-END statements which create a logical segment. In COBOL, a paragraph or a section also is considered to be a logical segment. A formatting facility could be built into the compiler which would cause more readable source program statements to be printed out. Examples are: alignment of BEGINs with corresponding ENDs with nested inner blocks properly indented; hierarchical data declaration in COBOL or PL/1 are properly indented to reflect structures even when the input is not punched in the proper column; or reordering of source statements in order to group declaration statements, executable statements, and format statements. Figure 4 shows an example of an ALGOL program printout with block number and level number generated by the compiler to reflect logical segments. This facility, although not a debugging feature in the sense that it helps to uncover errors, could greatly add to the readability of a program. Producing readable documentation also helps another person in maintaining the program.

3.1.5 Static Control Structure Concordance. The normal flow of control in a program is sequential. However, control may be transferred elsewhere in the program by a transfer of control command, e.g., GOTO, RETURN. A topological structure diagram showing every transfer of control at the request of the user would aid debugging. Figure 5 contains such a static control structure printout, showing that a transfer of control is occurring at line number n to line number m. Some compilers do not provide a total topological control structure trace but merely print out messages such as "Control can never reach the next statement" or "There is no path to this statement" (See Figure 6 and 7).

3.2 Link/Load Time Checks

In most batch-oriented installations, compilation produces a binary program file, usually stored on direct access storage. Subprograms and the main program are compiled separately and bound together by the link/loader which associates external references and adjusts addresses. If an external reference is to a system library routine, it link/loads this routine. The information available at this point enables the system to perform the following type of debugging or monitoring checks:

3.2.1 Formal and Actual Argument Checks. A subroutine or function declaration consists of the name of the subroutine or function followed by a list of actual parameters.

For a programming language such as ASA Standard FORTRAN, the definition of the language specifies that the actual arguments must agree in order, number and type with the corresponding formal arguments. A check on all three items would insure that the program is adhering to a "standard". In order to perform checks at link/load time, it is necessary to carry more information than the usual external symbol definition, for example, the list of formal argument names and their data types, the corresponding actual argument names and their data types. The advantages of performing the checks at link/load time rather than execution time is that the check would be done only once, rather than each time the subprogram is executed. The tests that could be performed include the following:

a) If the subroutine or function does contain a non-standard return, test that the return label is indeed a label.

b) In the case of FORTRAN, if one of the arguments of the subroutine or function is a subroutine name or a function name, the compiler could check to determine if the name is being declared as an external procedure.

c) Some implementations of FORTRAN compilers allow the number of formal arguments to differ from the number of actual arguments. If the standard language definition, e.g., ASA FORTRAN standard, calls for a match in the number, the check might insure that the program is adhering to the appropriate standard.

3.2.2 Static Subroutine Structure Analysis. The static information about subroutine structure consists of caller and callee relationship derived from the program text. In most languages, this caller-callee relationship is invariant at execution time. This information is available at link/load time and usually contained in the external symbol definition table. The link/loader could optionally produce a concordance of the source language subprogram caller/callee names. An example of the "CALL" concordance (Figure 8) and "CALLED-BY" concordance (Figure 9) was developed at NBS on the UNIVAC 1108 computer.

3.3 Execution Time Checks

Dynamic information about the actual running of the program is obtainable at this time. To embed tests, the compiler usually inserts code at appropriate points to be executed in conjunction with the worker program. Every test introduced will decrease the running efficiency of the worker program and it is necessary to exercise care when deciding what to measure and where to embed tests. Attention also has to be given to where the tests are to be inserted to assure uniform checks.

3.3.1 Dynamic Trace of Subroutine Calls. Actual subroutine paths could be traced at execution time. This information not only aids in debugging but is very useful for program activity analysis. LEAP (Lambda Efficiency Analysis Program) [11], a software monitor, contains a special section to perform subroutine structure analysis. A tree-like diagram of the paths through the hierarchy of subprogram calls from the main program allows the user to identify the most significant call chains. Figure 10 shows a portion of a dynamic subroutine trace developed at NBS for the UNIVAC 1108. It consists of ten subroutines with its call chain properly reflected by the indented printout.

3.3.2 Backward Trace of Subroutine Calls Upon Error Termination. Upon error termination, an identification of where the error occurred plus a backward trace of the subroutine calls aids debugging. Figure 11 shows an example of backward trace of subroutine calls when the program terminated with an error. This example is generated on the UNIVAC 1108 FORTRAN compiler. This compiler uses an extra word in the calling sequence to store the "walk-back" or the backward reference to permit this kind of backward trace.

3.3.3 Variable Trace. This is a dynamic display of the specified variable and its content at each instant in time. The display usually occurs as an instruction by instruction accounting of information or at every instance of a value change. It is not only useful for debugging, but also useful in spotting the value changes of certain variables for program analysis.

