Compaq C V6.4 for OpenVMS VAX Release Notes

February 5, 2001

© 2001 Compaq Computer Corporation.

COMPAQ, the Compaq logo, VAX, Alpha, VMS, and OpenVMS are registered in the U.S. Patent and Trademark Office. Tru64 is a trademark of Compaq Information Technologies Group, L.P. in the United States and other countries. Microsoft and Visual C++ are trademarks of Microsoft Corporation. UNIX is a trademark of The Open Group in the United States and other countries. All other product names mentioned herein may be trademarks of their respective companies.

Confidential computer software. Valid license from Compaq required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Compaq shall not be liable for technical or editorial errors or omissions contained herein. The information in this document is provided as is without warranty of any kind and is subject to change without notice. The warranties for Compaq products are set forth in the express limited warranty statements accompanying such products. Nothing herein should be construed as constituting an additional warranty.

Contents

1	Product name change: DEC C -> Compaq C	1
2	News for Application Developers	1
3	Introduction	1
4	Redistribution of Run-Time Libraries	2
4.1	Redistribution of the AACRT060.A Run-Time Components Kit	2
4.2	Redistribution of the DECC$CRTL.OLB Object Library	3
5	Installation Notes	3
5.1	The startup procedure	3
5.2	The CC command	3
5.3	The Compaq C/C++ Run Time Components kit	4
5.4	Installing both Compaq C and VAX C	4
5.5	Installing both Compaq C and Compaq C++	5
6	Installing and Using Multiple Compiler Versions	5
6.1	Displaying and selecting the compiler version	5
6.2	Side effects and restrictions on multiple versions	6
6.3	Installation Procedure Changes	7
6.4	Sample installation fragment	7
7	Compaq C for OpenVMS VAX Release Notes	8
7.1	New Features in Compaq C V6.4	8
7.2	New Features in Compaq C V6.2	12
7.3	New Features in DEC C V6.0	16
7.4	New Features in DEC C V5.7	18
7.5	New Features in DEC C V5.6	20
7.6	New Features in DEC C V5.5	20
7.7	New Features in DEC C V5.3	21
7.8	Changes to #include processing in DEC C V5.3	22
7.9	New Features in DEC C V5.2	26
7.10	Changes in DEC C RTL Header Files for V5.2 of DEC C/C++	28
7.11	New Features in DEC C V5.0	32
7.12	New Features in DEC C V4.0	38
7.13	Problems fixed in Compaq C V6.4	39
7.14	Problems fixed in Compaq C V6.2	40
7.15	Problems fixed in DEC C V6.0	40
7.16	Problems fixed in DEC C V5.7	41
7.17	Problems fixed in DEC C V5.6	42
7.18	Problems fixed in DEC C V5.5	43
7.19	Problems fixed in DEC C V5.3	46
7.20	Problems fixed in DEC C V5.2	49
7.21	Problems fixed in DEC C V5.0	52
7.22	Problems Fixed in V4.0-01	55
7.23	Restrictions and known bugs	57

Tables

1 New DEC C V5.2 Header Files 28

1 Product name change: DEC C -> Compaq C

Note that the change in the product name from DEC C to Compaq C reflects only the acquisiton of the corporation, and meeting the new requirements for product branding. The technical content of the compiler has not changed, and in particular the use of the string DEC or DECC in the software itself (e.g. names of predefined macros, command line qualifiers, filenames, symbol prefixes, etc) has not changed.

We have tried to use the name Compaq C when referring to the new version, or when making general references to the product that might encompass both old and new versions. We try to use the name DEC C only where an older version is specifically being designated or where the use involves software identifiers or strings that are not changing. There are bound to be instances that inadvertently use the "wrong" name. The important thing to know is that from a technical point of view, this is a new version of the DEC C compiler and the name change does not signify a change in technology that could introduce incompatibilites with previous versions of DEC C.

Similarly for DEC C++, which is being renamed to Compaq C++ in V5.7 for OpenVMS VAX.

2 News for Application Developers

There is a new product called "OpenVMS Enterprise Toolkit for Visual Studio" that supports remote development for OpenVMS using a Windows/NT desktop running a Visual Studio interface that is now shipping. See http://www.openvms.digital.com/commercial/et/et_index.html for more information.

A separate no-cost package called "Application Developer's Package for C/C++", which provides information and examples of building 3-tier client/server applications on Windows NT and OpenVMS Alpha systems also remains available. See http://www.openvms.digital.com/openvms/products/appdevpkg/ for more information.

3 Introduction

This document contains the release notes for Compaq C V6.4 for OpenVMS VAX. A significant capability provided by this compiler kit is the ability to build applications using new Compaq C Runtime Library features, and deliver those applications on earlier versions of OpenVMS when necessary. This capability is provided by two additonal files installed by this kit in SYS$LIBRARY: DECC$CRTL.EXE, and DECC$CRTL.OLB. The file SYS$LIBRARY:DECC$CRTL.README provides important detailed information on how such applications can be built. The technique described there requires recompilation using both the compiler and the CRTL header files provided with Compaq C V6.4 before relinking.

Also note that most of the Compaq C documentation is now being provided in html format, including help and release notes, as well as all of the Compaq C manuals. The URL to access the online help with a local browser is given at the start of the "$ help cc/decc" command. For additional information about the OpenVMS VAX version of Compaq C, see also:

- The *Compaq C User's Guide for OpenVMS Systems*

- The *Compaq C Migration Guide for OpenVMS VAX Systems*
- Enter the command HELP CC/DECC

For additional information about the Compaq C language and its supported library routines, see also:

- The *Compaq C Language Reference Manual*
- The *Compaq C Run-Time Library Reference Manual for OpenVMS Systems*

The Compaq C media contains three VMSINSTAL kits; in order: AACRT060.A CCnnn.A, CCnnn.B.

AACRT060 is the Compaq C/C++ Run-Time Components for OpenVMS VAX V6.0 kit. It includes image files and libraries for the Compaq C RTL, Compaq C++ Class Library and DECthreads. These files are updated versions of the binaries provided with V6.0 of OpenVMS VAX. Additional files useful for building applications which use both the VAX C RTL and Compaq C RTL are also included. This kit must *not* be installed on V6.1 of OpenVMS VAX systems or later. Starting with V6.1 of OpenVMS VAX, all of the components and functionality contained on the AACRT060 kit were provided with the base operating system. Please see the *Compaq C/C++ for OpenVMS VAX Run-Time Components Reference and Installation Manual* in that kit's documentation for installation and usage information, and the following section regarding redistribution of libraries.

The CCnnn.A and CCnnn.B savesets contain the compiler and documentation kits, where the nnn encodes the compiler version number.

4 Redistribution of Run-Time Libraries

This section describes how developers who need to redistribute run-time library components to their users' systems are required to do so. The Compaq C Version 6.2 kit contains two run-time library components that you may need to redistribute:

- Compaq C/C++ Run-Time Components for OpenVMS VAX V6.0 (AACRT060.A)
- Compaq C Run-Time Object Library (DECC$CRTL.OLB)

Redistribution of other components on the Compaq C Version 6.2 kit is prohibited. The redistribution rights set forth in the Software Product Description do not apply to the DECC$CRTL.EXE or DECC$CRTL.README files distributed with this kit.

4.1 Redistribution of the AACRT060.A Run-Time Components Kit

The Compaq C/C++ Run-Time Components kit has been engineered to allow for redistribution with any application that uses one or more of these components. Redistributing the Compaq C/C++ Run-Time Components kit requires that the entire kit and its documentation be shipped to the remote site receiving any such application. See the *Compaq C/C++ Run-Time Components for OpenVMS VAX Reference and Installation Guide* for detailed installation and redistribution instructions. Note that applications targeted to V6.1 and later versions of OpenVMS have no need of this kit.

4.2 Redistribution of the DECC$CRTL.OLB Object Library

Redistribution of this library is only required by those applications that need the features supplied by the library and need to be linked during or after installation on an end user target system. If you link your application against this library and ship either a shareable or executable image to your customers, then redistribution of the object library is not necessary. The linking process of your application causes those library modules to be incorporated into your resultant image.

There are two options that can be used to redistribute the DECC$CRTL.OLB object library. The options are based on whether or not the library is needed after the installation is completed.

The first option is for applications that link during installation, but have no need for the object library once installation is completed. For that set of developers, we recommend placing DECC$CRTL.OLB on your kit, but to link using the copy in VMI$KWD and not issue a PROVIDE_FILE option which would move this file onto the system. In other words, the object library resides only on your kit, is used during installation to link your application, but is not placed onto the end user system.

The second option is for applications that do need the object library after installation is completed. For this class of applications, the object library should be placed in a product-specific location on the target system and not in SYS$LIBRARY. The contents of this object library must not be inserted into the SYS$LIBRARY:STARLET.OLB library.

5 Installation Notes

For additional information about installing the kit, and for the location of installed components of this product, refer to the *Compaq C Installation Guide* accompanying these release notes.

5.1 The startup procedure

After installing Compaq C for OpenVMS VAX on a cluster, you must invoke the command @SYS$STARTUP:CCXX$STARTUP.COM on each node of the cluster.

5.2 The CC command

If you have VAX C installed on your system, the installation procedure will ask if you want the CC command to default to VAX C or Compaq C. The default response is Compaq C. If you choose the default response and let CC invoke Compaq C instead of VAX C, you may find that:

- Command files that expect CC to invoke VAX C may fail.
- Other layered products that expect CC to invoke VAX C may fail during installation or IVP.

Typically, this occurs when the code being compiled depends upon features of VAX C that are not present in the default "relaxed ANSI" mode of Compaq C. For example, the code may not comply with the C standard. If you wish to have CC invoke Compaq C, you may want to inform users that they can define a global symbol:

```
$ cc == "cc/vaxc"
```

that will allow their command procedures to continue to work.

So that users and other produces can determine the default for the CC command, Compaq C defines the logical DECC$CC_DEFAULT. This logical has the value /VAXC if CC invokes VAX C and the value /DECC if CC invokes Compaq C. This logical does not control the behavior of the CC command. That is determined at installation time.

