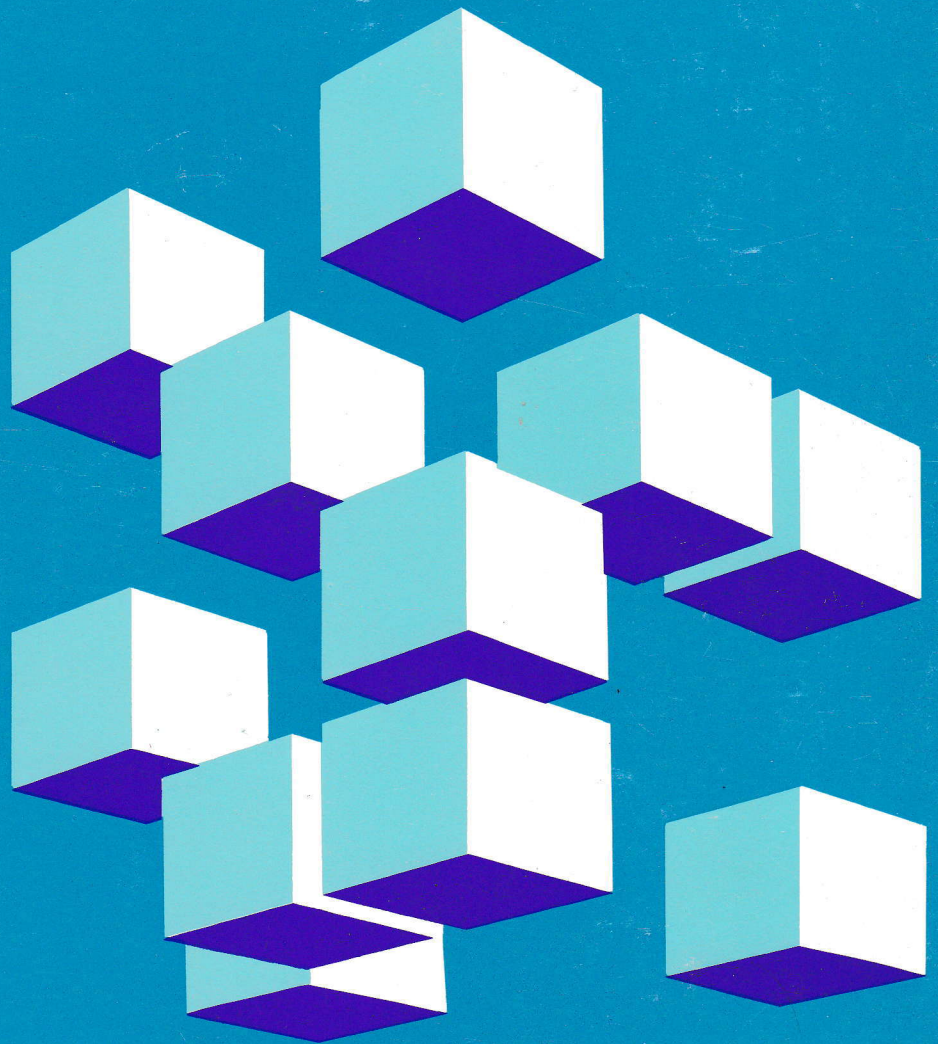


**IBM**

**VS Pascal  
Diagnosis Guide  
and Reference**

Licensed  
Program



Order Number LY27-9525-0  
File Number S370-37

Restricted Materials of IBM  
Licensed Materials-Property of IBM  
© Copyright IBM Corp. 1987

Program Numbers  
5668-767  
5668-717  
Release 1



# VS Pascal Diagnosis Guide and Reference

Licensed  
Program



### **First Edition (June 1987)**

This edition applies to Release 1 of VS Pascal, Licensed Programs 5668-767 (Compiler and Library) and 5668-717 (Library only), and to any subsequent releases until otherwise indicated in new editions or technical newsletters.

Changes are made periodically to this publication; before using this publication in connection with the operation of IBM systems, consult the latest *IBM System/370, 30xx, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM licensed program in this publication is not intended to state or imply that only IBM's program may be used. Any functionally equivalent program may be used instead.

Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality. If you request publications from the address given below, your order will be delayed because publications are not stocked there.

A form for readers' comments is provided at the back of this publication. If the form has been removed, comments may be addressed to IBM Corporation, P.O. Box 50020, Programming Publishing, San Jose, California, U.S.A. 95150. IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

This is a licensed document that contains restricted materials of International Business Machines Corporation. © Copyright International Business Machines Corporation 1987. All rights reserved.

---

## Preface

This book presents a systematic way of selecting keywords to describe a suspected failure in the VS Pascal Compiler or Library. It assumes that you have:

- Used all the debugging tools and options available to you, such as static debug statements, or the VS Pascal Interactive Debugging Tool.
- Examined all VS Pascal messages and error conditions produced by the program, and corrected them when possible.
- Ensured, as far as possible, that the error is occurring in the VS Pascal Compiler or Library and not in the application program.
- Noted the specific sequence of events preceding the error condition.

---

## How to Use This Book

If you suspect a product failure, follow the instructions for keyword selection in “Developing a Keyword String” on page 7.

Then use the keywords to search an indexed data base to determine whether your problem has been previously described in an APAR (authorized program analysis report). You may do the search yourself if you have access to a suitable data base such as Info Access or Early Warning System, or you can call the IBM Support Center to search the Software Support Facility. “Using the Keyword String as a Search Argument” on page 19 describes the search procedure.

If no APAR is identified for your product failure, you may be asked to submit one so that the product can be corrected if necessary. “Preparing an APAR” on page 21 describes this procedure.

---

## VS Pascal Publications

The publications in the VS Pascal library are:

- *VS Pascal Licensed Program Specifications*, GC26-4317, serves as the basis for warranty.
- *VS Pascal General Information*, GC26-4318, provides a general description of the VS Pascal licensed program.
- *VS Pascal Application Programming Guide*, SC26-4319, gives guidance on compiling, executing, and debugging VS Pascal programs.
- *VS Pascal Language Reference*, SC26-4320, provides information on the elements of the VS Pascal language.
- *VS Pascal Installation Guide for MVS*, SC26-4321, describes how to install VS Pascal under MVS/SP or MVS/XA.
- *VS Pascal Installation Guide for VM/System Product*, SC26-4342, describes how to install VS Pascal under VM/SP.
- *VS Pascal Reference Summary*, SX26-3760, is a convenient quick-reference summary of VS Pascal.