3.3.4 Snapshot. This is similar to the variable trace except that the variables and their values are recorded periodically on entering or exiting certain regions of the program.

3.3.5 Flow Trace. This is the dynamic display of every branch point of a running program. The trace records the decision points and exhibits the branches taken.

3.3.6 Array Bounds Checking. This checking is sometimes built into a computer as a precaution against altering of values incorrectly. The value of the subscript is tested to determine if it is within the specified dimension of the array element, and if it is also an integer constant. The check is useful in monitoring array elements and spotting the activity level of various parts of the table.

3.3.7 Effective Address Check. This is a feature provided for memory protection. Bounds registers are set to certain permitted address ranges and every effective address value is checked against these bound registers. In some cases, it is implemented as a hardware feature to avoid system overhead. This feature is especially important in a multiprogramming environment.

3.3.8 Value of the Control Variable Upon Exit. In an iteration loop such as FORTRAN-DO or ALGOL-FOR, the value of the control variable upon exit is undefined if the exit is due to exhaustion of the loop, otherwise it is the same as it was immediately preceding the execution of the exit condition. The undefined situation is handled by implementors in different ways. Some compilers even try to "guess" what the user intended. The trouble starts when programmers make use of these undefined situations on a particular implementation of the compiler, and later discover that the same program does not work correctly on a different implementation. Debugging this type of error is very difficult because it involves an understanding of the semantics of different compiler implementation of undefined situations. One of the debugging facilities for this particular undefined situation could be to report all later use of the control variable or intentionally set the value of the control variable to "undefined" (minus zero or some such number) when the exit of the loop is due to exhaustion of the loop.

3.3.9 GOTO Checks. Recently, there has been considerable interest in eliminating the GOTO statement [12, 13]. Dijkstra claims the use of GOTO statements is undesirable, and avoiding it would increase the readibility and proveability of the program. When GOTO cannot be avoided, and when label variables are allowed e.g. PL/1, the following kinds of checks could be performed by the compiler:

(a) Flag error if the transfer is made to itself or to a non-executable statement.

(b) Flag error if the transfer label value is negative or undefined.

(c) Flag error if the transfer label value is outside of the user's assigned program space.

(d) Flag warning if the transfer label goes within an iteration loop.

3.3.10 Truncation Error Warning. On an arithmetic or MOVE-data operation, some bits may be dropped due to computer word-length. Overflow to the left or to the right of the computer word is called truncation error. If such overflow is suspected or detected, a warning would be printed by the compiler.

4. CONCLUSIONS

Increasing concern with the quality of computer software today makes it important to evaluate critically the debugging facilities available in high-level languages. The debugging and monitoring aids described above are particularly useful during early implementation and initial system integration stages. They could be automated by embedding these checks at appropriate points in the compilation-execution process when all the needed information is available.

The list of features is not geared to any particular high-level programming language; however, some of the features described are applicable only to particular language constructs. Techniques of implementation and the question of how to invoke and suppress these debugging and monitoring facilities have not been addressed here.

Commercially available compilers usually provide some debugging facilities; however, certain trade-off decisions are made which usually sacrifice the extent of providing debugging aids in favor of efficiency of the compiler. Such a list might prove to be useful as an evaluation criteria in determining the capabilities of a compiler.

These debugging and monitoring facilities described above could be automated to aid programmers during program development stage. Certain bugs are very much problem and situation dependent. Such bugs are difficult even to anticipate. For instance, it would be in general impossible for the compiler to isolate a sequence of code and determine that it is a non-terminating loop. There also exists a whole class of "Timing Bugs" which, when certain combination of situation occurs, the program behaves abnormally.

The problems in the area of debugging and monitoring are many: The question of how much to output, what should be outputted, how to recover from error conditions, what should be the default situation, etc. All of these questions are still open and the decisions are usually left with the implementor.

There are other automated testing techniques for validating purposes, but they are beyond the scope of this paper. Such techniques include benchmark tests, exhaustive exercising of the program with different input data, and proof of correctness using formal logic, etc. Testing methods for validation purposes are very different from those mentioned here, which are limited to debugging and monitoring aids for program development purposes.