5.3 The Compaq C/C++ Run Time Components kit

On versions of OpenVMS VAX prior to V6.1 only, Compaq C requires the previous installation of the Compaq C/C++ Run Time Components kit. The Compaq C compiler will not install if the Compaq C/C++ Run Time Components kit is not installed on such systems. See the Compaq C/C++ Run Time Components kit release notes for more information. On versions of OpenVMS VAX from V6.1 onward, the run time components needed by Compaq C are shipped as part of the OpenVMS distribution and the Compaq C/C++ Run Time Components kit must not be installed on those systems.

When installing or upgrading to OpenVMS V5.* on a system that has Compaq C installed on it, certain files will be superseded by old versions that are incompatible with the newer versions contained in the Compaq C/C++ Run-Time Components kit. Therefore, the AACRT kit should be reinstalled after any update of OpenVMS. It is not necessary to reinstall the AACRT kit after updating to OpenVMS V6.1. It is not necessary to reinstall the AACRT kit prior to future upgrades of Compaq C.

The AACRT060 kit provides a version of DECthreads for use on OpenVMS VAX V5.5-2 through OpenVMS VAX V6.0. This version of DECthreads cooperates with the Compaq C RTL. Some existing applications may not run correctly if they use both the new version of DECthreads and the OpenVMS V5.* versions of SYS$LIBRARY:VAXCRTL*.EXE. If you upgrade to VMS 6.1, the system versions of the DECthreads headers are newer than the version which ship on this kit. Therefore, you will need to reinstall Compaq C V6.4 so that the pre-6.1 DECthreads headers will be removed from DECC$RTLDEF.TLB.

You must not redistribute individual components contained in the AACRT kit. See Section 4 for information on redistributing run-time components.

Finally, note that the Compaq C RTL was substantially enhanced in versions 6.2 and 7.0 of OpenVMS VAX. Many new API's were added for XPG4 localization in V6.2, and for UNIX compatibility and to support the ISO C addendum in V7.0 of OpenVMS. These features and other RTL improvements incorporated with them are not available on pre-6.2 versions of OpenVMS. Since the Compaq C RTL is part of the OpenVMS operating system, to benefit from the continuing feature and performance enhancements in the Compaq C RTL it will generally be necessary to upgrade your version of OpenVMS. However, in cases where upgrading is not an option, it is possible to build Compaq C images that use some newer features of the Compaq C RTL that will run on earlier versions of the operating system that lack those RTL features. This involves special compilation, and linking against an object library as described in SYS$LIBRARY:DECC$CRTL.README.

5.4 Installing both Compaq C and VAX C

If you wish to use both VAX C V3.* and Compaq C on a system, you must install Compaq C after installing VAX C. Installing Compaq C modifies the CC command so that CC/DECC invokes the Compaq C compiler and CC/VAXC invokes the VAX C compiler. If you install VAX C after Compaq C, you will not be able to activate Compaq C with the CC command. If you do install VAX C after installing Compaq C, you must then reinstall Compaq C.

5.5 Installing both Compaq C and Compaq C++

If you wish to use both DEC C++ V1.3 (or higher, including Compaq C++ V5.7) for OpenVMS VAX and Compaq C V6.4 for OpenVMS VAX, you may install the C and C++ kits in either order. Beginning with DEC C V4.0 and DEC C++ V1.3, the two kit installation procedures were made to be compatible.

If you wish to use both Compaq C V6.4 for OpenVMS VAX and a version of DEC C++ for OpenVMS VAX prior to V1.3, you must install Compaq C after installing DEC C++.

For information on interactions between the DEC C and DEC C++ installations specific to V5.3, see Section 7.8 in these release notes.

6 Installing and Using Multiple Compiler Versions

Compaq C V6.4 provides initial support for installing and using multiple versions of the compiler on the same node.

During installation of V6.4, if a V6.0 or V6.2 version of the compiler is already installed, you will be given the opportunity to preserve that compiler rather than overwrite it. If you choose to preserve the currently-installed compiler, you will then be given an opportunity to keep the currently-installed compiler as the system default and install the new compiler as an alternate. By default, preserving the currently installed system compiler is performed by making it an alternate compiler, and installing the new compiler as the system default.

The choice to use an alternate compiler instead of the installed system compiler can be made by users, by running a command procedure that changes the behavior of the cc command for the process that invokes it.

6.1 Displaying and selecting the compiler version

This kit provides two command procedures to display and control which Compaq C compiler is used by a process.

- SYS$SYSTEM:DECC$SHOW_VERSIONS.COM

  This procedure displays what Compaq C compilers are available on the system, together with their version numbers. It also displays which compiler is the default for the current process. The procedure takes an optional version number argument. An example:

```
$ @sys$system:decc$show_versions.com

The following  Compaq C compiler(s) are available in SYS$SYSTEM:

Filename                      Version
--------------------------------------
DECC$COMPILER.EXE             V6.4-005
DECC$COMPILER_T06_04-001.EXE  T6.4-001
DECC$COMPILER_V06_00-001.EXE  V6.0-001    Process Default
DECC$COMPILER_V06_02-003.EXE  V6.2-003

$ @sys$system:decc$show_versions.com V6.0

The following  Compaq C compiler(s) are available in SYS$SYSTEM:

Filename                      Version
--------------------------------------
DECC$COMPILER_V06_00-001.EXE  V6.0-001    Process Default
```

- SYS$SYSTEM:DECC$SET_VERSION.COM

This procedure sets up process logicals that point to an alternate Compaq C compiler in sys$system, or it removes the process logicals to revert back to using the default system compiler. The procedure takes one argument, a version number or "SYSTEM" (if no arguments are specified you will be prompted). The SYSTEM argument selects the installed system compiler, which is the one displayed with the filename DECC$COMPILER.EXE in the output of decc$show_versions.com. Alternate compilers are shown in that output with their version number appended to the simple filename. For example, the V6.2-003 compiler is given the name: DECC$COMPILER_V06_02-003.EXE .

To select a compiler, pass a full ident string or enough of the ident string to be unique. For example: to select the V6.2-003 compiler from our list above we can pass V6.2-003 or V6.2 to the decc$set_version.com routine. However to select a 6.4 compiler a full ident string would be required to distinguish between the T6.4-001 and the V6.4-005 compiler.

```
@sys$system:decc$set_version.com V6.2-003

@sho logical DECC$*
(LNM$PROCESS_TABLE)
  "DECC$COMPILER" =
    "SYS$COMMON:[SYSEXE]DECC$COMPILER_V06_02-003.EXE"
  "DECC$MSG_SHR" =
    "SYS$COMMON:[SYSMSG]DECC$MSG_SHR_V06_02-003.EXE"

$ @decc$set_version.com 6.4

The following V6.4 Compaq C compiler(s) are available in SYS$SYSTEM:

Filename                      Version
--------------------------------------
DECC$COMPILER.EXE             V6.4-005
DECC$COMPILER_T06_04-001.EXE  T6.4-001

Ambiguous version number, please be specify a full version number, ex: V6.4-005
Version number : V6.4-005

$ sho logical DECC$COMPILER
   "DECC$COMPILER" = "SYS$COMMON:[SYSEXE]DECC$COMPILER_V06_04-005.EXE" (LNM$PROCESS_TABLE)
$ sho logical DECC$msg_shr
   "DECC$MSG_SHR" = "SYS$COMMON:[SYSMSG]DECC$MSG_SHR_V06_04-005.EXE" (LNM$PROCESS_TABLE)
```

When this procedure is run in a process, subsequent cc commands invoke the selected compiler version (until the procedure is run again). The process-level logicals used do not affect other processes or users on the system.

6.2 Side effects and restrictions on multiple versions

When you install this kit, it provides the latest CLD file (to establish DCL tables), the latest decc$rtldef.tlb, and the latest documentation, even if you select the option of having the new compiler as an alternate compiler. The new CLD file and the new decc$rtldef.tlb do not adversely impact a preexisting V6-based compiler because they are upwardly compatible. HOWEVER, if you attempt to use a new compiler option with an older compiler, you will no longer get a DCL error that the option is not supported. Instead, the option will be reported by VCG. For example, note that before installation of V6.4, the following would have produced "%DCL-W-IVQUAL, unrecognized qualifier ...."

```
$@sys$system:decc$set_version.com 6.0
$cc /stand=c99 foo.c  ! /standard=c99 new in 6.2
$ %VCG-F-INVCMDVAL, "C99" is an invalid command qualifier value.
```

And because you must have the newest CLD file and header files in order to use the newest compiler, if you run an older installation procedure to put an older compiler back on your system, you must then re-run the 6.4 installation to get the newest files.

Note that there are two logical names involved in establishing the compiler version - one for the compiler image and one for its message file. This version of the compiler will issue a diagnostic if it is invoked with the wrong version of the message file - but prevous versions of the compiler do not detect this situation. If you find that an older version of the compiler is issuing diagnostics that don't make sense for the code construct they're attached to, or if the message text is missing and only a message number is issued, check that you have matched versions of the files designated by the two logicals using the command "$ show logical decc$compiler*". The response should show matching version-numbered files as in the example selecting the 6.2 compiler. Or if you are using the system compiler, the response should be "%SHOW-S-NOTRAN, no translation for logical name DECC$COMPILER*".

6.3 Installation Procedure Changes

When you install Compaq C V6.4 on a system that already has a 6.0 or higher compiler installed, you will be given the opportunity to preserve the currently-installed system compiler. To do this, answer yes to the following question (the xxx will be replaced by the full version number of the existing system compiler):

```
"Should the existing xxx system compiler be preserved [NO]:"
```

If you answer no, the installation will procede in the traditional manner, overwriting the currently-installed system compiler.

If you answer yes, you will be asked an additional question. To get the traditional behavior of installing the kit compiler as the system default, answer NO to the question:

```
"Should this xxx system compiler remain the
                        default when cc is typed [NO]:"
```

Since you have previoiusly asked to preserve the existing system compiler, that compiler is made an alternate compiler before installing the new system compiler from the kit. If you answer yes to the question, the kit compiler will be installed as an alternate compiler with a name based on its ident and the existing system default compiler will remain the default.