---

## Related Publications

- *OS/VS Message Library: VS2 System Messages*, GC38-1002
- *MVS/Extended Architecture Message Library: System Messages*, GC28-1156
- *Virtual Machine/System Product System Messages and Codes*, SC19-6204
- *OS/VS2 MVS Supervisor Services and Macro Instructions*, GC28-0683
- *MVS/Extended Architecture Supervisor Services and Macro Instructions*, GC28-1154
- *Field Engineering Programming Systems General Information*, G229-2228

---

## Contents

---

<b>Part 1: Guide</b> .....	<b>1</b>
<b>Introduction to Guide Information</b> .....	<b>3</b>
Keyword Usage .....	3
Diagnosis Procedure .....	5
<b>Developing a Keyword String</b> .....	<b>7</b>
Component Identification Keyword .....	7
Type of Failure Keyword .....	7
Abnormal Termination Problems .....	8
Procedure .....	8
Example .....	9
Message Problems .....	10
Procedure .....	11
Examples .....	13
No Response Problems .....	14
Procedure .....	14
Example .....	14
Documentation Problems .....	15
Procedure .....	15
Incorrect Output Problems .....	16
Procedure .....	16
Example .....	17
Performance Problems .....	17
Procedure .....	17
Example .....	17
<b>Using the Keyword String as a Search Argument</b> .....	<b>19</b>
Procedure .....	19
<b>Preparing an APAR</b> .....	<b>21</b>
Procedure .....	21
<b>Part 2: Reference</b> .....	<b>23</b>
<b>Section 1: Program Overview</b> .....	<b>25</b>
The VS Pascal Compiler .....	25
The VS Pascal Library .....	26
General Run-Time Routines .....	26
Input/Output Routines .....	27
Error Handling Routines .....	27
Conversion Routines .....	28
Mathematical Routines .....	28
String Routines .....	28
Storage Management Routines .....	29
The Debugging Library .....	29
Break Point Handling .....	29

<b>Section 2: Service Aids</b> .....	<b>31</b>
Special Compiler Options .....	31
Special Compiler Directives .....	32
Special Compiler Routines .....	32
Special Run-Time Options .....	33
VS Pascal Interactive Debugging Tool .....	33
<b>Appendix A. Compiler Options</b> .....	<b>35</b>
<b>Appendix B. Run-Time Options</b> .....	<b>37</b>
<b>Appendix C. Creating a Test Case</b> .....	<b>39</b>

---

## Figures

1. Flowchart of a VS Pascal Keyword String	4
2. VS Pascal Symptom Table	7
3. Statement Names as Modifier Keywords	9
4. Special Compiler Options	31
5. Special Compiler Directives	32
6. Special Compiler Routines	32
7. Special Run-Time Options	33
8. VS Pascal Compiler Options	35
9. VS Pascal Compiler Options	37





---

**Part 1: Guide**





---

## Introduction to Guide Information


VS Pascal Compiler and Library product failures can be described through the use of keywords. A keyword is a word or abbreviation used to describe a single aspect of a product failure. A set of keywords for a given problem is called a keyword string. The keyword string is used as a search argument in an IBM software support data base, such as the Software Support Facility (SSF) or the Early Warning System (EWS). To describe a product failure so you can search a data base, you must develop a set of keywords.

When the problem is described in the software support data base with the same set of keywords, the search yields matching descriptions of the problem and a correction is usually known.

If the problem is not listed, use the keywords to help prepare an APAR. Keywords are intended to ensure that any two people will identically describe the same type of problem caused by the same program error. For additional information on keywords and APAR preparation, see *Field Engineering Programming System General Information*.

---

## Keyword Usage



The first keyword identifies the product by its **Component Identification Number**. A search of a software support data base with this keyword alone would detect all reported problems for the entire licensed program. Each keyword added makes the search argument more specific, thereby reducing the types of problem descriptions to those shown in Figure 1 on page 4.

The procedures contained in the rest of this Guide will help you develop a full keyword string.

- “Component Identification Keyword” on page 7 lists the licensed program number used to identify VS Pascal products.
- “Type of Failure Keyword” on page 7 tells how to specify different types of failures.
- “Using the Keyword String as a Search Argument” on page 19 explains how the keyword string is used to search a software support data base.
- “Preparing an APAR” on page 21 explains what materials are needed to prepare an authorized program analysis report (APAR).

Figure 1 on page 4 illustrates the process of creating a keyword string.

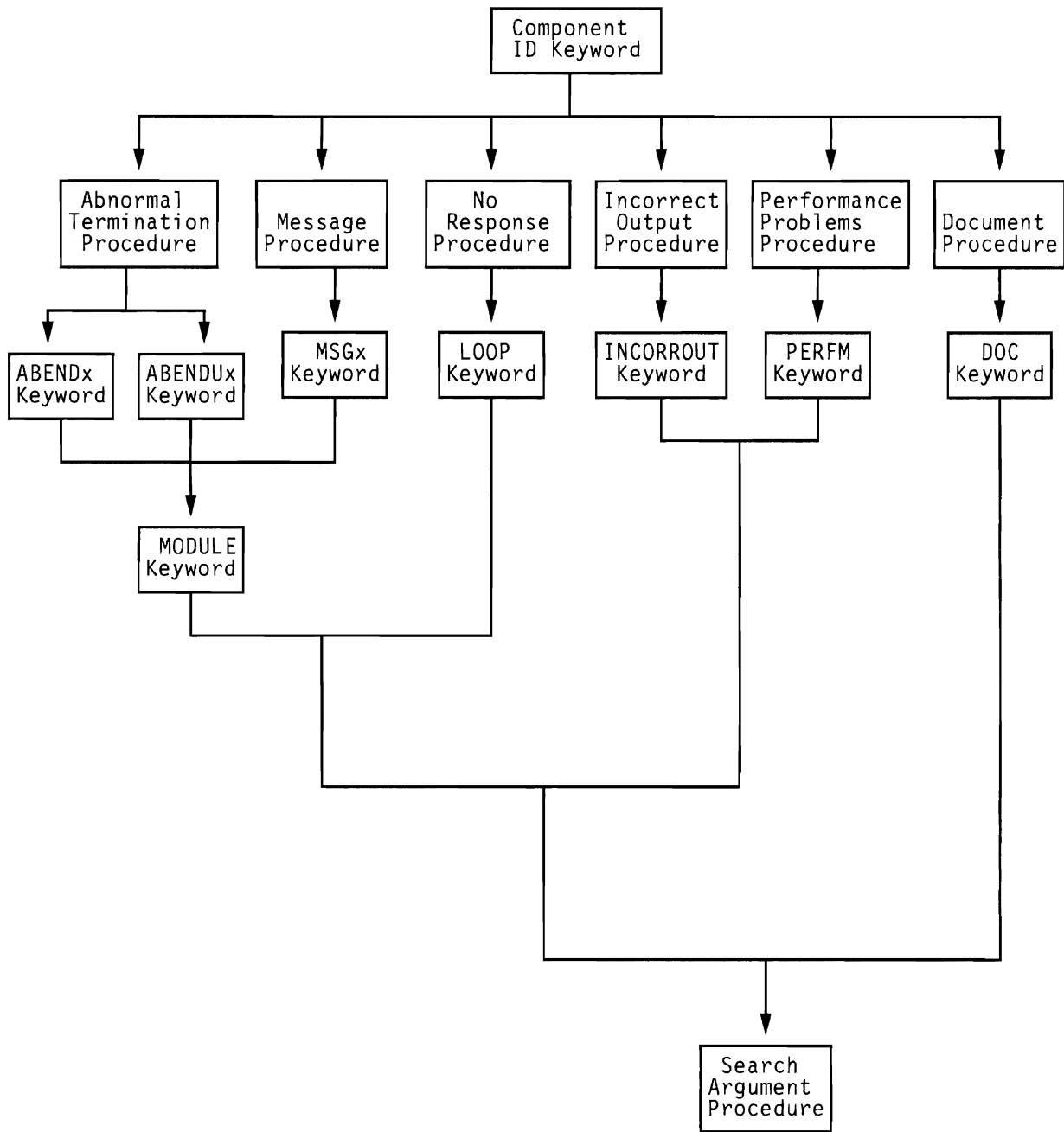


Figure 1. Flowchart of a VS Pascal Keyword String

## Diagnosis Procedure

Use these procedures to gather the diagnostic information needed to develop keyword search arguments for a software support data base.