The author would like to thank Mrs. Frances E. Holberton for giving her very valuable suggestions. The author also wishes to acknowledge Dr. Rona Stillman for reading the paper. Dr. Selden Stewart and Mr. Justin Walker provided some of the computer examples.

```
      PROGRAM     PART3     TRACE                           CDC 6600 FTN V3.0-P291 OPT=0  08/04/72  13.42.03.

VARIABLES      SN  TYPE            RELOCATION
 1122  MCFVB      LOGICAL                        REFS       45       154   DEFINED      113
 1123  MCGVB      LOGICAL                        REFS       45       160   DEFINED      114
 1132  MCHVB      LOGICAL                        REFS       45       142       154   DEFINED      115
 1124  MCIVB      LOGICAL                        REFS       45       160   DEFINED      116
 1125  MCJVB      LOGICAL                        REFS       45       160   DEFINED      147
 1126  MCKVB      LOGICAL                        REFS       45       154   DEFINED      149
 1127  MCLVB      LOGICAL                        REFS       45       154   DEFINED      150
 1130  MCMVB      LOGICAL                        REFS       45       154   DEFINED      151
 1131  MCNVB      LOGICAL                        REFS       45       132       154   DEFINED      152
 1151  MOCVI      INTEGER                        REFS    2*762      1039      1040      1041     1042
                                              DEFINED      760      1036
 1142  MRRVI      INTEGER                        REFS      379       383       386       390      393      396      399
                                                  404      464       470       476       483      489      499      544
                                                  549      606       612       618       625      640      684      689
                                                  741      746       750       756       763      769      777      782
                                                  786      790       797       802       897      909
                                              DEFINED      378       382       385       389      392      395      398
                                                  403      463       469       475       482      488      498      502
                                                  505      508       511       514       517      520      523      526
                                                  529      532       535       538       541      605      611      617
                                                  624      638       642       645       648      651      654      657
                                                  660      663       666       669       672      675      678      681
                                                  737      744       748       752       758      766      773      780
                                                  784      788       792       800       896      908
 1152  NECVI      INTEGER                        REFS      762   DEFINED       761
 1137  NUVI       INTEGER                     DEFINED       73   I/O REFS       75        94       95       96      103
                                                  154      160       199       216       219      238      241      260
                                                  263      284       287       308       311      328      331      339
                                                  371      379       383       386       390      393      396      399
                                                  404      406       452       464       470      476      483      489
                                                  499      544       549       551       595      606      612      618
                                                  625      640       684       689       691      730      734      741
                                                  746      750       756       763       769      771      777      782
                                                  786      790       797       802       803      841      855      892
                                                  897      906       909       918       956      972      977      984
                                                 1026     1031      1043      1046      1058


STATEMENT LABELS               DEF LINE    REFERENCES
   27  70        FMT               91          88        94
    0  71        FMT               76          75
   35  72        FMT               92          89        95
   43  73        FMT               93          90        96
   51  160       FMT              104         103
   64  161       FMT              158         154
   73  162       FMT              163         160
  101  170       FMT              200         199
  117  171       FMT              217         216
```