6.4 Sample installation fragment

```
  Compaq C V6.4 for OpenVMS VAX Systems
!
!  Copyright 2001 Compaq Computer Corporation
!
!  Compaq and the Compaq logo Registered in U.S. Patent and Trademark Office.
!
!  Confidential computer software. Valid license from Compaq required for
!  possession, use or copying. Consistent with FAR 12.211 and 12.212,
!  Commercial Computer Software, Computer Software Documentation, and
!  Technical Data for Commercial Items are licensed to the U.S. Government
!  under vendor's standard commercial license.
!

    A C V6.0-001 compiler was found on your system.
    Type YES to keep this compiler on your system either as
    the default system compiler, or as an alternate compiler.
    Type NO to supersede C V6.0-001.
```

```
* Should the existing V6.0-001 system
     compiler be preserved [NO]: yes

    Type NO to have the compiler on this kit become the default system compiler
    and to have the currently installed compiler saved as an alternate compiler.
    Type YES to keep the current system compiler as the default compiler,
    and to have the compiler on this kit available as an alternate compiler.
    Alternate compilers can be invoked with the cc command after invoking
    sys$system:decc$set_version.com passing a version_number.

* Should this V6.0-001 system compiler remain
     the default when cc is typed [NO]: yes

        Product:      C
        Producer:     DEC
        Version:      6.4
        Release Date: 05-JAN-2001

* Does this product have an authorization key registered and loaded? yes

....

    Your OpenVMS system will now be modified to include the
    following new and modified files:

    [SYSEXE]DECC$COMPILER_V06_04-005.EXE     [new]
    [SYSMSG]DECC$MSG_SHR_V06_04-005.EXE      [new]
    [SYSHLP]HELPLIB.HLP                   [modified]
    [SYSHLP.CC$VAX_HELP_064]*.*           [new]
    [SYSHLP]CC064.RELEASE_NOTES           [new]
    [SYSLIB]DCLTABLES.EXE                 [modified]
    [SYSLIB]DECC$RTLDEF.TLB               [new]
    [DECC$LIB.REFERENCE.SYS$STARLET_C]*.H [new]
    [DECC$LIB.REFERENCE.DECC$RTLDEF]*.H   [new]
    [SYSLIB]DECC$OLB.EXE                  [new]
    [SYSLIB]DECC$OLB.README               [new]
    [SYSLIB]DECC$OLB.OLB                  [new]
    [SYS$STARTUP]CCXX$STARTUP.COM         [modified]
    [SYSEXE]DECC$SET_VERSIONS.COM         [new]
    [SYSEXE]DECC$SHOW_VERSIONS.COM        [new]
    [SYSTEST]DECC$IVP.COM                 [modified]
```

---

7 Compaq C for OpenVMS VAX Release Notes

7.1 New Features in Compaq C V6.4

In addition to the support for installing multiple compiler versions, this version provides most of the language-feature support for the new C99 standard that will be implemented for VAX. The C99 standard (ISO/IEC 9899:1999) was published by ISO in December, 1999 and adopted as an ANSI standard in April, 2000. An official copy of the standard can be purchased and downloaded as a PDF file for less than $20US from either or

While Compaq C for OpenVMS Alpha will support all of C99, the VAX version is intended to provide only those features whose implementation is shared with Alpha. In particular, features that involve changes to the VAX code generator or to VAX-specific run-time libraries are not planned. Most notably, the new data types "_Complex" and "long long int" are not planned for VAX, nor are the new C99 "type-generic" math library functions, other new math functions, or the hexadecimal notation for floating point values. [Note that since the new C99 "long long int" data type is not implemented for VAX, there is no change to

the rules for determining the type of a decimal integer constant; only the Alpha implementation is affected by the new C99 rules for decimal integer constants.]

The following specific enhancements were made:

- Language modes, `/STANDARD=C99`, and message groups

  The compiler's default language mode remains /STANDARD=RELAXED_ANSI89, which accepts nearly all language extensions as well as standard C89 and C99 features. It excludes only K&R ("common" mode), VAX C, and Microsoft features that conflict with standard C. The /STANDARD=ANSI89 mode continues to implement strictly the 1990 ANSI/ISO C standard (commonly called C89), issuing all required diagnostics as well as a number of optional diagnostics that help detect source code constructs that are not portable under the C89 standard (digraphs from the 1994 Amendment are also recognized in this mode, even though they were not specified in the 1990 standard). The ISOC94 keyword can still be added to any of the modes (except VAXC) to predefine the macro __STDC_VERSION__, as specified in Amendment 1 to the C89 standard.

  The command line will recognize a new language mode qualifier, /STANDARD=C99, but it will issue a warning and treat it exactly the same as /STANDARD=RELAXED_ANSI89, which accepts all of the C99 features that are implemented for VAX.

  Since the standard is quite new, use of C99 features is not really portable in a practical sense yet. Also, the term "ANSI C" or "standard C" will be ambiguous for some time to come (i.e. do you mean C89 or C99). To help with this situation, the compiler has added three new message groups for messages that report the use of features in the following categories:

```
noc89  - features not in C89
noc99  - features not in C99
newc99 - features that are new in C99.
```

The existing group, `noansi`, which is now somewhat ambiguous in name, is retained as a synonym for noc89.

- `#pragma names`

  This pragma offers the same kinds of control over the mapping of external identifiers into object module symbols as does the command line qualifier /NAMES, and it uses the same keywords (except that the "lowercase" keyword is not supported). But as a pragma, the controls can be applied selectively to regions of declarations. The pragma has a save/restore stack that is also managed by `#pragma environment`, and so it is well-suited to use in header files. The effect of "`#pragma environment header_defaults`" is to set NAMES to "uppercase,truncated", which is the compiler default.

  One important use for this feature is to make it easier to use command line option `/NAMES=AS_IS`. Both the C99 standard and the C++ standard require that external names be treated as case-sensitive, and 3rd party libraries and Java native methods are starting to rely on case-sensitivity (C99 requires a minimum of 31 characters significant, while C++ requires all characters

significant). Therefore we expect the use of /NAMES=AS_IS to become much more widespread.

The Compaq C run-time library was implemented with all symbols duplicated and spelled both in uppercase and lowercase to allow C programs compiled with any of the /NAMES= settings to work. But traditional practice on OpenVMS combined with compiler defaults of /NAMES=UPPER has resulted in nearly all existing object libraries and shared images to contain all uppercase names (both in references and in definitions), even though C source code using these libraries typically declares the names in lowercase or mixed case. Usually, the header files to access these libraries contain macro definitions to replace lowercase names by uppercase names to allow client programs to be compiled /NAMES=AS_IS. But macro definitions are problematic because every external name has to have a macro.

The new pragma allows header files to specify just once that the external names they declare are to be uppercased in the object module, regardless of the NAMES setting used in the rest of the compilation. The NAMES setting in effect at the first declaration of an external name is the one that takes effect, thus the setting specified in a header file will not be overridden by a subsequent redeclaration in the user's program (which might specify a different NAMES setting). Note that the automatic Prologue/Epilogue header file inclusion feature described in section 1.7.4 of the User's Guide (in connection with pointer_size pragmas) can also be used to specify the NAMES setting for all headers in a given directory or text library, without having to edit each header directly.

```
Syntax:

#pragma names <stack-option>
#pragma names <case-option>[, <length-option>]
#pragma names <length-option>[, <case-option>]

Where <stack-option> is one of:

    save      - save the current names state
    restore   - restore a saved names state

<case-option> is one of:

    uppercase - uppercase external names
    as_is     - do not change case

and <length-option> is one of

    truncated - truncate at 31 characters
    shortened - shorten to 31 using CRC
```

- New "_m" suffix forces pragmas to expand macros

  As specified in the Language Reference Manual, there is a fixed "grandfathered" list of `#pragma` directives that always undergo macro expansion in the preprocessor before being translated. No other `#pragma` directives normally undergo macro expansion. But since there are sometimes circumstances where macro expansion is needed on a particular instance of a pragma, a general mechanism has been added such that spelling the name of a `#pragma` directive with a trailing "_m" suffix will cause that directive to undergo macro expansion. An example of use follows in the next bullet

item for #pragma assert non_zero. The suffix is permitted on all pragmas, including those that are already specified as undergoing macro expansion (in which case it has no effect).

- New non_zero keyword for #pragma assert

  This new keyword allows you to assert that a particular constant-expression must be non-zero at compile time, and supply a text string to be output if the assertion proves false. The text string is output with a warning message that includes the source text of the expression.

  ```
  Syntax:
     #pragma assert non_zero(<constant-expression>) <string-literal>
  ```

Note that the constant-expression is a C language constant-expression, not just a preprocessor #if expression. And while #pragma assert itself does not perform macro expansion, the alternate form #pragma assert_m can be used to cause macro expansion to take place, which is most often what is desired, as in the second example below (since "offsetof" is a macro).

```
Example:
   #pragma assert   non_zero(sizeof(a) == 12) "wrong size a"
   #pragma assert_m non_zero(offsetof(s,b) == 4) "wrong offset b"
```

If the sizeof a is not 12, and the offset of member b in the struct named by type b is not 4, the following diagnostics are output:

```
CC-W-ASSERTFAIL, The assertion "(sizeof(a) == 12)" was not true.
wrong size a.
CC-W-ASSERTFAIL, The assertion "(offsetof(s,b) == 4)" was not true.
wrong offset b.
```

- New use of static keyword in array bounds.

  C99 permits the keyword "static" to be used within the outermost array bound of a formal parameter in a prototype function declaration. The effect is to assert to the compiler that at each call to the function, the corresponding actual argument will provide access to at least as many array elements as are specified in the declared array bound. Consider the following two function definitions:

  ```
  void foo(int a[1000]){ ... }
  void bar(int b[static 1000]) { ... }
  ```

The declaration of foo is absolutely equivalent to one that declares "a" to be "int *". When compiling the body of foo, the compiler has no information about how many array elements may exist. The declaration of bar differs in that it asserts to the compiler that it may assume that at least 1000 array elements exist and may be safely accessed. The intent is to provide a hint to the optimizer about what can be safely pre-fetched.