1. Correct all problems diagnosed by VS Pascal messages and ensure that any messages previously generated have nothing to do with the problem being worked on. See *VS Pascal Application Programming Guide* for the VS Pascal messages.
2. If your program has been compiled with the NOOPTIMIZE option, use the VS Pascal Interactive Debugging Tool or Pascal static debug statements to identify a problem that occurs during run time. You can use the VS Pascal Interactive Debugging Tool for run-time analysis of your program and its error. Pascal static debug can isolate the problem and determine whether it is a user error, a compiler-produced code error, or a run-time library error.

For more information on the VS Pascal Interactive Debugging Tool, see “VS Pascal Interactive Debugging Tool” on page 33. An overview of the static debug statements and options is given in “Section 2: Service Aids” on page 31.

3. If the compilation fails using the OPTIMIZE option, use the VS Pascal interactive debugging tool or try to recompile the program with the NOOPTIMIZE option. If you are successful, compile the program at that level to bypass the problem until a fix is found and applied. Remember to note the conditions and options in effect when the failure occurred.
4. Determine if the program has been changed since it last compiled or executed successfully. If it has, examine the changes. If the error is occurring in the changed code and cannot be corrected, note the change that caused the error. If possible, retain copies of both the original and the changed programs.
5. If the program failed during execution, try executing the program with different run time options. Note any change from the previous execution.

In any program failure, try compiling with different compile options. Note any change from the previous compilation.

6. After the failure has been explored and identified, consider writing a small test case that re-creates the problem. This test case should help you to:
  - Distinguish between an error in the application program and an error in VS Pascal
  - Choose keywords that best describe the error
  - Pinpoint the problem

See Appendix C, “Creating a Test Case” on page 39, for details on constructing a test case from a failing program.

7. If the problem occurs during execution, specify the following options in the compilation, in addition to the options originally specified, to produce maximum diagnostic information.

SOURCE  
PXREF  
XREF(LONG)  
UCODE  
OUCODE  
LIST  
GOSTMT

8. Note the sequence of events that leads to the error condition. This information may be useful in developing a set of keywords, and will be needed if an APAR is required.
9. Begin developing the set of keywords, using the procedure in “Component Identification Keyword” on page 7.

## Developing a Keyword String

The following sections will help you select the correct keywords to describe a product failure and build a keyword string.

### Component Identification Keyword

The component identification number is the first keyword in the keyword string. It indicates the library within the software support data base that contains known problem descriptions for the product.

**566876701** is the component identification keyword for the VS Pascal Compiler and Library. **566871701** is the component identification keyword for the VS Pascal Library.

Continue the diagnosis procedure with “Type of Failure Keyword.”

### Type of Failure Keyword

From the list in Figure 2, select the symptom that best describes the problem. Then turn to the page referenced for that type of problem. If more than one keyword describes the problem, use the one that appears first in the list.

Symptom	Keyword	Description	Reference
Abnormal Termination	ABEND <sub>x</sub> ABENDU <sub>x</sub>	The compiler or user program has terminated abnormally without a message as described in this guide.	“Abnormal Termination Problems” on page 8
Message Problems	MSG <sub>x</sub>	A message indicates a compiler or user program error, or seems itself to be an error.	“Message Problems” on page 10
No Response	LOOP	The program seems not to be doing anything, or is doing something repetitively.	“No Response Problems” on page 14
Documentation Problems	DOC	The information in one of the VS Pascal publications is incorrect or missing.	“Documentation Problems” on page 15
Incorrect Output	INCORROUT	The output from the program is missing or invalid.	“Incorrect Output Problems” on page 16
Slow Response	PERFM	The performance of the program is degraded.	“Performance Problems” on page 17

Figure 2. VS Pascal Symptom Table



## Abnormal Termination Problems

Use the ABENDx keyword procedure when a program exception occurs:

- Within VS Pascal
- Within a VS Pascal-compiled program
- Whenever the VS Pascal compiler or a VS Pascal-compiled program terminates without a message

Do not use this keyword if termination was accompanied by a message with the prefix “AMP”. For those situations, refer to the procedure for “Message Problems” on page 10.

Do not use this keyword if termination was forced because too much time was spent in a wait state or in an endless loop. For those situations, refer to the procedure for “No Response Problems” on page 14.

### Procedure

If the problem occurs during installation, use the modifier INSTALL.

If the problem occurs during compile or run time:

1. Use the CMPL modifier if the problem occurs during the compilation of your program.
2. Use the EXEC modifier if the problem occurs during run time. If the problem occurs during execution of your program, there is a strong probability that it is a user error. Before continuing, verify that the problem is not the result of incorrect coding.
3. Determine with what compiler and run-time options the failure occurs. If it occurs at only some levels, indicate in the keyword string the options at which it does occur. Select the appropriate modifier keyword from the list shown in Appendix A, “Compiler Options” on page 35 and Appendix B, “Run-Time Options” on page 37.
4. Try to determine what statement is causing the problem. This may be done using the TRACE procedure in your code, or by using the VS Pascal Interactive Debugging Tool. In the debugging tool, either the TRACE command or the SET TRACE ON command may help. For compile-time abends, adding %WRITE directives to your code may help isolate the problem.

For a complete description of how to request and interpret a trace back map, see *VS Pascal Application Programming Guide*.

Use the following keywords as modifier keywords to describe the statement you think is causing the error.

AND	OF <sup>6</sup>
ARRAY	OR
ASSERT <sup>1</sup>	OTHERWISE <sup>1 7</sup>
BEGIN	PACKED
CASE	PROCEDURE
CONST	PROGRAM
CONTINUE <sup>1</sup>	RANGE <sup>1</sup>
DEF <sup>1</sup>	RECORD
DIV	REF <sup>1</sup>
DO <sup>2</sup>	REPEAT
DOWNTO <sup>3</sup>	RETURN <sup>1</sup>
ELSE <sup>4</sup>	SET
END <sup>5</sup>	SPACE <sup>1</sup>
FILE	STATIC <sup>1</sup>
FOR	THEN <sup>4</sup>
FUNCTION	TO <sup>3</sup>
GOTO	TYPE
IF	UNTIL <sup>8</sup>
IN	VALUE <sup>1</sup>
LABEL	VAR
LEAVE <sup>1</sup>	WHILE
MOD	WITH
NIL	XOR <sup>1</sup>
NOT	

Figure 3. Statement Names as Modifier Keywords

**Notes to Figure 3:**

- <sup>1</sup> LONGLVL(EXTENDED) only.
- <sup>2</sup> Use FOR, WHILE or WITH as keyword as appropriate.
- <sup>3</sup> Use FOR as keyword.
- <sup>4</sup> Use IF as keyword.
- <sup>5</sup> Use BEGIN, CASE or RECORD as keyword as appropriate.
- <sup>6</sup> Use ARRAY, CASE, FILE, SET or SPACE as keyword as appropriate.
- <sup>7</sup> Use CASE as keyword.
- <sup>8</sup> Use REPEAT as keyword.