Figure 2

```
109  109  000036    30 0          77  MASTER-IN-CTR            PICTURE     9(6)  VALUE 0.
110  110  000037    31 0          77  REC-CTR                  PICTURE     9(6)  VALUE 0.
111  111  000040    32 0          77  PRINT-CTR                PICTURE     9(6)  VALUE 0.
112  112  000041    33 0          77  CARD-CTR                 PICTURE     9(6)  VALUE  0.
113  113  000042    34 0  001070  77  C0                       PICTURE     99   VALUE 0.                 DRM*
114  114  000043    35 0  001080  77  C1                       PICTURE     99   VALUE 1.                 DRM*
115  115  000044    36 0  001090  77  C2                       PICTURE     99   VALUE 2.                 DRM*
116  116  000045    37 0  001100  77  C3                       PICTURE     99   VALUE 3.                 DRM*
117  117  000046    38 0  001110  77  C4                       PICTURE     99   VALUE 4.                 DRM*
118  118  000047    39 0  001120  77  C5                       PICTURE     99   VALUE 5.                 DRM*
119  119  000050    40 0  001130  77  C6                       PICTURE     99   VALUE 6.                 DRM*
120  120  000051    41 0  001140  77  C7                       PICTURE     99   VALUE 7.                 DRM*
121  121  000052    42 0  001150  77  C8                       PICTURE     99   VALUE 8.                 DRM*
122  122  000053    43 0  001160  77  C9                       PICTURE     99   VALUE 9.                 DRM*
123  123  000054    44 0  001170  77  C10                      PICTURE     99   VALUE 10.                DRM*
124  124  000055    45 0  001180  77  C11                      PICTURE     99   VALUE 11.                DRM*
125  125  000056    46 0  001190  77  C12                      PICTURE     99   VALUE 12.                DRM*
126  126  000057    47 0  001200  01  FILLER.                                                            DRM
127  127  000057    47 0  001210          05  WHOLE-GRABBER    PICTURE     X(1320).                       DRM
128  128  000057    47 0  001220          05  APR     REDEFINES WHOLE-GRABBER.                           DRM
129  129  000057    47 0  001230              07  PLACER       PICTURE     X OCCURS 1320 TIMES.          DRM
ERROR     0130              REDEFINES NAME NOT SAME AS PREVIOUSLY DEFINED AREA--REDEF IS MADE TO PREVIOUS.
130  130  000057    47 0  001240          05  BPR     REDEFINES  APR.                                    DRM
131  131  000057    47 0  001250              07  NUMS         PICTURE     9(6).                          DRM
132  132  000060    48 0  001260              07  FILLER       PICTURE     X(1314).                       DRM
133  133  000413   267 0  001270  01  FILLER.                                                            DRM
134  134  000413   267 0  001280          05  BEGIN-HOLDER     PICTURE     9(4).                          DRM
135  135  000414   268 0  001290  01  FILLER.                                                            DRM
136  136  000414   268 0  001300      05  REMAR                PICTURE     X(6).                          DRM
137  137  000414   268 0  001310      05  NUMBER-2   REDEFINES  REMAR.                                    DRM
138  138  000414   268 0  001320        07  NUM-3              PICTURE     9(3).                          DRM
139  139  000414   268 3  001330        07  FILLER             PICTURE     X(3).                          DRM
ERROR     0140              REDEFINES NAME NOT SAME AS PREVIOUSLY DEFINED AREA--REDEF IS MADE TO PREVIOUS.
140  140  000414   268 0  001340      05  NUMBER-3   REDEFINES   NUMBER-2.                               DRM
141  141  000414   268 0  001350        07  NUM-2              PICTURE     9(2).                          DRM
142  142  000414   268 2  001360        07  FILLER             PICTURE     X(4).                          DRM
ERROR     0143              REDEFINES NAME NOT SAME AS PREVIOUSLY DEFINED AREA--REDEF IS MADE TO PREVIOUS.
143  143  000414   268 0  001370      05  NUMBER-4   REDEFINES   NUMBER-3.                               DRM
144  144  000414   268 0  001380        07  NUM-6              PICTURE     9(6).                          DRM
145  145  000415   269 0  001390  01  FILLER.                                                            DRM
146  146  000415   269 0  001400        05  DATA-NAME-4        PICTURE     Z(3)9.                         DRM
147  147  000415   269 0  001410        05  D-4   REDEFINES    DATA-NAME-4.                               DRM
148  148  000415   269 0  001420          07  DATA-Y           PICTURE     X   OCCURS 4 TIMES.            DRM
149  149  000416   270 0  001430  01  HOLD-CARD-REC.                                                     DRM
150  150  000416   270 0  001440          05  FILLER           PICTURE     X(132).                        DRM
151  151  000444   292 0  001450  01  FILLER.                                                            DRM
152  152  000444   292 0  001460          07  HOLD-RESEQUENCE PICTURE      9  VALUE 0.                    DRM
153  153  000444   292 0  001470            88  YOU-MUST-RESEQUENCE PICTURE 9  VALUE 1.                   DRM
154  154  000445   293 0  001480  01  FILLER.                                                            DRM
155  155  000445   293 0  001490          07  PRINT-HOLD       PICTURE     9 VALUE 0.                     DRM
156  156  000445   293 0  001500            88  PRINT-CARD     PICTURE     9 VALUE 1.                     DRM
157  157  000446   294 0  001510  01  FILLER.                                                            DRM
158  158  000446   294 0  001520          07  TAPE-HOLD        PICTURE     9 VALUE 0.                     DRM
159  159  000446   294 0  001530            88  PRINT-CHG-SKIP PICTURE     9 VALUE 1.                     DRM
160  160  000447   295 0  001540  01  FILLER.                                                            DRM
161  161  000447   295 0  001550      03  SC-1.                                                          DRM
162  162  000447   295 0  001560          05  FILLER           PICTURE     X  VALUE QUOTE.                DRM
163  163  000447   295 1  001570          05  FILLER           PICTURE     X(13)  VALUE                   DRM
```