- New type _Bool

  This is the C99-specified boolean type. It is not recognized in VAXC, COMMON, or the strict ANSI89 mode. An object of this type occupies one byte and is an unsigned integer, but its value can only be either 0 or 1. It is permitted to use _Bool as the type for a bit-field. When a value of any

scalar type (any arithmetic type including floating point and the new complex types, or any pointer type) is converted to _Bool, the result is zero if the value would compare equal to 0 (e.g. if the pointer is NULL), and otherwise the result is 1. It is intended to be used in conjuction with a new standard header, <stdbool.h>, but that is not required. The content of the new header is simply as follows:

```
#define bool _Bool
#define true 1
#define false 0
#define __bool_true_false_are_defined 1
```

- New type keywords `_Complex` and `_Imaginary`

  Although neither of these C99 types will be implemented on VAX, their use will be diagnosed. Since these names begin with an underscore followed by an uppercase letter, they are in the namespace reserved to the C implementation and should not be used even in strict ANSI89 mode.

7.2 New Features in Compaq C V6.2

This release primarily contains a number of new language features from the in-process revision to the C standard, C9X (expected to be C99), and from the gcc compiler (to aid compatibility with source code from Linux systems). It also has diagnostic message improvements, as well as bug fixes and miscellaneous improvements.

The following specific enhancements were made:

- more html-format documentation

  The Compaq C manuals, help, and release notes are now installed in html form in a subdirectory of SYS$HELP. They can be accessed from a locally-running browser through

  `file:/sys$common/syshlp/cc$vax_help_062/index.htm`

  To access these documents from a browser running on a different system, you can copy the entire directory contents to a directory or server system that is accessible to your browser.

- diagnostic for unbalanced pragma state save/restore

  The compiler now issues a diagnostic when a #pragma stack is not empty at the end of a compilation (i.e. when the program issues a `#pragma <pragma-name> save` directive without ever issuing a corresponding `#pragma <pragma-name> restore`

- additional diagnostics to help locate unmatched braces

  When a closing brace is omitted, the parser error is generally reported against a location far from the point of the actual coding error. Additional diagnostics now attempt to find which opening brace was not matched, based on heuristic observation of the indentation style used in the rest of the source code. In testing, these messages have proved quite accurate, but if the source code is very inconsistent in this style, or uses an unusual style, the messages may not be very effective. We would appreciate feedback on this feature, particularly testcases where the diagnostic misidentified which brace was unmatched.

- new keywords for /accept qualifier

  New keywords for the /accept command line qualifier:

  - [no]c99_keywords

    Controls whether or not the new keywords being introduced in the C standard that are in the C89 namespace for user identifiers (inline and restrict) are accepted without double leading underscores.

  - [no]gccinline

    The gcc compiler implements an inline function qualifier for functions with external linkage that gives similar capabilites to the C9X feature described below, but the details of usage are somewhat different (basically the combination of extern and inline keywords makes an inline definition, instead of the exlusive use of the inline keyword without the extern keyword). This option controls which variation of the feature is implemented.

- extern inline functions (C9X and gcc)

  A new keyword, `inline`, has been introduced which can be used as a declaration specifier in the declaration of a function. With static functions, this has the same effect as applying `#pragma inline` to the function. When the specifier is applied to a function with external linkage, besides suggesting to the compiler that calls within that translation unit be inlined, there are additional rules that allow calls to the function also to be inlined in other translation units or else called as an external function at the compiler's discretion:

  - If the inline keyword is used on a function declaration with external linkage, then the function must also be defined in the same translation unit.
  - If all of the file scope declarations of the function use the inline keyword but do not use the extern keyword, then the definition in that translation unit is called an inline definition, and no externally-callable definition is produced by that compilation unit. Otherwise, the compilation unit does produce an externally-callable definition.
  - An inline definition must not contain a definition of a modifiable object with static storage duration, and it must not refer to an identifier with internal linkage. These restrictions do not apply to the externally-callable definition.
  - As usual, at most one compilation unit in an entire program can supply an externally-callable definition of a given function.
  - Any call to a function with external linkage may be translated as a call to an external function, regardless of the presence of the inline qualifier. It follows from this and the previous point that any function with external linkage that is called must have exactly one externally-callable definition among all the compilation units of an entire program.
  - The address of an inline function with external linkage is always computed as the address of the unique externally-callable definition, never the address of an inline definition.

- A call to an inline function made through a pointer to the externally-callable definition may still be inlined, or translated as a call to an inline definition, if the compiler can determine the name of the function whose address was stored in the pointer.

- intermixed declarations and code, "for" loop declarations (C9X and gcc)

  The C++ language has always allowed these, and they are now being added to C. The rules are the same as for C++. Within a compound statement, statements and declarations may be freely interspersed. This allows declarations to be placed nearer to their point of first use without introducing additional nested compound statements.

  And in the `for` statement, the first clause may be a declaration whose scope includes the remaining clauses of the `for` header and the entire loop body. This is normally used to declare and initialize a local loop control variable, e.g.

  ```
  for (int i=0; i<10; i++)
      printf("%d\n", i);
  ```

- __func__ predeclared identifier (C9X and gcc)

  Anywhere within the body of a function definition, code can assume that there is visible an identifier named __func__ that is declared as a static array of char initialized with the spelling of the function's name. E.g a function defined as

  ```
  void foo(void) {printf("%s\n", __func__);}
  ```

  will print "foo".

- compound literals (C9X and gcc)

  A compound literal is a new form of expression that constructs the value of an object, including objects of array, struct, or union type. In C89, passing a struct value to a function typically involves declaring a named object of the type, initializing its members, and passing that object to the function. A compound literal is an unnamed object specified by syntax consisting of a parenthesized type name (i.e. the same syntax as a cast operator) followed by a brace-enclosed list of initializers. Note that the initializer list can use the recently-introduced designator syntax.

  E.g. to construct an array of 1000 ints that are all zero except for array element 5, which is to have a value of 7, you can write: (int [1000]){[5] = 7}.

  A compound literal object is an lvalue. The object it designates has static storage duration if it occurs outside of all function definitions, and otherwise has automatic storage duration associated with the nearest enclosing block.

- comment introducers optionally detected within comments, to help find unterminated comments

  The new message "nestedcomment" can be enabled to report occurrences of "/*" inside of a comment introduced by "/*". This often indicates that a terminating "*/" was omitted, although certain coding practices may also produce many occurrences that are harmless. The message is also enabled by enabling the "level4", "unused", or "questcode" groups.

- __align synonym for _align (ANSI namespace)

The long-supported _align storage class modifier is in the namespace reserved for users' identifiers in some contexts under the C standard, and so it could not be recognized as a keyword in strict ANSI mode (/standard=ansi89). An alternate spelling with two leading underscores (putting it in the namespace reserved to the C implementation) is now recognized in all modes so that the feature can be used when compiling in strict ANSI mode.

- __typeof__ unary operator (gcc)

  The gcc compiler provides an operator named "__typeof__" that can be used much like the standard C operator "sizeof", except that instead of producing a value (the size of an expression or type) it produces a type (the type of the expression or type that is its single operand). It can be convenient to use in macros for generating declarations or casts that use the same type as some expression or typename supplied as an argument to the macro.

- pointer arithmetic on void and function pointers (gcc)

  Pointer arithmetic on void* pointers or pointers to functions formerly produced hard errors. The gcc compiler allows this, treating both as if they were char* pointers. This behavior has been adopted, with an appropriate warning instead of an error.

- string initializers optionally enclosed in parentheses (gcc)

  In standard C, a string literal ordinarily has type char *. A special case is made when a string literal is the initializer for an array of char, in which case it is essentially treated as an array object that provides the values (and, in the case of an incomplete array, the size) of the array of char being initialized.

  Also in standard C, a string literal enclosed in parentheses is not itself a string literal, and so this special case would not apply - instead the parenthesized literal would be treated as a single pointer of type char *, which is not a valid initializer for an object of type array of char.

  The gcc compiler allows a parenthesized string literal to initialize a char array, and the construct commonly appears in Linux source code. So that behavior has been adopted, with an appropriate warning.

- difference of addresses in same object are constants (gcc)

  If the & operator is used to obtain the addresses of two lvalues within the same object, and the lvalues are specified with integral constant expressions, then the result depends only on the layout of the object, and for practical purposes can be computed at compile time much like the integral constant expressions that are required to be produced by sizeof and offsetof.

  E.g. for any array "a" of int, the value of &a[4] - &a[3] must be sizeof(int), and sizeof(int) is an integral constant expression. But the C standard's specification of expressions that must be treated by the compiler as integral constant expressions does not include use of the & operator, and so &a[4] - &a[3] is not required to be treated as such, and Compaq C has not previously done so. But the C standard also explicitly recognizes that implementations may treat additional forms of expressions as constant expressions, and gcc and other compilers do treat these cases as integral constant expressions. Now Compaq C does as well.

7.3 New Features in DEC C V6.0

- New command line qualifier /[NO]PROTOTYPES:

```
    /[NO]PROTOTYPES=(FILE=<filename>,
                    [NO]IDENTIFIERS,
                    [NO]STATIC_FUNCTIONS)

    default is /[NO]PROTOTYPES

Syntax Description:

    /[NO]PROTOTYPES
            Creates an output file containing function
            prototypes for all global functions defined in
            the module which is being compiled.

    keywords:

        IDENTIFIERS
            negatable optional parameter, which indicates
            that identififer names are to be included in
            the prototype declarations that appear in the
            output file.  The default is NOIDENTIFIERS

        STATIC_FUNCTIONS
            negatable optional parameter, which indicates
            that prototypes for static function
            definitions are to be included in the output
            file.  The default is NOSTATIC_FUNCTIONS

        FILE=<filename>
            an optional parameter specifying the output
            file name.  When not specified the output file
            name will be have the same defaults as listing
            file, except that the .CH file extension is
            used instead of the .LIS extension.
```

- New command line qualifier, /SHOW=MESSAGES. This qualifier adds a new section to the listing file showing all of the compiler's diagnostic messages that are enabled at the start of the compilation, after the command line has been processed. The listing shows the message identifer, the severity, and the parameterized text of each enabled message, reflecting the effects of the /standard and /warnings command line qualifiers (except that severity-based suppression, /warnings=noinformationals or /nowarnings, is not reflected). The /warnings=verbose qualifier causes this listing to be expanded with the "Description" and "User Action" text following the text for each enabled message.
- New informationals to report apparently unnecessary #include files and CDD records. The most useful of these, UNUSEDTOP, reports only headers explicitly #included at the top level in the compilation unit that did not provide anything used by the rest of the compilation. This message is enabled at level4. Other diagnostics to report on the effects of #includes nested within other headers, and on CDD records, are enabled at level5. All of these messages can be enabled by the message group UNUSED. Unlike any other messages, these messages must be enabled on the command line in order to be effective. The processing that analyzes the dependencies on included files is signficant, and it must be started before processing of the input files begins. Any #pragma message directives within the source have no effect on these messages, their state is determined only by processing the command line.