**Example**

A sample keyword string showing an 0C1 abend in the library might look like this:

```
Component Identification: 566871701
Type of Failure:         ABENDOC1
Modifiers:               EXEC
                        GOTO
```

## Message Problems

Use the MSGx keyword procedure for any of these conditions:

- A message is issued indicating either a compiler or user program error.
- A message is issued under conditions that should not have caused it to be issued.
- A message contains invalid data or is missing data.

Do not use this procedure if you received a message as the result of an abnormal termination. In that case, see “Abnormal Termination Problems” on page 8.

**Syntax and Semantic Messages:** Each VS Pascal compiler syntax or semantic error message is identified by a string of three characters: *nnn*. *nnn* is the message number.

For the purpose of identification, consider each of these messages to have a message identifier of *AMPLnnn*.

**Compiler Messages:** Each VS Pascal compiler message (except for syntax and semantic error messages) is identified by a string of eight characters: *AMPpnnnc*.

where:

- AMP** is the message prefix identifying all VS Pascal compiler messages.
- p* is a letter representing the compiler phase issuing the message. *p* will be either L, O, or T.
- nnn* is the message number.
- c* is either an “I” for informational messages, a “W” for warning messages, an “E” for normal error conditions, or an “S” for severe error conditions.

**Library Messages:** Each VS Pascal library message is identified by a string of eight characters: *AMPXnnnc*.

where:

- AMPX** is the message prefix identifying all VS Pascal library messages.
- nnn* is the message number.
- c* is either an “I” for informational messages, an “E” for normal error conditions, or an “S” for severe error conditions.

**Debugging Messages:** Each VS Pascal debugging message is identified by a string of eight characters: *AMPDnnn*.

where:

- AMPD** is the message prefix identifying all VS Pascal library messages.
- nnn* is the message number.

### Procedure

1. Replace the *x* of *MSG<sub>x</sub>* with the complete message identifier but do not include the severity character (if any). Remember that syntax and semantic errors have an implied prefix of *AMPL*.
2. Use the name of the module and routine that caused the message to be issued as the module name keyword. You can use the trace back report to determine the failing routine.

To determine what module and routine caused the error, look at the top-most routine and module shown in the trace back report (as shown below). These will usually be the module and routine you report as failing.

```
AMPX036S Assertion failure checking error
AMPL999S COMPILER ERROR: NOTIFY VS PASCAL SUPPORT
      TRACE BACK OF CALLED ROUTINES
ROUTINE                               STMT AT ADDRESS IN MODULE
MAXLENFUNC                             25 0006119C AMPLSTRG
CALL                                    56 00060BD6 AMPLCALL
FACTOR                                  16 0006B482 AMPLFACT
TERM                                     3 00055F30 AMPLXPR
SIMPLEEXPRESSION                        7 00056EA6 AMPLXPR
EXPRESSION                              3 0005767E AMPLXPR
WRITE                                   52 0005A3F4 AMPLREAD
CALL                                    13 000608CA AMPLCALL
STATEMENT                               84 0003CBBE AMPLSTMT
BODY                                    92 0002180C AMPLMAIN
BLOCK                                   51 0002202E AMPLMAIN
PROGRAMME                               36 000225CE AMPLMAIN
<MAIN-PROGRAM>                          42 000248A4 AMPLMAIN
VSPASCAL                                0002020A
```

```
AMPX900S EXECUTION NOT ALLOWED TO CONTINUE
```

Your type of failure as shown above is “MSGAMPX036 MSGAMPL999 MSGAMPX900” and your modifiers are “AMPLSTRG MAXLENFUNC” in this case.

There are several cases where the top-most module and routine would not be reported. These instances are shown below.

For example, when compiling, ignore the top-most occurrences of the following modules and routines:

```
AMPLINSY ERROR
AMPLINSY WARNING
AMPLINSY INFORMATION
AMPLINSY NORMAL_MSG
AMPLINSY IMMEDIATE_MSG
AMPOMISC ERROR
AMPOMISC OVERFLOW
AMPTXMSC ERROR
AMPTXMSC FATAL
```

An example of the trace back report when compiling is shown below.

```

AMPT998S *** TRANSLATOR ERROR: NOTIFY VS PASCAL SUPPORT ***
BD: bad call
FAILED AT HALFMULT/<MAIN-PROGRAM>/1
  TRACE BACK OF CALLED ROUTINES
ROUTINE                STMT AT ADDRESS IN MODULE
FATAL                  9    00026244  AMPTXMSC
BD                     3    0003D264  AMPTXA
BXD                    7    0003D4C2  AMPTXA
XMPYI                  21   0005823C  AMPTXE
GENBINARY              24   0003EE36  AMPTGEN
EXPRESSION              37   0002DA50  AMPTRAN
EXPRESSION              9    0002D85C  AMPTRAN
EXPRESSION              9    0002D85C  AMPTRAN
EXPRESSION              6    0002D818  AMPTRAN
DOBB                   13   0002DFD2  AMPTRAN
TRANSLATE              54   0002E47E  AMPTRAN
<MAIN-PROGRAM>        7    00021EEE  AMPTMAIN
VSPASCAL                7    0002020A
  
```

Your type of failure as shown above is “MSGAMPT998” and your modifiers are “AMPTXA BD” in this case because AMPTXMSC/FATAL should be ignored.

Another example is when you execute with the MAINT run-time option. You should ignore only the top-most occurrences of the following modules and routines:

```

AMPXTRAC AMPXTRAC
AMPXCHKR AMPXERR
AMPXCHKR AMPXDIAG
AMPXCHKR AMPXCHKR
  
```

In addition, ignore all occurrences of routine AMPXMEMF. An example of the trace back report is shown below.

```

AMPX023E Exponent underflow exception
  TRACE BACK OF CALLED ROUTINES
ROUTINE                STMT AT ADDRESS IN MODULE
AMPXTRAC                9    0002E756  AMPXTRAC
AMPXERR                 81   000262F6  AMPXCHKR
AMPXDIAG                37   00026B82  AMPXCHKR
CHARX                   17   00022B00  AMPXPIC
PICTURE                  7    000242B8  AMPXPIC
<MAIN-PROGRAM>         1    000200F0  PICTBUG
AMPXMEMF                7    00025808
VSPASCAL                7    000203FA
  
```

Your type of failure as shown above is “MSGAMPX023” and your modifiers are “AMPXPIC CHARX” in this case because AMPXTRAC/AMPXTRAC, AMPXCHKR/AMPXERR and AMPXCHKR/AMPXDIAG should not be reported.

If, after ignoring the specified routines, the top-most routine is in your program (the module generally won't start with AMPX), then you should not report the failing module and routine.