Figure 3

```
PAGE:       5

@J FOR SYNTAX,SYNTAX
COMPILED BY UNIVAC 1107/8 ALGOL ON 17 FEB 70 AT 13:42:44      1 MAY 66 LEVEL 2

                   BEGIN                                                        B1
BLOCK 1     LEVEL 1
                   STRING STACK(100),   COMPARE(21),   PROG(500),
                         FULE(TITLE(12),3,CHAIN(21),2,ARROW(1),2,PROM(14),1,COMM(10),
                               2,PROC(12));
                   STRING ARRAY RECOG(80:1:500),  TABLE(11:1:100);
                   INTEGER ARRAY NTAB(1:100);
                   STRING INCO(10);
                   STRING ARRAY RES(10:1:75);
                   STRING MATCHRES(10);
                   FORMAT FORM2(A,X1,S10)I
                   LOCAL LABEL GONE;
                   FORMAT FORM1(A,S80);
                   INTEGER OTHERWISE;
BLOCK 2     LEVEL 2
               INTEGER PROCEDURE INDEX(ITEM,C);
                   STRING C,ITEM;
                   VALUE C,ITEM;
                   BEGIN INTEGER K;                                             B2
                   FOR K=(1,1,80) DO
                        IF ITEM(K) EQL C THEN GO TO MATCH;
                   K=0;
               MATCH:   INDEX=K;
  END BLOCK 2
                   END INDEX;                                                   E2
BLOCK 3     LEVEL 2
               INTEGER PROCEDURE NUMEL(LIST);
                   STRING LIST;
                   VALUE LIST;
                   BEGIN INTEGER I,IJ,NM;                                       B3
```

Figure 4

UNCLASSIFIED

```
FTN10 01  08-07-72   11.670   ***************************************************************** LABEL ******  PAGE   1

  1        C**********************************************************************        F4330010
  2        C*****                                                                         F4330020
  3        C*****                           KFI - (433)                                   F4330030
  4        C*****                                                                         F4330040
  5        C**********************************************************************        F4330050
  6        C*****INTEGER FUNCTION OF LOGICAL ARGUMENT(TEST 3)                              F4330060
  7              FUNCTION KFI(AWVB)                                                       F4330070
  8              LOGICAL AWVB                                                             F4330080
  9              IF (AWVB) GO TO 4331                                                     F4330090
 10        4330  IF (.NOT.AWVB) GO TO 4332                                                F4330100
 11              RETURN                                                                   F4330110
 12        4331  KFI = 2                                                                  F4330120
 13              GO TO 4330                                                               F4330130
 14        4332  KFI = 0                                                                  F4330140
 15              RETURN                                                                   F4330150
 16              END                                                                      F4330160


TRANSFERS....

FROM LINE# TO LINE#   FROM LINE# TO LINE#   FROM LINE# TO LINE#   FROM LINE# TO LINE#   FROM LINE# TO LINE#

     15    RETURN          13         10          11    RETURN          10         14           9         12
```