- The builtin function __ALLOCA( ), which has been available on Alpha since the earliest releases, has been implemented for VAX. Either #include <builtins.h> or use #pragma builtins to enable it.
- Variable length arrays from the C9X review draft have been implemented. This feature permits array objects with auto storage class, and array typedefs declared at block scope, to have bounds that are runtime-computed expressions. It also permits the declaration and definition of functions whose parameters are arrays dimensioned by other parameters (similar to Fortran assumed-shape arrays). The following example illustrates both uses. Note that the definition of function sub( ) uses prototype syntax and that the dimension parameters precede the array parameter that uses them. In order to define a function with the dimension parameters following the array parameter that uses them, it is necessary to write the function definition using K&R syntax (since that syntax allows the declarations of the types of the parameters to be written in a different order from the parameters themselves). K&R function definitions should generally be avoided.

```
#include <stdio.h>
#include <stdlib.h>

void sub(int, int, int[*][*]);

int main(int argc, char **argv)
{
    if (argc != 3) {
        printf("Specify two array bound arguments.\n");
        exit(EXIT_FAILURE);
    }
    {
        int dim1 = atoi(argv[1]);
        int dim2 = atoi(argv[2]);
        int a[dim1][dim2];
        int i, j, k = 0;
        for (i = 0; i < dim1; i++) {
            for (j = 0; j < dim2; j++) {
                a[i][j] = k++;
            }
        }
        printf("dim1 = %d, dim2 = %d.",
               sizeof(a)/sizeof(a[0]),
               sizeof(a[0])/sizeof(int));

        sub(dim1, dim2, a);
        sub(dim2, dim1, a);
    }
    exit(EXIT_SUCCESS);
}

void sub(int sub1, int sub2, int suba[sub1][sub2])
{
    int i, j, k = 0;
    printf("\nIn sub, sub1 = %d, sub2 = %d.",
                       sub1,      sub2);
    for (i = 0; i < sub1; i++) {
        printf("\n");
        for (j = 0; j < sub2; j++) {
            printf("%4d", suba[i][j]);
        }
    }
}
```

Finally, note that variable length arrays can often be used in place of the non-standard __ALLOCA( ) builtin function, an important difference being that the storage allocated by __ALLOCA is not normally freed until return from the function containing the call, while the storage allocated for a variable length array is freed on exit from the block in which it is allocated. If __ALLOCA is called within the scope of a declaration for a variable length array object, an error is issued (MIXALLOCAVLAV). On the Alpha platform, such usage produces a warning and causes the storage allocated by __ALLOCA to be freed when the block containing the variable length array declaration is exited.

7.4 New Features in DEC C V5.7

This release contains a number of new features aimed primarily at ease of use and programmer productivity.

- Enhanced diagnostic message controls. The `/WARNINGS` command line qualifier and its matching `#pragma message` have had a number of new features added in an upwardly-compatible way. Refer to the online help for `/WARNINGS` for specific usage information. Features include:
  - Specify whether a message is issued only once per compilation, or at each occurrence.
  - Reduce the severity of any message that has a default severity of informational or warning, or increase the severity of any message. Reducing a warning to an informational can allow generation of a "warning-free object module", without suppressing the diagnostic altogether. Increasing the severity of an informational or warning to an error can help enforce programming practices by causing specific diagnostics to "break" a build.
  - Control optional messages using a single numeric "importance level". The "check" group of messages basically allowed enabling a large number of additional messages, some useful, some not very useful in many cases. Messages have now been grouped into 5 importance levels, named level1-level6. The default is level3. The "check" group is now treated as a synonym for level5. The "all" group is treated as a synonym for level6. The level1 and level2 groups correspond to "quiet" and slightly more "noisy" versions of Digital UNIX compilers, respectively. Enabling a level enables optional messages at that level and all lower levels. Disabling a level disables optional messages at that level and all higher levels.
  - Control optional messages using functional groups. The previous functional groups (c_to_cxx, check, portable, all) have been retained, and a number of new groups have been added. Many of the names for the new functional groups correspond to groups recognized by the "lint" utility on DIGITAL UNIX: ALIGNMENT, DEFUNCT, NOANSI, OBSOLESCENT, OVERFLOW, PERFORMANCE, PREPROCESSOR, QUESTCODE, RETURNCHECKS, UNINIT, UNUSED, CDD.
  - /WARNINGS=VERBOSE adds explanatory help text following each diagnostic message output.

  Besides the new features, the entire set of compiler messages was reviewed and updated. As a result, the exact set of messages reported by a default compilation is somehwat different. Overall, the default level3 setting is slightly quieter, particularly because the default mode of relaxed_ansi does not report uses of language extensions. Also, the severity of many warning

messages has been reduced to informational. Finally, the Messages subtopic for CC now contains useful additional information about each message.

- New command line qualifier /SHOW=SYMBOLS. This will add a symbol table map to the listing (if a listing is requested). This is similar, but not identical, to the output from the VAX C compiler.
- New command line qualifier /SHOW=BRIEF. This qualifier is similar to the new qualifier /SHOW=SYMBOLS, except that unreferenced symbols declared in header files are omitted.
- New command line qualifier /CROSS_REFERENCE, or equivalently /SHOW=CROSS_REFERENCE. This qualifier adds a list of line numbers at which each listed symbol is referenced (if a listing is requested). If the /SHOW qualifier is omitted, this qualifier causes the /SHOW=BRIEF symbols to be listed. When appropriate, the line number designating a reference to a symbol is annotated with a suffix indicating the way in which the symbol was used on that line, as follows:
  - = Assigned or initialized.
  - & Address taken.
  - () Function called.
  - * Simple dereference.
  - -> Member dereference.
  - . Member selection (no indirection).
  - [] Subscripted (i.e. using [] syntax).
  - b Invoked as a builtin function.
- New command line qualifier /ACCEPT=[NO]feature. This qualifier tells the compiler to accept (or reject) particular language features, regardless of the setting of the /STANDARD qualifier. There are two features that can be controlled in this way:
  - VAXC_KEYWORDS. Specifying this feature tells the compiler to recognize and process the following identifiers as keywords: _align, globaldef, globalref, globalvalue, noshare, readonly, variant_struct, variant_union. Specifying NOVAXC_KEYWORDS tells the compiler to treat these as ordinary identifiers. The default is to recognize these as keywords in all language modes other than strict ANSI and common modes.
  - RESTRICT_KEYWORD. Specifying this feature tells the compiler to recognize and process the C9X keyword restrict as a type qualifier keyword. By default, in current language modes only the reserved-namespace spelling __restrict is treated as a keyword.
- New command line qualifier /NAMES=SHORTENED. External symbol names longer than 31 characters are, by default, truncated to 31 characters by the compiler in order to conform to the linker limit, as they always have been. This new option instructs the compiler to shorten the name without losing all information about the characters that were removed. The shortened name contains a CRC encoding of the characters removed, similar to way that the C++ compiler treats its mangled names that very often exceeed 31 characters. This allows programs containing long external names that are not unique within the first 31 characters to be linked successfully. Naturally, if a program contains external names longer than 31 characters, all of its

modules must be compiled with the same setting of this qualifier in order to link successfully. The default is /NAMES=TRUNCATED.

- New command line qualifier /REPOSITORY=dirspec. This qualifier is only useful in conjunction with /NAMES=SHORTENED, and when the default directory specification of [.CXX_REPOSITORY] is not acceptable. When the compiler shortens a name under the /NAMES=SHORTENED option, it also writes a mapping from the shortened name to the original full-length name in the repository. The CXXDEMANGLE utility, which now also ships with the C compiler, can be used to find the original name corresponding to a shortened name. That utility also assumes that the shortened name repository is located in [.CXX_REPOSITORY] unless a different directory is explicitly specified. See the help for CXXDEMANGLE. An option to perform compatible shortening on long names with extern "C" linkage is planned for a future release of C++. Note that a shortened C name is formed using a convention that will never match a C++ "mangled" name, so a single repository can be shared by C and C++ compilations.

7.5 New Features in DEC C V5.6

This is primarily a maintenance release focused on bug fixes, usability and message improvements, and providing V7.1 runtime library features on prior versions of VMS. Please see the file SYS$LIBRARY:DECC$CRTL.README for specific information on building applications that need to use new DEC C runtime library features on older versions of OpenVMS.

- Message group C_TO_CXX. This message group contains an optional set of diagnostics that report the use of a number of C language constructs that are not compatible with, or have a slightly different meaning in, the C++ language. This group may be enabled explicitly either on the command line (/WARN=ENABLE=C_TO_CXX) or by #pragma message enable (c_to_cxx).

- New diagnostics to detect simple expressions with side effects that are undefined in ANSI C. The C standard formalized defacto rules about side effects in terms of sequence points. An expression that modifies the same object more than once, or that modifies an object and fetches its value for a purpose other than computing the modified value, has undefined behavior unless there is an intervening sequence point. The compiler now warns about such expressions (only for objects that are simple declared variables).

- Source listings now include statement level nesting. The annotation at the left margin of the source listing now includes the statement nesting level in effect at the end of that source line. The statement nesting level appears as a simple integer before the listing line number. The block of a function definition is level 1. Outside of function definitions, this field is blank.

7.6 New Features in DEC C V5.5

This is primarily a maintenance release focused on bug fixes, with very limited new functionality.