An example of the trace back report for a protection exception is shown below.

```
AMPX014S Protection exception
      TRACE BACK OF CALLED ROUTINES
ROUTINE                               STMT AT ADDRESS IN MODULE
AMPXTRAC                               9   00028C46  AMPXTRAC
AMPXERR                                81   00021C3E  AMPXCHKR
AMPXDIAG                                37   000224CA  AMPXCHKR
<MAIN-PROGRAM>                          1   000200CA  TEMP
AMPXMEMF                                00021150
VSPASCAL                                000202F2
```

AMPX900S EXECUTION NOT ALLOWED TO CONTINUE

Your type of failure as shown above is “MSGAMPX014” and there are no modifiers because the error appears to have occurred in the main program of TEMP.

3. Determine with what compiler and run-time options the failure occurs. If it occurs at only some levels, indicate in the keyword string the options at which it does occur. Select the appropriate modifier keyword from the list shown in Appendix A, “Compiler Options” on page 35 and Appendix B, “Run-Time Options” on page 37.

### Examples

If the message 9 occurs during compilation, use the following keywords:

```
Component Identification: 566871701
Type of Failure:         MSGAMPL009
Modifier:                CMPL
```

If the message AMPX067E occurs during execution, use the following keywords:

```
Component Identification: 566871701
Type of Failure:         MSGAMPX067
Modifier:                EXEC
```

## No Response Problems

Use the LOOP keyword procedure when a program seems not to be doing anything or is doing something repetitively. However, if the problem looks like a WAIT, it is probably a system problem and you should follow your installation's procedures for resolution.

### Procedure

If the problem occurs during installation time, use the modifier INSTALL.

If the problem occurs during compile or run time:

1. If the failure occurs during execution, there is a strong probability that this is a user error. Carefully check your VS Pascal source program to be sure it does not contain an endless loop. Adding a WRITE statement and recompiling may help to detect such a program error.
2. If you are running in batch mode and the error is a system abend indicating not enough time, it is possible that inadequate time was allotted to compile or execute the program. Increase the time allotment and recompile or reexecute.
3. If the loop occurs during compilation, use the modifier CMPL. If the loop occurs during execution, use the modifier EXEC.
4. Determine with what compiler and run-time options the failure occurs. If it occurs at only some levels, indicate in the keyword string the options at which it does occur. Select the appropriate modifier keyword from the list shown in Appendix A, "Compiler Options" on page 35 and Appendix B, "Run-Time Options" on page 37.

### Example

If the compiler appears to loop, a set of keywords describing it would look something like this:

```
Component Identification: 566876701
Type of Failure:         LOOP
Modifier:                CMPL
```

## Documentation Problems

Use the DOC keyword procedure when a program problem appears to be caused by incorrect or missing information in one of the VS Pascal publications.

### Procedure

1. Locate the page where the problem occurs in the document, and prepare a description of the error and the problem it caused.
2. Decide whether this documentation problem is severe enough to cause lost time for other users.

If the problem is not severe, fill out the Reader's Comment Form attached to the back of that manual giving the problem description you developed. Be sure to include your name and return address so IBM can respond to your comments.

If the problem is likely to cause lost time for other users, continue creating your keyword string to determine whether IBM has a record of the problem. Should it be a new problem, you will be asked to submit a severity 4 (DOC) APAR.

3. Use the order number on the cover of the document, along with the DOC keyword as the type of failure keyword, but omit the hyphens. For example, instead of LY27-9525 (this book), enter LY279525. Your set of keywords would look like this so far:

```
Component Identification: 566876701  
Type of Failure:         DOC LY279525
```

4. If, after searching the IBM software support data base, you do not find a matching problem description. **return here to continue.**
5. Before assuming your search argument is not in the data base, you may want to try the search again, using the following format:

```
Component Identification: 566876701  
Type of Failure:         DOC LY279525
```

This method allows you to search for all problems reported for that document number.



## Incorrect Output Problems

Use the INCORROUT keyword procedure when output appears to be incorrect or missing, but the program terminates normally otherwise.

If the failure occurred during execution, there is a strong probability that this is a user error. Check your VS Pascal source program to be sure it is free of program errors. Adding a WRITE statement and recompiling or using the VS Pascal debugging tool can help you to detect such an error.

### Procedure

1. If the problem occurs during installation, use the modifier INSTALL. If the incorrect output occurs during compilation, use the modifier CMPL. If the incorrect output occurs during execution, use the modifier EXEC.
2. If you suspect incorrect or missing output from a compilation or execution that otherwise compiled or executed successfully, select a modifier keyword from the following list to describe the type of error in the output. These modifier keywords may also be used for an installation problem.

<b>Modifier</b>	<b>Type of Incorrect Output</b>
<b>MISSING</b>	Some expected output was missing.
<b>DUPLICATE</b>	Some records or data were duplicated, but not repeated endlessly (in that case, see “No Response Problems” on page 14).
<b>INVALID</b>	The proper amount of output appeared, but it was not what was expected.

3. If the failure occurred during compilation, select another modifier keyword from the following list to describe the portion of the output in which the error occurred.

<b>Modifier</b>	<b>Portion of Output in Error</b>
<b>ERROR</b>	Error summary of listing
<b>EXTSYM</b>	External symbol listing
<b>OBJECT</b>	Machine-language object program
<b>PXREF</b>	Page cross-reference listing
<b>SOURCE</b>	Source listing
<b>STAT</b>	Statistics, parameters and options summary
<b>TERMERR</b>	Console error listing
<b>TRACE</b>	Any trace backs produced
<b>XREF</b>	Cross-reference listing

4. Determine with what compiler and run-time options the failure occurs (this step is not appropriate for installation problems). If it occurs at only some levels, indicate in the keyword string the options at which it does occur. Select the appropriate modifier keyword from the list shown in Appendix A, “Compiler Options” on page 35 and Appendix B, “Run-Time Options” on page 37.

### Example

If you think the compiler has produced an incorrect cross-reference listing only when compiling with the XREF(LONG) option, your keyword string would look like this:

```
Component Identification: 566876701
Type of Failure:         INCORROUT
Modifiers:                CMPL
                        INVALID XREF
                        LONGXREF
```

## Performance Problems

Use the PERFM keyword procedure when a performance problem cannot be corrected by system tuning, and performance is below expectations documented in an IBM product publication.

### Procedure

1. Record the actual performance and the expected performance measurements for your system configuration. Note the order number and page of the IBM document that is the source of your performance expectations. You will be asked for this information if you contact the IBM support center; if you prepare materials for an APAR, you should also include this information in the error description.
2. If the performance problem occurs during compilation, use the modifier CMPL. If the performance problem occurs during execution, use the modifier EXEC.
3. Determine with what compiler and run-time options the failure occurs. If it occurs at only some levels, indicate in the keyword string the options at which it does occur. Select the appropriate modifier keyword from the list shown in Appendix A, “Compiler Options” on page 35 and Appendix B, “Run-Time Options” on page 37.

### Example

If a performance problem occurs during compilation, your keyword string would look like this:

```
Component Identification: 566876701
Type of Failure:         PERFM
Modifiers:                CMPL
```



---

## Using the Keyword String as a Search Argument

The following procedure explains how to use the set of keywords you've developed.