Figure 5

```
00271                                  000244   7201 00 00 0 000000 0004                SLJ        NIO2$
00271                                  000245   7404 00 00 0 000254 0001                J          8869L
00272    90.    8866  WRITE(NUVI,8867) KCHVI                                        F4100800
00272                                  000246   7413 13 00 0 000000 0003         8866L  LMJ        X11,NWOU$
00272                                  000247   0000 00 01 0 000013 0000                +          0001,NUVI
00272                                  000250   0000 00 00 0 000051 0000                +          8867F
00273                                  000251   0000 13 12 0 000112 0000                +          0272,(W.B.)
00273                                  000252   0000 01 00 0 000014 0000                +          KCHVI
00273                                  000253   7201 00 00 0 000000 0004                SLJ        NIO2$
00275    81.    8867  FORMAT (//36H  DO RANGE ENDING AT STATEMENT LABEL,I5,40H  IS IN ERF4100810
00275    82.          1ROR. SKIP TO NEXT TEST SEGMENT)                              F4100820
00276    83.    8868  FORMAT (//35H  THIS SEGMENT SUCCESSFULLY TESTED.)             F4100830
00277    84.    8869  REWIND INVI                                                   F4100840
00277                                  000254   7413 13 00 0 000000 0007         8869L  LMJ        X11,NREW$
00277                                  000255   0000 00 01 0 000012 0000                +          0001,INVI
00310    85.          RETURN                                                        F4100850
00300    86.    C*****.    END OF TEST SEGMENT 410                                  F4100860
00300                                  000256   0000 13 17 0 000112 0000                +          0277,(W.B.)
00300                                  000257   2300 03 00 0 000261 0001                LR         R3,$+2
00300                                  000260   7404 00 00 0 000267 0001                J          NBPROS+183
00301  .DIAGNOSTIC.  CONTROL CAN NEVER REACH THE NEXT STATEMENT
00301    87.          STOP
00301                                  000261   0000 14 00 0 000112 0000                +          0300,(W.B.)
00301                                  000262   7413 13 00 0 000000 0013                LMJ        X11,NSTOP$
00301                                  000263   0000 00 00 0 000107 0000                +          (005050505050505)
00301                                  000264   2300 03 00 0 000266 0001                LR         R3,$+2
00301                                  000265   7404 00 00 0 000267 0001                J          NBPROS+183
00301                                  000266   0000 14 01 0 000112 0000                +          0301,(W.B.)
00301                                  000267   2700 13 00 0 000113 0000         BEXIT  LX         X11,(W.B.)
00301                                  000270   7113 06 00 0 000116 0000                DL         A6,NTEMP$+2
00301                                  000271   7113 14 00 0 000120 0000                DL         A12,NTEMP$+4
00301                                  000272   7113 16 00 0 000122 0000                DL         A14,NTEMP$+6
00301                                  000273   2700 01 00 0 000124 0000                LX         X1,NTEMP$+8
00301                                  000274   2700 02 00 0 000125 0000                LX         X2,NTEMP$+9
00301                                  000275   2300 04 00 0 000126 0000                LR         R4,NTEMP$+10
00301                                  000276   2300 05 00 0 000127 0000                LR         R5,NTEMP$+11
00301                                  000277   2300 06 00 0 000130 0000                LR         R6,NTEMP$+12
00301                                  000300   2300 07 00 0 000131 0000                LR         R7,NTEMP$+13
00301                                  000301   5015 00 00 1 000110 0000                TZ,S1      .INJP$
00301                                  000302   7404 00 00 1 000110 0000                J          .INJP$
00301                                  000303   7404 00 00 0 000000 0014                J          NERR3$
00301                                  000304   0601 13 00 0 000113 0000         SUBR2  SX,H2      X11,(W.B.)
00301                                  000305   7112 06 00 0 000116 0000                DS         A6,NTEMP$+2
00301                                  000306   7112 14 00 0 000120 0000                DS         A12,NTEMP$+4
00301                                  000307   7112 16 00 0 000122 0000                DS         A14,NTEMP$+6
00301                                  000310   0600 01 00 0 000124 0000                SX         X1,NTEMP$+8
00301                                  000311   0600 02 00 0 000125 0000                SX         X2,NTEMP$+9
00301                                  000312   0400 04 00 0 000126 0000                SR         R4,NTEMP$+10
00301                                  000313   0400 05 00 0 000127 0000                SR         R5,NTEMP$+11
00301                                  000314   0400 06 00 0 000130 0000                SR         R6,NTEMP$+12
00301                                  000315   0400 07 00 0 000131 0000                SR         R7,NTEMP$+13
00302    88.          END
00304                                  000316   7404 00 00 0 000000 0001                J          NBPROS

      END OF UNIVAC 1108 FORTRAN V COMPILATION.      2 .DIAGNOSTIC. MESSAGE(S)
PHASE 1 TIME =   0 SEC.
PHASE 2 TIME =   0 SEC.
PHASE 3 TIME =   1 SEC.
```

Figure 6

```
          PROGRAM     PART3     TRACE                    CDC 6600 FTN V3.0-P291

CARD NO.  SEVERITY             DIAGNOSTIC

     630       I               THERE IS NO PATH TO THIS STATEMENT
    1069      FE               UNDEFINED STATEMENT NUMBERS, SEE BELOW

          UNDEFINED LABELS
          221
```

Figure 7

```
CALL CDNCOROANCE.

RDUTINE A       CALLED B
                       C
                       O
                       E
                       F
                       G
                       H
                       J
                       K

RDUTINE B       CALLED C
                       D
                       E
                       F
                       G
                       H
                       J
                       K

RDUTINE C       CALLED O
                       E
                       F
                       G
                       H
                       J
                       K

RDUTINE D       CALLED E
                       F
                       G
                       H
                       J
                       K

ROUTINE E       CALLED F
                       G
                       H
                       J
                       K

RDUTINE F       CALLED G
                       H
                       J
                       K

ROUTINE G       CALLED H
                       J
                       K

ROUTINE H       CALLED J
                       K

RDUTINE J       CALLED K

RDUTINE MAIN    CALLED A
```

Figure 8

```
CALLED-BY CONCORDANCE.

SUBROUTINE A        WAS CALLED BY MAIN

SUBROUTINE B        WAS CALLED BY A

SUBROUTINE C        WAS CALLED BY A
                                  B

SUBROUTINE D        WAS CALLED BY A
                                  B
                                  C

SUBROUTINE E        WAS CALLED BY A
                                  B
                                  C
                                  D

SUBROUTINE F        WAS CALLED BY A
                                  B
                                  C
                                  D
                                  E

SUBROUTINE G        WAS CALLED BY A
                                  B
                                  C
                                  D
                                  E
                                  F

SUBROUTINE H        WAS CALLED BY A
                                  B
                                  C
                                  D
                                  E
                                  F
                                  G

SUBROUTINE J        WAS CALLED BY A
                                  B
                                  C
                                  D
                                  E
                                  F
                                  G
                                  H

SUBROUTINE K        WAS CALLED BY A
                                  B
                                  C
                                  D
                                  E
                                  F
                                  G
                                  H
                                  J
```