- New type qualifier "__restrict"

The ongoing work to revise the ANSI C language standard will likely incorporate a new type qualifier keyword "restrict" (the existing ANSI type qualifiers are "const" and "volatile"). This feature has been present in the Cray C compiler for some time and is also being adopted by other vendors. The type qualifier applies only to pointer types, and its basic purpose is to assert to the compiler that memory accesses made through a pointer declared with this type do not overlap with other memory accesses within the scope of that pointer, permitting additional optimization. The qualifier (with double leading underscores to avoid violating the ANSI89 namespace) is recognized and its correct compile-time usage is verified, but it does not trigger additional optimizations.

- Initial macros shown in listing file

  At the end of the listing file there is now a section containing a list of all macros in effect at the start of the compilation, along with their values. This includes both those predefined by the compiler (except for ANSI-mandated macros that cannot be undefined or redfined) and the result of applying all /DEFINE and /UNDEFINE qualifiers.

7.7 New Features in DEC C V5.3

- New qualifier keyword value /STANDARD=MS.

  This qualifier enables language compatibility features to accept some of the language extensions present in 32-bit Microsoft C compilers, and causes the predefined macros "__MS" and "__DECC_MODE_MS" to become defined with a value of 1. It does not provide complete compatibility with a particular version of Microsoft's compiler, but a limited selection of relatively minor extensions that can ease porting of C code developed under Microsoft C. Examples include unnamed struct and unions (same syntax as unnamed unions in C++, similar function to variant struct and union in VAX C), and relaxation of pointer and integer comparisons. It does not include such major extensions as structured exception handling or thread local storage.

- New qualifier keyword value /ASSUME=[NO]HEADER_TYPE_DEFAULT

  The negated form of this value disables the compiler's supplying of a default filetype extension of ".H" for source files included by the #include preprocessing directive. This feature is primarily for compatibility with the C++ compiler, where the rules for ANSI C++ header file specifications conflict with the notion of having the compiler supply a filetype. The default is /ASSUME=HEADER_TYPE_DEFAULT, which enables the compiler to supply the filetype ".H" for included files as it always has. Also see Section 7.8 for more information on changes to #include processing.

- Attribute controls for psects using #pragma extern_model

  The extern_model pragma has been enhanced to allow explicit control over most psect attributes, not just shr/noshr. The syntax is:

```
#pragma extern_model model_spec [attr[,attr]...]

where model_spec is one of:
     common_block
     relaxed_refdef
     strict_refdef "name"
     strict_refdef /* No attr specifications allowed. */
     globalvalue   /* No attr specifications allowed. */

attr is chosen from (at most one from each line):
     gbl lcl       /* Not allowed with relaxed_refdef */
     shr noshr
     wrt nowrt
     pic nopic
     ovr con
     2 long 3 quad 4 octa 9 page
```

A description of these attributes may be found in table 4-4 of the DEC C User's Guide for OpenVMS Systems, and more complete information on each may be found in the OpenVMS Linker Utility Manual. The default attributes are: noshr, pic. For strict_refdef the default is con, and for common_block and relaxed_refdef the default is ovr. The default for wrt/nowrt is determined by the first variable placed in the psect. If the variable has the const type qualifier (or the readonly modifier) the psect will be set to nowrt, otherwise it is set to wrt.

NOTE: These attributes are normally used by system programmers who need to perform declarations normally done in macro. Most of these attributes are not needed in normal C programs. Also note that the setting of attributes is supported only through the `#pragma` mechanism, and not through the `/EXTERN_MODEL` command line qualifier.

- New compiler-generated psect $ADDRESS_DATA

  By default, previous versions of the compiler have placed all static-extent const data in the psect named $CODE. When the const data involved link-time addresses, this caused the entire $CODE psect to become non-shared. In V5.3, static-extent const data initialized with link-time addresses is placed in a new psect named $ADDRESS_DATA, leaving $CODE sharable. For example, given the declarations:

  ```
  static const int a = 5;
  static const int * const b = &a;
  ```

  variable a will be placed in the $CODE psect because it is initialized to a true compile-time constant value, while variable b will be placed in $ADDRESS_DATA because it is initialized to the address of a, which may differ among different image activations.

7.8 Changes to #include processing in DEC C V5.3

To support processing of ''prologue'' and ''epilogue'' file inclusion, the V5.2 compiler introduced substantial changes to the processing of the `#include` directive that allowed for increased code commonality between the OpenVMS Alpha and VAX versions of the compiler. In V5.3, further changes have been made to make the actual `#include` searching behavior identical for the OpenVMS VAX and Alpha compilers, and to support new ANSI C++ requirements on header file naming conventions. The following are some of the highlights of these modifications. For a complete description of #include processing, see the discussion of file inclusion in the `cc` online help for the `/include_directory` qualifier (`$ help cc/decc/include`).

- New qualifer option, `/assume=[no]header_type_default`

This option can disable the default file type mechanism for header files. Following VAX C, the DEC C compiler has traditionally supplied a default file type of ".H" for filenames specified without any file type extension in a `#include` directive using ANSI C syntax. Similarly, the DEC C++ compiler has supplied a default file type of ".HXX". However, the emerging ANSI standard for C++ now requires that, for example, `#include <iostream>` refer to a different file than `#include <iostream.hxx>`. V5.3 of both DEC C and DEC C++ support this capability through the /assume=noheader_ type_default qualifier option. Under this option, both DEC C and DEC C++ supply a default file type of just "." for files named in standard-syntax #include directives. Thus, for example, if a header file directory contains two files named "STDIO." and "STDIO.H", the directive `#include <stdio>` will cause DEC C to select "STDIO.H" by default. But under /assume=noheader_ type_default, "STDIO." will be selected. Besides matching the ANSI C++ requirement, this behavior is also more compatible with most other C compilers including UNIX and Windows/NT.

- Meaning of empty string in /INCLUDE_DIRECTORY

  The UNIX convention of using -I without a pathname to disable searching of the "standard places" is now fully supported by the /INCLUDE_DIRECTORY qualifier. If an empty string occurs as any element in the list of specifications supplied by this qualifier, the compiler does not search any of its default directories, logical names, or text libraries and uses only the specifications from the command line to find include files.

- Kit changes for text library packaging of system headers

  With this release, the DEC C kit for VAX has been changed to match the OpenVMS Alpha kit with respect to header file installation and searching. Previous releases have depended on the definition of a system logical name DECC$LIBRARY_INCLUDE to find plain text copies of all system header files. In V5.3, DEC C directly searches text libraries for the default runtime system header files just as the Alpha version of the compiler has always done. Previously, the VAX compiler was limited to searching a maximum of two default text libraries (DECC$TEXT_LIBRARY and DECC$RTLDEF.TLB) in addition to any specified by the user on the command line. This limitation has been removed, and as on Alpha the compiler may search several additional default text libraries (see the `cc/decc` online help for details).

  As on Alpha, the V5.3 DEC C installation procedure for VAX will create header file "REFERENCE" directories containing plain text copies of the header files packaged in the text libraries. This is as a user convenience for viewing and using the SEARCH command, and the compiler will not search these directories by default. Therefore, the system startup procedure for the compiler no longer defines the system logical name DECC$LIBRARY_ INCLUDE, and the installation kit deassigns that logical name if it is already defined as a system logical during the installation.

  To maintain functional compatibility with previous releases, however, the V5.3 compiler will still check for a definition of the logical name DECC$LIBRARY_ INCLUDE, and if it is defined then the compiler will use it to search for plain text copies of header files, and it will avoid searching the default text libraries for headers included with the quote or angle-bracket `#include` syntax. But by default, the logical name is undefined and the compiler will search the default text libraries to find such headers.

The V5.3 kit installation normally extracts reference copies of the headers as mentioned above. The modules from each separate text library file are extracted to a separate directory as shown in the following list - the last component of each directory name matches the file name of the .TLB library file whose modules it contains, and the default filetype for files in each directory is as shown (unless /assume=noheader_type_default is in effect):

```
SYS$COMMON:[DECC$LIB.REFERENCE.DECC$RTLDEF].H
SYS$COMMON:[DECC$LIB.REFERENCE.SYS$STARLET_C].H

NOTE:  OpenVMS VAX through version V7.0 does not
       supply a SYS$STARLET_C.TLB as OpenVMS Alpha
       has always done.  The starlet headers are
       constructed by the compiler kit and placed
       in the SYS$STARLET_C directory as well as
       in DECC$RTLDEF.TLB and its directory.  A
       future version of OpenVMS VAX is expected
       to ship SYS$STARLET_C.TLB as on Alpha.
```

Although the Alpha compiler normally finds system headers in the default text libraries, in the absence of a definition for the logical names DECC$LIBRARY_INCLUDE or DECC$SYSTEM_INCLUDE it first searches directories with names the same as the reference directories shown above, but with "REFERENCE" changed to "INCLUDE". Normally, no such directories exist on Alpha and so the modules are found in the default text libraries - but if they do exist the files in them will supercede modules of the same name in the default text libraries. Previous versions of the VAX compiler have behaved in a similar way except that they expected a single "flat" structure for include files:

```
SYS$COMMON:[DECC$LIB.INCLUDE].H
```

In previous versions of the VAX compiler, these directories were required to be fully populated since the compiler was unable to search all of the default text libraries anticipated. The V5.3 compiler can search all of the default text libraries, so these directories do not need to exist. Therefore, the V5.3 kit installation looks for a preexisting SYS$COMMON:[DECC$LIB]INCLUDE.DIR, and renames it to OLD_INCLUDE.DIR. It does not create and populate new [DECC$LIB.INCLUDE.*] directories. But to match the Alpha compiler's behavior, the V5.3 VAX compiler does search these directories if they exist and neither DECC$LIBRARY_INCLUDE nor DECC$SYSTEM_INCLUDE are defined as logical names, so any files in them will supercede modules of the same name in the default text libraries just as on Alpha.

- Restriction on total number of text libraries searched

  There remains a VAX-only restriction that the total number of text libraries searched, including user-specified libraries using the /LIBRARY qualifier, cannot exceed 16. If the limit of 16 total text libraries is exceeded in a compilation, the following warning message occurs:

  ```
  %CC-W-TOOMANYTXTLIB, Too many text libraries.
        Library 17th-library-name and subsequent will not
        be searched.
  ```

  If no errors are caused by missing include files, you can disable this warning with the /warn=disable=TOOMANYTXTLIB qualifier.