Searches against a software support data base will be most successful if VS Pascal diagnosticians follow these rules:

- Use only the keywords given in this book.
- Spell keywords exactly as they are presented in this book.
- Include all appropriate keywords in any discussion with IBM support personnel or in any written description of your problem.

If you decide to contact IBM to search the Software Support Facility, be prepared to supply your:

- Customer number
- The set(s) of keywords used to search the software support data base

---

### Procedure

If you do the search yourself, follow the procedure below:

1. Search the software support data base, using the full set of keywords you've developed. Given the following list:

Component Identification:	566876701
Type of Failure:	MSGAMPL999
	MSGAMPX059
Module Name:	AMPLINSY
Routine Name:	ENDOFFLINE
Statement Number:	2

your keyword string would be:

566876701 MSGAMPL999 MSGAMPX059 AMPLINSY ENDOFFLINE 2

2. Eliminate those APAR fixes that have already been applied to your system from the list of possible matches.
3. Compare each remaining APAR closing description with the symptoms of your current problem.
4. If a match is found, the problem's solution can be applied to your system by an application PTF. You can order the application PTF from the IBM Support Center. You may already have the program temporary fix (PTF) at your installation, and only need to install it from the correct program update (PUT) tape.

**Note:** For information on applying a specific PTF, refer to the cover letter associated with it.

5. If a match is not found, broaden the search, using the following techniques.
  - a. One by one, drop keywords from the right of your keyword string. (The keyword string was developed in a specific sequence to make this technique possible.)
  - b. Consider using a synonym as a replacement for a modifier keyword. The problem may have been entered into the data base using a slightly different expression than the now-recommended format.
6. If a match is not found using the preceding techniques, go to “Preparing an APAR” on page 21.

## Preparing an APAR

An APAR is an Authorized Program Analysis Report. This report identifies a problem caused by a suspected defect in a current unaltered release of a program.

You should prepare an APAR only after the preceding diagnostic procedures have been followed, and the keyword search proves unsuccessful.

---

### Procedure

1. If you think you need to submit an APAR, contact the IBM Support Center. Often you will already have been in contact with IBM while searching a data base and attempting to resolve the problem. In either case, be prepared to supply your:
  - Customer number
  - The keyword string(s) used to search the software support data base
2. The support personnel will review the problem with you, and give you an APAR number when you and they have agreed that an APAR is necessary.
3. You may be asked to supply various types of information to help prepare the APAR. Applicable items from the following list may be necessary:
  - Job control statements
  - Link-edit or loader map output
  - Compiler listings, including:
    - Source listing
    - Intermediate code listings
    - Object listing
    - Cross-reference listing
  - Any error trace backs
  - Any debugging output
  - Under VM/SP, any spooled CMS console output or special CMS EXECs
  - Machine-readable copy of the application program experiencing the problem
  - A description of the application program and the organization of its data sets



---

**Part 2: Reference**





## Section 1: Program Overview

This section describes the major functions of the VS Pascal Compiler and Library. This section is intended to give you a basis for communicating with an IBM program specialist about possible program failures.

---

### The VS Pascal Compiler

The VS Pascal compiler consists of four logical phases. The first compiler phase is the L phase, which transforms VS Pascal code into intermediate code. The second compiler phase is the O phase, which optimizes the intermediate code. The third compiler phase is the T phase, which transforms intermediate code into 370 machine code. Under MVS, there is an I phase which calls the L, O, and T phases.

U-code is the intermediate language that VS Pascal generates from the source code. VS Pascal uses U-code to communicate between each phase of the compiler.

The functions performed by each phase are as follows:

#### Phase L Functions:

- Parses the VS Pascal source code
- Checks for syntax and semantic errors
- Translates the source code into U-code
- Prints the source listings
- Records and prints errors on source listings

#### Phase O Functions:

- Optimizes U-code
- Processes Boolean expressions
- Performs range checking

#### Phase T Functions:

- Performs common subexpression elimination
- Translates U-code into machine code
- Prints pseudoassembler listing
- Prints external symbol dictionary (ESD)
- Prints compiler statistics

#### Phase I Function:

- Invokes the VS Pascal compiler under MVS

---

## The VS Pascal Library

The VS Pascal library is a collection of subprograms that are combined, as needed, with object modules produced by the VS Pascal compiler to form an executable load module.

As the compiler examines Pascal source statements and translates them into an object module, it identifies the need for certain library operations. At the corresponding points in the object module, the compiler inserts calls to the appropriate library routines. Copies of these library routines may be link-edited and made part of the load module, or be linked dynamically at execution time.

The VS Pascal library contains seven types of subprograms:

- General run-time routines
- Input/output routines
- Error handling routines
- Conversion routines
- Mathematical routines
- String routines
- Storage management routines

### General Run-Time Routines

The run-time routines provide miscellaneous functions such as program initialization and starting and stopping the MVS clock.

The VS Pascal compiler may be invoked under the following operating system environments:

- Multiple Virtual Storage/System Product (MVS/SP) Version 1, Release 3 or later (with or without the Time Sharing Option (TSO) or Time Sharing Option/Extended (TSO/E)).
- Multiple Virtual Storage/Extended Architecture (MVS/XA), which includes the following licensed programs:
  - MVS/System Product (MVS/SP) Version 2, Release 1.1 or later (with or without the Time Sharing Option or Time Sharing Option/Extended).
  - MVS/Extended Architecture (MVS/XA) Data Facility Product Release 1.
- Virtual Machine/System Product (VM/SP) Version 1, Release 4 or later (with or without the High Performance Option).

The compiler translates the VS Pascal source program into an object module. This object module is link-edited with all the appropriate run-time routines and other object modules it calls to form a load module. VS Pascal uses standard System/370 linkage conventions. See “Register and Storage Usage” in *VS Pascal Application Programming Guide*.

Upon invoking a VS Pascal program, the routine which is responsible for establishing the VS Pascal run-time environment gains control and performs the following functions:

1. Storage is obtained in which dynamic storage areas (DSA) are allocated and deallocated (the stack).
2. The VS Pascal communication work area (PCWA) is created and initialized.
3. The begin clock function (MVS only) is initiated.
4. An environment is set up to intercept program interrupts (fixed-point overflow, divide by zero, and so forth).
5. The main program is called.
6. Upon return from the main program, any open files are closed.
7. Acquired storage is freed.
8. The end clock function (MVS only) is called.
9. Control is returned to the system.

## Input/Output Routines

VS Pascal uses MVS access methods to implement its input/output facilities. VS Pascal file variables are associated with a data set by means of a ddname. Queued sequential access method (QSAM) is used for sequential data sets. The basic partitioned access method (BPAM) is used for partitioned data sets or MACLIBs (for VM/SP). The basic direct access method (BDAM) is used for random record access.

Each VS Pascal file is associated with a specific ddname. See the DDNAME open option in *VS Pascal Application Programming Guide* for information on ddnames.

At run-time, a data control block (DCB) is associated with every VS Pascal file variable. The DCB contains information describing specific attributes of the associated data set. For information on the DCB attributes, see “Data Set DCB Attributes” in *VS Pascal Application Programming Guide*.