Figure 9

```
DYNAMIC SUBROUTINE TRACE FOR MAIN
MEASUREMENT BEGUN AT    68393.429 SEC SINCE MIDNITE
A
      B
            C
                  D
                        E
                              F
                                    G
                                          H
                                                J
                                                      K
                                                K
                                          J
                                                K
                                          K
                                    H
                                          J
                                                K
                                          K
                                    J
                                          K
                                    K
                              G
                                    H
                                          J
                                                K
                                          K
                                    J
                                          K
                                    K
                              H
                                    J
                                          K
                                    K
                              J
                                    K
                              K
                        F
                              G
                                    H
                                          J
                                                K
                                          K
                                    J
                                          K
                                    K
                              H
                                    J
                                          K
                                    K
                              J
                                    K
                              K
                        G
                              H
                                    J
                                          K
```

Figure 10

```
SUBR1 - (170) SUBROUTINE SUBPROGRAM
              WITHOUT AN ARGUMENT

ASA REF.  - 8.4.1

RESULTS

 ERROR TERMINATION IN NEXPI$ ROUTINE

NEXPI$ CALLED  AT  SEQUENCE  NUMBER  00120  OF  SUBR2

SUBR2  CALLED  AT  SEQUENCE  NUMBER  00102  OF  MAIN PROGRAM
```

Figure 11

5. REFERENCES

[1] Rustin R. (ed.) "Debugging Techniques in Large Systems" Courant Computer Science Symposium 1, 1971.

[2] Evans, T. G. and D. L. Darley, "On-line Debugging Techniques - A Survey" In AFIPS, FJCC Vol. 29, 1966, 37-50.

[3] Evans, T. G. and D. L. Darley, "DEBUG - An Extension to Current On-line Debugging Techniques", Comm. ACM 8,5 (May 1965) 321-326.

[4] Freeman, D. N., "Error Correction in CORC - The Cornell Computing Language", AFIPS Vol. 26, SJCC, 1965, 15-31.

[5] Cowan, D. D. and J. W. Graham, "Design Characteristics of the WATFOR Compiler", SIGPLAN Notices Vol. 5, No. 7, July 1970, 25-36.

[6] Rosen, S., R. A. Spurgean and J. K. Donnelly, "PUFFT- The Purdue University Fast FORTRAN Translator", Comm, ACM, 8, 11 (Nov. 1965).

[7] Balzer, R. N. "EXDAMS - Extendable Debugging and Monitoring System", AFIPS Vol. 34, SJCC 1969, 567-580.

[8] Dijkstra, E. W. "Structured Programming", In NATO Conference on Software Engineering Technique, Buxton, J. N. and Randell, B. (Eds.) April 1970.

[9] Wirth, N. "Program Development by Stepwise Refinement", Comm. ACM 14,4 (April 1971), 221-227.

[10] American Standard FORTRAN, Published by American Standards Assn. Inc. UDC 681.3, 1966.

[11] Lambda Corporation, "A Guide to Program Improvement with LEAP", Available from Lambda Corp., 1501 Wilson Blvd, Arlington, Virginia 22209, Jan. 1970.

[12] Knuth, D. E. and R. W. Floyd, "Notes on Avoiding GOTO Statement", Report No. CS-148, Computer Science Department, Stanford Univ, 1970.

[13] Dijkstra, E. W. "GOTO Statement Considered Harmful", Comm. ACM 11, 3(March 1968), 147.

FORM NBS-114A (1-71)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET	1. PUBLICATION OR REPORT NO. NBS TN-763	2. Gov't Accession No.	3. Recipient's Accession No.

4. TITLE AND SUBTITLE A Set of Debugging and Monitoring Facilities to Improve Diagnostic Capabilities of a Compiler.	5. Publication Date March 1973 6. Performing Organization Code
7. AUTHOR(S) Elizabeth N. Fong	8. Performing Organization
9. PERFORMING ORGANIZATION NAME AND ADDRESS NATIONAL BUREAU OF STANDARDS DEPARTMENT OF COMMERCE WASHINGTON, D.C. 20234	10. Project/Task/Work Unit No. 11. Contract/Grant No.
12. Sponsoring Organization Name and Address Same as No. 9	13. Type of Report & Period Covered Final 14. Sponsoring Agency Code

15. SUPPLEMENTARY NOTES

16. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here.)

Increasing concern with the quality of computer software today makes it important to evaluate critically the debugging facilities available in high-level languages. This paper presents a collection of program debugging and monitoring facilities to improve the diagnostic capabilities of a compiler. A distinction is made between debugging and monitoring facilities performed at compile time, at link/load time and at execution time. These facilities are described in terms of this breakdown with a conscious attempt to move the detection of errors from execution time to compile or link/load time, and to collect information when the information is available during the compilation process.