If errors are caused by missing system header files and *17th-library-name* is DECC$TEXT_LIBRARY or one of the compiler default text libraries from SYS$LIBRARY, then the solution is to reduce the number of user-specified command-line libraries. An alternative is for the compiler to get system headers from ordinary files in directories. This requires that the `#include` directives for system headers in user source code have the standard `#include` syntax rather than the Digital-specific module-name syntax. If reference copies of headers were extracted at installation time, you can specify the installed `[.REFERENCE.*]` subdirectories in a search list logical used in a `/include` qualifier or in the `DECC$SYSTEM_INCLUDE` logical as appropriate.

If several users on a system encounter the problem of missing system header files because they use too many text libraries, then the system manager may want to install the reference copies of system headers and then rename (or copy) the `[.REFERENCE.*]` subdirectories to `[.INCLUDE.*]`. The compiler searches the `[.INCLUDE.*]` directories by default, so it would not then be necessary for individual users to modify build procedures to look for plain text copies. It would still, however, be necessary to suppress the `TOOMANYTXTLIB` warning message.

- Interaction with older versions of DEC C++

  The DEC C and DEC C++ compilers both ship copies of the DEC C RTL header files and arrange for starlet header files to be constructed and made available during installation. The two compilers have traditionally accessed these headers from the same locations, so that upgrading one of the compilers would upgrade the CRTL and starlet headers accessed by the other compiler even if the other compiler was not upgraded. The V5.3 compilers continue in this fashion, but the interaction with the header files is more complex given the change in the way the new compilers find them. It is generally recommended that if you have both DEC C and DEC C++ installed, that you upgrade them to the same version at the same time, and especially so for upgrading to V5.3.

  Nevertheless, the V5.3 DEC C VAX kit attempts to upgrade the headers without damaging the operation of an earlier installed version of DEC C++. The kit checks for the existence and version of SYS$SYSTEM:CXX$COMPILER.EXE. If it exists and is earlier than V5.3, the kit generates a common compiler startup procedure to define a system logical CXX$LIBRARY_INCLUDE as follows:

  - CXX$LIBRARY_INCLUDE defined using DECC$LIBRARY_INCLUDE

    If the logical name CXX$LIBRARY_INCLUDE is currently defined as a system logical, and it contains the logical name DECC$LIBRARY_INCLUDE as an element in its translation, then the new definition will match the old except that the dependence on DECC$LIBRARY_INCLUDE is removed, and that element of the translation is replaced by the following three elements:

    ```
    SYS$COMMON:[DECC$LIB.REFERENCE.DECC$RTLDEF],
    SYS$COMMON:[DECC$LIB.REFERENCE.SYS$STARLET_C],
    SYS$LIBRARY
    ```

  - CXX$LIBRARY_INCLUDE defined without DECC$LIBRARY_INCLUDE

If the logical name CXX$LIBRARY_INCLUDE is defined as a system logical, but its translation does not contain DECC$LIBRARY_INCLUDE as an element, then the startup procedure will define it as it is currently defined.

- CXX$LIBRARY_INCLUDE not defined

  If the logical name CXX$LIBRARY_INCLUDE is not currently defined as a system logical, then the startup procedure will not define it.

The first case is the usual case, and it will cause the older DEC C++ compiler to search the new REFERENCE copies of the C headers provided by the C installation. While the V5.3 compilers themselves never implicitly search the REFERENCE copies of header files, a V5.3 installation may cause an older compiler to search the REFERENCE copies. So if you are upgrading DEC C to V5.3 and are not planning to upgrade DEC C++ right away, then you should accept the default of installing reference copies - otherwise the C++ compiler will be unable to find the updated headers and you will need to define CXX$LIBRARY_INCLUDE (manually) to access the old header files. The installation displays the current definitions for all DECC$ and CXX$ logical names before making any changes. Note that the old headers may have been renamed to [.OLD_INCLUDE], depending on whether or not they were previously installed in the default location. The V5.3 kits do not directly support installation of the REFERENCE headers in other than the default location.

Note that when the DEC C++ compiler is upgraded to V5.3, it will deassign the system logical CXX$LIBRARY_INCLUDE and generate a startup procedure without a definition for it. With both C and C++ compilers at V5.3, the REFERENCE copies are for human convenience only.

Under no circumstances does the V5.3 C kit define the logical name DECC$LIBRARY_INCLUDE. However, the V5.3 DEC C++ kit may define this logical name if its installation kit detects a version of DEC C older than V5.3. Thus, the compilers may be upgraded in either order. But upgrading only one of the compilers will leave the older compiler searching for system header files in the new REFERENCE directories which are intended only for programmer convenience from V5.3 onward.

7.9 New Features in DEC C V5.2

- The command line switch /STANDARD=ISOC94 has been added

  This qualifier enables digraph processing and pre-defines the macro __STDC_VERSION__ to the value 199409L. The macro __STDC_VERSION__ is used in certain header files to enable the declaration of additional wide character functions as specified in Amendment 1 to ISO/ANSI C. (Amendment 1, which specifies digraphs, alternate spellings for certain operators, and a number of new wide character functions, was officially adopted by ISO in November of 1994.)

  The meaning of /STANDARD with no keywords has been changed from /STANDARD=ANSI89 to /STANDARD=(ANSI89, ISOC94), i.e. strict conformance to the full new standard. The default mode of the compiler (/NOSTANDARD) is still RELAXED_ANSI89. The meaning of /STANDARD=ISOC94 is /STANDARD=(RELAXED_ANSI, ISOC94), i.e. requesting just the additional features from the amendment builds on the

default mode. The ISOC94 keyword may be used with any of the other keywords to the /STANDARD qualifier except VAXC.

- New builtins for DEC C on OpenVMS/VAX

  Four new builtins (_HALT, _MFTR, _MTPR, _READ_GPR) are now supported by DEC C for OpenVMS VAX for use by privileged applications. These builtins were adapted from the VAX C builtins by the same name, and generally work the same as the VAX C builtins. As with the other builtins on DEC C for VAX, they are used in conjuction with the header file builtins.h, which describes their interfaces. _READ_GPR is unprivileged. The other three will all generate a reserved opcode fault if executed in a mode other than Kernel.

- #pragma extern_prefix {SAVE | RESTORE | __save | __restore | "prefix_ to_use"}

  This feature is for use by RTL header file developers only to prefix function entry points with "prefix_to_use". Please note that the generated symbol name is all uppercase. The functionality should match that of the extern_ prefix pragma in DEC C++. Like the other pragmas with save/restore options, the state of this pragma is also controlled by #pragma environment.

  Note there is one known limitation with this feature: If a function is implicitly declared (ie, no prototype) then it does not get prefixed.

- Implicit processing of prologue/epilogue files before and after each #include.

  When the compiler encounters a #include preprocessing directive, it first determines the location of the file or text library module to be included. It then checks to see if one or both of the two following specially named files or modules exist in the same location: __DECC_INCLUDE_PROLOGUE.H, __DECC_INCLUDE_EPILOGUE.H (in the case of a text library, the .H is stripped off). If they do, then the content of each is read into memory. The text of the prologue file (if it exists) is processed JUST BEFORE the text of the file specified by the #include, and the text of the epilogue file (if it exists) is processed JUST AFTER the text of the file specified by the #include. Subsequent #includes that refer to files from the same location use the saved text from any prologue/epilogue file found there. The prologue/epilogue files are otherwise treated as if they had been #included explicitly, and #line directives are generated for them if /prepoc output is produced, and they appear as dependencies if /mms_dependency output is produced.

  The "location" is the VMS directory containing the #included file or the text library file containing the #included module. For this purpose, "directory" means the result of using the $PARSE/$SEARCH system services with concealed device name logicals translated. So if a #included file is found through a concealed device logical that hides a search list, the check for prologue/epilogue files is still specific to the individual directories making up the search list.

  This feature is primarily for compatibility with the DEC C and DEC C++ compilers for Alpha, where the addition of features to select 32-bit or 64-bit pointers has created a demand for a mechanism to address the short-term problem of automatically establishing a system-default compilation environment for the processing of system header files without having to edit all such files (which come from a number of different sources). The introduction of #pragma environment in V5.0 simplified the process of establishing a known compilation environment for an individual header file,

which would not be changed by (and would not change) the environment of other header files or the main source file, without having to list explicitly each pragma implemented in the particular compiler version of interest. But that mechanism requires editing each header file needing to have that kind of stable compilation environment. The implicit processing of prologue/epilogue files is intended to support a prologue file containing "#pragma environment save" followed by one or more pragmas to establish the desired environment (typically either "#pragma environment header_defaults" or "#pragma environment command_line"), and an epilogue file containing "#pragma environment restore". By copying these two specially-named files or modules to a directory or text library, all of the header files or modules in that directory or library can be made to have the same compilation environment independent of command line switches and enclosing files or modules.

7.10 Changes in DEC C RTL Header Files for V5.2 of DEC C/C++

The release notes in this section describe changes to the header files shipped with DEC C V5.2 for OpenVMS Systems. These header files contain enhancements and changes made to the DEC C Run-Time Library for OpenVMS Systems.

New function prototypes and structure definitions which define new functionality in the DEC C Run-Time Library correspond to new functionality added to the DEC C Run-Time Library which is shipped with OpenVMS V7.0.

- New Header Files Added

  A total of 20 new header files were added to the DEC C RTL suite of header files. Header files were added for implementation of Amendment 1 of the ISO C standard, compatibility with UNIX systems, and for introduction of new functions. Table 1 lists those headers added for DEC C V5.2.