## Error Handling Routines

VS Pascal detects many kinds of errors during the program execution. Upon detection of an error, the VS Pascal run-time library will provide error handling.

Certain errors are considered fatal by the run-time library. Examples of these errors are operation exceptions and protection exceptions. When a fatal error occurs, VS Pascal will produce a message describing the error, display a trace back, and terminate the program.

Other errors such as checking errors will not stop program execution. The user must determine the extent to which the nonfatal errors affect program results. VS Pascal performs the following actions when a nonfatal error occurs:

1. A message describing the error is produced. The message is displayed on the terminal if the user is executing in VM/CMS or TSO, or written to the SYSPRINT data set otherwise.
2. If the program was compiled and link-edited with the DEBUG option, and the program was not executed with the DEBUG run-time option, then a symbolic

dump of the variables in the procedure experiencing the error will be produced. The dump is displayed on the terminal if the user is executing under VM/CMS or TSO, or written to the SYSPRINT data set otherwise.

3. If the program was compiled and link-edited with the DEBUG option, and the program was executed with the DEBUG run-time option, then the interactive debugging tool will be invoked as if a break point had been encountered.

VS Pascal will allow a specific number of nonfatal errors to occur before the program is terminated. This number is set by the ERRCOUNT run-time option. The default is 20.

VS Pascal also provides a mechanism for the user to gain control when a run-time error occurs. When such an error occurs, a procedure called ONERROR is called to perform any necessary action before generating a diagnostic message. A default ONERROR routine (which does nothing) is provided in the VS Pascal library.

The user may write a version of ONERROR and declare it as an EXTERNAL procedure. The procedure will be invoked when an error occurs. An example of this is shown in *VS Pascal Application Programming Guide*.

VS Pascal also provides four error handling routines to handle different classes of run-time errors:

- AMPXCHKR—intercepts run-time checking errors.
- AMPXDIAG—intercepts program exceptions.
- AMPXERR—general run-time error handler.
- AMPXIOER—I/O error intercept routine.

## Conversion Routines

There are several places where VS Pascal must perform data conversions. They take place when you are doing I/O on TEXT files and when you use the READSTR and WRITESTR routines.

## Mathematical Routines

The predefined functions are provided as VS Pascal functions. The VS Pascal compiler changes the user provided name (such as SIN) to an internal name (such as AMPXSIN).

## String Routines

The predefined functions and procedures are provided as VS Pascal functions and procedures. The VS Pascal compiler changes the user provided names (such as SUBSTR) to an internal name (such as AMPXSUBS). Several routines are provided in two forms: long and short. The short form is always used if possible. In order to use the short form, the VS Pascal compiler must determine that the resulting string will be less than 2048 bytes long. If the size can't be limited by compiler analysis, the compiler uses the long form which passes the results through the heap.

## Storage Management Routines

All VS Pascal dynamic variables are allocated from a pool of storage called the heap. The NEW function generates a call to the internal procedure AMPXNEW (or AMPXVNEW for pointers to variant records). This procedure allocates storage within a heap. If a heap has not yet been created, NEW will obtain storage from the operating system to create a heap.

The DISPOSE procedure generates a call to the procedure AMPXDISP. This procedure deallocates the heap storage acquired by a previous call to AMPXNEW.

The MARK procedure generates a call to the procedure AMPXMARK. This procedure creates a new heap from which subsequent calls to AMPXNEW will obtain storage.

The RELEASE procedure generates a call to the procedure AMPXRLSE. This procedure frees a heap that was previously created via the AMPXMARK procedure. Subsequent calls to AMPXNEW will obtain storage from the heap which was active before the call to AMPXMARK.

Data required by the run-time environment will be allocated in a separate heap controlled by AMPXINew and AMPXIDSP. Thus the I/O control blocks and debugging tables are in a distinct area.

---

## The Debugging Library

The interactive debugging tool is invoked by setting a compiler option at compile time. This option sets up the debugging tables and other necessary data needed to run the debugging tool. When the load module is executed, the DEBUG run-time option will activate the debugging environment.

### Break Point Handling

The debugging break points are handled by setting the address of the desired break point into a break table after finding the correct routine to set the break point in. The first two bytes of the actual code statement are stored so the original code can be restored. An illegal operation code is then placed into the code. A branch back to the program is set in the break table to know where to return after the break point is hit. When the illegal operation code is found, the break table is searched to check if that is an actual break point or a source program error. If it is a break point, then the program execution is stopped before the statement is executed and the debugging command prompt is given.



---

## Section 2: Service Aids

For more comprehensive information on the following statements and options, refer to *VS Pascal Language Reference* and *VS Pascal Application Programming Guide*.

**Note:** Any of the following items which are not listed in the above manuals are for diagnostic purposes only. IBM will not accept APARs on these items. These items are marked “FOR DIAGNOSTIC PURPOSES ONLY.”

---

### Special Compiler Options

The VS Pascal compiler contains many options which may help in isolating problems in the compiler or a user program. This list is shown in Figure 4.

Compiler Option	Description
CHECK	Enables run-time checking for a user program.
COMPILER	Allows the compiler to be invoked with certain run-time options. FOR DIAGNOSTIC PURPOSES ONLY.
DEBUG	Allows the VS Pascal interactive debugging tool to be used with a program.
LIST	Generates a pseudoassembler listing for the user program being compiled.
LOG	Generates a log of processing in the final compiler pass (code generation). FOR DIAGNOSTIC PURPOSES ONLY.
OLOG	Generates a log of processing in the middle compiler pass (intermediate code optimization). FOR DIAGNOSTIC PURPOSES ONLY.
OUCODE	Generates a listing of the optimized intermediate code. FOR DIAGNOSTIC PURPOSES ONLY.
UCODE	Generates a listing of the intermediate code produced by the compiler. FOR DIAGNOSTIC PURPOSES ONLY.
WRITE	Allows %WRITE statements to output messages while a program is being compiled.

Figure 4. Special Compiler Options



## Special Compiler Directives

The VS Pascal compiler contains special directives which may help in isolating problems in the compiler or a user program. Figure 5 shows a list of these compiler directives.

Compiler Directive	Description
%CHECK	Enables run-time checking for a portion of a user program.
%LIST	Generates a pseudoassembler listing for a portion of the user program being compiled.
%WRITE	Issues a message during compilation of a program. This is useful in isolating exactly where the VS Pascal compiler is failing.

Figure 5. Special Compiler Directives

## Special Compiler Routines

The VS Pascal compiler contains special routines which may help in isolating problems in a user program. Figure 6 shows a list of the compiler routines.

Compiler Routine	Description
ADDR	Returns the address of a given variable.
AMPXMDMP	Dumps the storage pools currently in use. This is declared as: <pre>PROCEDURE AMPXMDMP; EXTERNAL;</pre> and may be called from a user program. <b>FOR DIAGNOSTIC PURPOSES ONLY.</b>
HBOUND	Returns the maximum subscript of an array variable or type.
HIGHEST	Returns the maximum value of a variable or type.
LBOUND	Returns the minimum subscript of an array variable or type.
LOWEST	Returns the minimum value of a variable or type.
ONERROR	Allows run-time errors to be trapped and handled specially.
SIZEOF	Returns the size of a given variable or type.
TRACE	Produces a trace back of the program executing.