17. KEY WORDS (Alphabetical order, separated by semicolons)
Compiler; debugging; error diagnostic; high-level programming languages; monitoring; procedural-oriented languages.

18. AVAILABILITY STATEMENT	19. SECURITY CLASS (THIS REPORT)	21. NO. OF PAGES
[X] UNLIMITED.	UNCLASSIFIED	25
[ ] FOR OFFICIAL DISTRIBUTION. DO NOT RELEASE TO NTIS.	20. SECURITY CLASS (THIS PAGE) UNCLASSIFIED	22. Price $.50 Domestic Postpaid $.35 GPO Bookstore

USCOMM-DC 66244-P71

☆ U. S. GOVERNMENT PRINTING OFFICE : 1973 O - 511-324 (203)

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH reports National Bureau of Standards research and development in physics, mathematics, and chemistry. Comprehensive scientific papers give complete details of the work, including laboratory data, experimental procedures, and theoretical and mathematical analyses. Illustrated with photographs, drawings, and charts. Includes listings of other NBS papers as issued.

Published in two sections, available separately:

- **Physics and Chemistry (Section A)**

Papers of interest primarily to scientists working in these fields. This section covers a broad range of physical and chemical research, with major emphasis on standards of physical measurement, fundamental constants, and properties of matter. Issued six times a year. Annual subscription: Domestic, $17.00; Foreign, $21.25.

- **Mathematical Sciences (Section B)**

Studies and compilations designed mainly for the mathematician and theoretical physicist. Topics in mathematical statistics, theory of experiment design, numerical analysis, theoretical physics and chemistry, logical design and programming of computers and computer systems. Short numerical tables. Issued quarterly. Annual subscription: Domestic, $9.00; Foreign, $11.25.

TECHNICAL NEWS BULLETIN

The best single source of information concerning the Bureau's measurement, research, developmental, cooperative, and publication activities, this monthly publication is designed for the industry-oriented individual whose daily work involves intimate contact with science and technology—*for engineers, chemists, physicists, research managers, product-development managers, and company executives.* Includes listing of all NBS papers as issued. Annual subscription: Domestic, $6.50; Foreign, $8.25.

NONPERIODICALS

Applied Mathematics Series. Mathematical tables, manuals, and studies.

Building Science Series. Research results, test methods, and performance criteria of building materials, components, systems, and structures.

Handbooks. Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications. Proceedings of NBS conferences, bibliographies, annual reports, wall charts, pamphlets, etc.

Monographs. Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

National Standard Reference Data Series. NSRDS provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated.

Product Standards. Provide requirements for sizes, types, quality, and methods for testing various industrial products. These standards are developed cooperatively with interested Government and industry groups and provide the basis for common understanding of product characteristics for both buyers and sellers. Their use is voluntary.

Technical Notes. This series consists of communications and reports (covering both other-agency and NBS-sponsored work) of limited or transitory interest.

Federal Information Processing Standards Publications. This series is the official publication within the Federal Government for information on standards adopted and promulgated under the Public Law 89–306, and Bureau of the Budget Circular A–86 entitled, Standardization of Data Elements and Codes in Data Systems.

Consumer Information Series. Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

BIBLIOGRAPHIC SUBSCRIPTION SERVICES

The following current-awareness and literature-survey bibliographies are issued periodically by the Bureau:

Cryogenic Data Center Current Awareness Service (Publications and Reports of Interest in Cryogenics). A literature survey issued weekly. Annual subscription: Domestic, $20.00; foreign, $25.00.

Liquefied Natural Gas. A literature survey issued quarterly. Annual subscription: $20.00.

Superconducting Devices and Materials. A literature survey issued quarterly. Annual subscription: $20.00. Send subscription orders and remittances for the preceding bibliographic services to the U.S. Department of Commerce, National Technical Information Service, Springfield, Va. 22151.

Electromagnetic Metrology Current Awareness Service (Abstracts of Selected Articles on Measurement Techniques and Standards of Electromagnetic Quantities from D-C to Millimeter-Wave Frequencies). Issued monthly. Annual subscription: $100.00 (Special rates for multi-subscriptions). Send subscription order and remittance to the Electromagnetic Metrology Information Center, Electromagnetics Division, National Bureau of Standards, Boulder, Colo. 80302.

Order NBS publications (except Bibliographic Subscription Services) from: Superintendent of Documents, Government Printing Office, Washington, D.C. 20402.

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Washington, D.C. 20234

OFFICIAL BUSINESS

Penalty for Private Use, $300

POSTAGE AND FEES PAID
U.S. DEPARTMENT OF COMMERCE
215

U.S.MAIL