Figure 6. Special Compiler Routines

---

## Special Run-Time Options

VS Pascal allows several special run-time options which may help in isolating problems in a user program. Figure 7 shows a list of these run-time options.

Run-Time Option	Description
COUNT	Provides a statement execution count for a program.
DEBUG	Activates the VS Pascal interactive debugging tool
ERRCOUNT	Terminates program execution after a specified number of errors have occurred.
ERRFILE	Directs run-time error messages to a specified output device.
MAINT	Includes all run-time library routines called in any trace backs.
SETMEM	Initializes local variable storage to a common value at each routine invocation.

Figure 7. Special Run-Time Options

---

## VS Pascal Interactive Debugging Tool

You can use the VS Pascal interactive debugging tool to help determine the statement causing an error. With the VS Pascal interactive debugging tool, you can suspend program execution, continue execution, examine variable values, display storage, trace program execution, view a program trace back, count statement execution and issue system commands (CMS only). You can also use the VS Pascal interactive debugging tool to debug optimized code, with some restrictions. For more information on the debugging tool, see *VS Pascal Application Programming Guide*.

There is one diagnostic command contained in the VS Pascal interactive debugging tool—DG. DG will display the values contained in general registers 12 and 13. This is FOR DIAGNOSTIC PURPOSES ONLY.



## Appendix A. Compiler Options

Use the modifier keywords shown below to identify compiler options in the keyword string.

<b>Option</b>	<b>Modifier Keywords</b>
CHECK or NOCHECK	CHECK, NOCHECK
DEBUG or NODEBUG	DEBUG, NODEBUG
DDNAME	DDNAME
FLAG	FLAG
GOSTMT or NOGOSTMT	GOSTMT, NOGOSTMT
GRAPHIC or NOGRAPHIC	GRAPHIC, NOGRAPHIC
LANGLVL (EXTENDED ANSI83)	LANGLVL, EXTENDED, ANSI83
LINECOUNT (number)	LINECOUNT
LIST or NOLIST	LIST, NOLIST
MARGINS (number,number)	MARGINS
OPTIMIZE or NOOPTIMIZE	OPT, NOOPT
PAGEWIDTH (number)	PW
PXREF or NOPXREF	PXREF, NOPXREF
SEQUENCE (number,number) or NOSEQUENCE	SEQ, NOSEQ
SOURCE or NOSOURCE	SOURCE, NOSOURCE
STDFLAG	STDFLAG
WRITE or NOWRITE	WRITE, NOWRITE
XREF (LONG SHORT) or NOXREF	LONGXREF, SHRTXREF, NOXREF

Figure 8. VS Pascal Compiler Options



## Appendix B. Run-Time Options

Use the modifier keywords shown below to identify run-time options in the keyword string.

<b>Option</b>	<b>Modifier Keywords</b>
COUNT	RCOUNT
DEBUG	RDEBUG
ERRCOUNT (number)	RERRCNT
ERRFILE (ddname)	RERRFILE
HEAP = number	RHEAP
MAINT	RMAINT
NOCHECK	RNOCHECK
NOSPIE	RNOSPIE
STACK = number	RSTACK
SETMEM	RSETMEM

Figure 9. VS Pascal Compiler Options



---

## Appendix C. Creating a Test Case

To create a small test case from a large program that appears to be failing, these suggestions may be used:

- Remove any unreferenced identifiers. These may be found by using the XREF(LONG) option.
- If the program is failing at compile time, remove any code that has not been processed at the time of failure (except that necessary to ensure the syntactic and semantic validity of the program).
- If the program is failing at run time, remove any code that has not been executed (and any references to that code that would cause a syntactically or semantically invalid program). This may be found by using the COUNT run-time option.
- If the failure is occurring in a procedure or function, try placing the failing code in the main program.
- If the failure is occurring at compile time in a procedure or function, try removing all code and declarations from any other routines (namely, just keeping the procedure header and BEGIN/END block).
- If the failure is occurring in a structured statement (for example, an IF, CASE, WITH, FOR, WHILE or REPEAT statement), try placing the failing statement in the mainline code.
- Remove any code that appears to be unrelated to the failure and is not necessary for syntactic or semantic validity.
- If an error occurs using structured variables, try using scalar variables.
- If your program uses %INCLUDE statements, put all the %INCLUDE member code in the main file.
- If your program uses segment units, put all the declarations which are not in the program unit into the program unit.

If the program still fails in the same way after completing one of these steps, try others in the above list, making your program as small as possible but still causing the failure. Save the last failing test case.

**Note:** These suggestions are not exhaustive. There may be other ways to create a smaller program, but the above list has been found to yield good results.





VS Pascal  
Diagnosis Guide and Reference

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

**Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.**

If you have applied any technical newsletters (TNLs) to this book, please list them here: \_\_\_\_\_

Chapter/Section \_\_\_\_\_

\_\_\_\_\_ Page No. \_\_\_\_\_

Comments:

Note: Staples can cause problems with automated mail sorting equipment.  
Please use pressure sensitive or other gummed tape to seal this form.

If you want a reply, please complete the following information.

Name \_\_\_\_\_ Phone No. (\_\_\_\_) \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_  
\_\_\_\_\_

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Fold and tape

Please do not staple

Fold and tape



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation  
P.O. Box 50020  
Programming Publishing  
San Jose, California 95150



Fold and tape

Please do not staple

Fold and tape



VS Pascal  
Diagnosis Guide and Reference

This manual is part of a library that serves as a reference source for system analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

**Note: Do not use this form to request IBM publications. If you do, your order will be delayed because publications are not stocked at the address printed on the reverse side. Instead, you should direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.**

If you have applied any technical newsletters (TNLs) to this book, please list them here: \_\_\_\_\_

Chapter/Section \_\_\_\_\_

\_\_\_\_\_ Page No. \_\_\_\_\_

Comments:

Note: Staples can cause problems with automated mail sorting equipment.  
Please use pressure sensitive or other gummed tape to seal this form.

If you want a reply, please complete the following information.

Name \_\_\_\_\_ Phone No. (\_\_\_\_) \_\_\_\_\_

Company \_\_\_\_\_

Address \_\_\_\_\_

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

Reader's Comment Form

Fold and tape

Please do not staple

Fold and tape



NO POSTAGE  
NECESSARY  
IF MAILED  
IN THE  
UNITED STATES

**BUSINESS REPLY MAIL**  
FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE

IBM Corporation  
P.O. Box 50020  
Programming Publishing  
San Jose, California 95150



Fold and tape

Please do not staple

Fold and tape

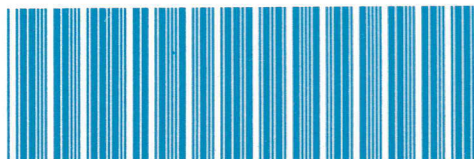




VS Pascal  
Diagnosis Guide  
and Reference

Restricted Materials of IBM  
Licensed Materials-Property of IBM  
© Copyright IBM Corp. 1987  
File Number S370-37

LY27-9525-00



Printed in U.S.A.