Table 1 New DEC C V5.2 Header Files

Header File	Description
<dirent.h>	Directory Manipulation Functions
<ftw.h>	File Tree Walking
<if.h>	Socket Packet Transport Mechanism
<if_arp.h>	Socket Address Resolution Protocol
<ioctl.h>	I/O Controls for Special Files
<iso646.h>	Alternative Spelling for Language Tokens
<libgen.h>	Filename Manipulation
<memory.h>	String Handling
<mman.h>	Mapping Pages of Memory
<nameser.h>	Maximum Domain Name Size
<pwd.h>	Password File Access Functions
<resolv.h>	Resolver Configuration File
<resource.h>	Declarations for Resource Operations

(continued on next page)

Table 1 (Cont.) New DEC C V5.2 Header Files

Header File	Description
<strings.h>	String Handling
<timers.h>	Clock and Timer Functions
<times.h>	File Access and Modifications Times Structure
<tzfile.h>	Time Zone Information
<utsname.h>	User Information
<wait.h>	Declarations for Process Waiting
<wctype.h>	Wide Character Classification and Mapping

- New Functions Defined In Header Files

  OpenVMS V7.0 introduces many new DEC C RTL functions which have been added to fulfill the request of application developers and to implement those functions defined by ISO C Amendment 1. These functions have been implemented on both OpenVMS Alpha V7.0 and OpenVMS VAX V7.0. These functions are documented in the *DEC C Run-time Library Reference Manual for OpenVMS Systems*.

```
basename()       herror()        seed48()        sysconf()
bcmp()           hostalias()     seekdir()       telldir()
bcopy()          hstrerror()     setenv()        tempnam()
btowc()          index()         sethostent()    towctrans()
bzero()          initstate()     setitimer()     truncate()
closedir()       ioctl()         setnetent()     tzset()
confstr()        jrand48()       setprotoent()   ualarm()
dirname()        lcong48()       setservent()    uname()
drand48()        lrand48()       setstate()      unlink()
endhostent()     mbrlen()        sigaction()     unsetenv()
endnetent()      mbrtowc()       sigaddset()     usleep()
endprotoent()    mbsinit()       sigdelset()     vfwprintf()
endservent()     mbsrtowcs()     sigemptyset()   vswprintf()
erand48()        memccpy()       sigfillset()    vwprintf()
ffs()            mkstemp()       sigismember()   wait3()
fpathconf()      mmap()          siglongjmp()    wait4()
ftruncate()      mprotect()      sigmask()       waitpid()
ftw()            mrand48()       sigpending()    wcrtomb()
fwide()          msync()         sigprocmask()   wcsrtombs()
fwprintf()       munmap()        sigsetjmp()     wcsstr()
fwscanf()        nrand48()       sigsuspend()    wctob()
getclock()       opendir()       socket_fd()     wctrans()
getdtablesize()  pathconf()      srand48()       wmemchr()
gethostent()     pclose()        srandom()       wmemcmp()
getitimer()      popen()         strcasecmp()    wmemcpy()
getlogin()       putenv()        strdup()        wmemmove()
getpagesize()    random()        strncasecmp()   wmemset()
getpwnam()       readdir()       strsep()        wprintf()
getpwuid()       rewinddir()     swab()          wscanf()
getservent()     rindex()        swprintf()
gettimeofday()   rmdir()         swscanf()
```

  The following functions are specific to OpenVMS Alpha V7.0. These functions are the implementations specific to 64-bit pointers. *(Alpha only.)*

```
_basename64()   _mbsrtowcs64()  _strpbrk64()    _wcsncat64()
_bsearch64()    _memccpy64()    _strptime64()   _wcsncpy64()
_calloc64()     _memchr64()     _strrchr64()    _wcspbrk64()
_catgets64()    _memcpy64()     _strsep64()     _wcsrchr64()
_ctermid64()    _memmove64()    _strstr64()     _wcsrtombs64()
_cuserid64()    _memset64()     _strtod64()     _wcsstr64()
_dirname64()    _mktemp64()     _strtok64()     _wcstok64()
_fgetname64()   _mmap64()       _strtol64()     _wcstol64()
_fgets64()      _qsort64()      _strtoll64()    _wcstoul64()
_fgetws64()     _realloc64()    _strtoq64()     _wcswcs64()
_gcvt64()       _rindex64()     _strtoul64()    _wmemchr64()
_getcwd64()     _strcat64()     _strtoull64()   _wmemcpy64()
_getname64()    _strchr64()     _strtouq64()    _wmemmove64()
_gets64()       _strcpy64()     _tmpnam64()     _wmemset64()
_index64()      _strdup64()     _wcscat64()
_longname64()   _strncat64()    _wcschr64()
_malloc64()     _strncpy64()    _wcscpy64()
```

While each of these functions are defined in the DEC C V5.2 header files, those definitions are protected by using *if __VMS_VER >= 70000000* conditional compilation.

- Usage of Feature-Test Macros

  The header files shipped with DEC C V5.2 have been enhanced to support feature test macros for selecting standards for APIs, multiple version support and for compatibility with old versions of DEC C or OpenVMS. Please see the DEC C Run-time Library Reference Manual, section 1.5 "Feature-Test Macros for Header-File Control", for a complete description of the feature test macros that are available.

- Different Default Behavior After OpenVMS V7.0

  The functions *wait( )*, *kill( )*, *exit( )*, *geteuid( )*, and *getuid( )* have new default behavior for programs which are recompiled under OpenVMS V7.0 or later. To the retain the old behavior, use the _VMS_V6_SOURCE feature-test macro, as described in the reference manual.

- Upgrade to Support 4.4BSD Sockets

  As of OpenVMS V7.0, the socket definitions in the socket family of header files has added support for 4.4BSD sockets. To instruct the header files to use this support, define either *_SOCKADDR_LEN* or *_XOPEN_SOURCE_ EXTENDED* during the compilation.

  The functions *gethostbyaddr( )*, *gethostbyname( )*, *recvmsg( )*, *sendmsg( )*, *accept( )*, *bind( )*, *connect( )*, *getpeername( )*, *getsockname( )*, *recvfrom( )*, and *sendto( )* have a second implementation which uses a new *sockaddr* structure defined in <socket.h>.

- Integration of Timezone Support

  The DEC C RTL on OpenVMS V7.0 has added full support for Universal Coordinated Time using a public domain timezone package. When compiling on OpenVMS V7.0, the functions *gmtime( )* and *localtime( )* have a second implementation which use extensions to the *tm* structure defined in <time.h>. To retain the ANSI C definition of this structure, define either *_ANSI_C_SOURCE* or *_DECC_V4_SOURCE*. Note that compiling with the */standard=ansi* qualifier implies *_ANSI_C_SOURCE*.

- Integration of ISO C Amendment 1 Behavior

The DEC C RTL on OpenVMS V7.0 has added full support for Amendment 1 of the ISO C Standard. When compiling on OpenVMS V7.0, the functions *wcfstime( )* and *wcstok( )* have a second implementation which implement the semantic changes required by this amendment. To retain the XPG4 semantics of these functions, define either *_XOPEN_SOURCE* or *_DECC_V4_SOURCE*.

- Upgrade to Support 4.4BSD Curses

  Document changes to curses.h...

  Note

  The default definitions used during compilation for OpenVMS Alpha have been changed to *__VMS_CURSES* which is the same as OpenVMS VAX. To restore the original default curses package, the user must define *__BSD44_CURSES*.

- FILE Structure Changed to Increase Open File Limit

  Changes were made to the FILE type definition in <stdio.h> to support an extended open file limit in the DEC C RTL. As of OpenVMS V7.0, the number of files which can be open simultaneously will be raised from 256 to 65535 on OpenVMS AXP and 2047 on OpenVMS VAX. This number is based on the OpenVMS sysgen CHANNELCNT parameter which specifies the number of permanent I/O channels available to the system. The maximum CHANNELCNT on OpenVMS AXP is 65535. On OpenVMS VAX it is 2047.

  In order to support more than 256 open files, the field in the FILE type containing the file descriptor "_file" had to be changed from a char type to an int type.

  The definition of the FILE type in stdio.h changed from:

```
typedef struct _iobuf {

    int      _cnt;            // bytes remaining in buffer
    char    *_ptr;            // I/O buffer ptr
    char    *_base;           // buffer address
    unsigned char   _flag;    // flags
    unsigned char   _file;    // file descriptor number
    unsigned char   _pad1;    // modifiable buffer flags
    unsigned char   _pad2;    // pad for longword alignment

} *FILE;
```

  to:

```
typedef struct _iobuf {

    int      _cnt;            // bytes remaining in buffer
    char    *_ptr;            // I/O buffer ptr
    char    *_base;           // buffer address
    unsigned char   _flag;    // flags
    unsigned char   _padfile; // old file descriptor numbe
    unsigned char   _pad1;    // modifiable buffer flags
    unsigned char   _pad2;    // pad for longword alignment
    int             _file;    // file descriptor number

} *FILE;
```

This change was coded using the __VMS_VER macro. As such programs compiled with a version of stdio.h containing support for an increased open file limit can be linked with a version of the DEC C RTL which either does or does not contain this support. Programs compiled with a version of stdio.h providing the new FILE type definition which link on earlier OpenVMS versions obviously not be able to make use of this new functionality.

- Header <stdlib.h> No Longer Includes <types.h>

  As part of feature test macro work, <stdlib.h> no longer includes <types.h>. This will affect some DEC C V5.0 programs that included stdlib.h and expected a type such as pid_t to be defined. The user must change their source to explicitly include <types.h>.

7.11 New Features in DEC C V5.0

- Change to the documentation of /standard=portable

  The meaning of /standard=portable was previously documented as putting the compiler in VAX C mode and enabling the portability group of messages. The DEC C compiler for OpenVMS Alpha prior to V5.0 implemented this behavior, while the DEC C compiler for OpenVMS VAX implemented this qualifier by putting the compiler in relaxed ANSI mode and enabling the portability group. Feedback from users overwhelmingly indicated a preference for the behavior of the VAX compiler. Therefore the V5.0 compiler for OpenVMS Alpha has been changed to behave the same as the VAX compiler, and the documentation for both platforms is being updated to reflect this.

- Major upgrade to the preprocessor

  The part of the compiler that implements preprocessing constructs has undergone a major upgrade. In most ways, the effect of this change should be invisible. Unless you have encountered problems with the preprocessor in previous releases that are now fixed, you should not expect to see much change except perhaps some additional compile-time improvement if you use heavily redundant #includes, or if you use #pragma message to control message reporting for preprocessor constructs. Because of the nature of the changes made, errors and warnings that are detected during preprocessing will typically be reported with different message text and different message identifiers from previous releases. If your code relies on the identifiers or text for messages issued by the preprocessor, you will have to assess the new messages and their identifiers to get equivalent behavior.

  The general nature of the changes to the preprocessor are to improve its reliability and its compatibility with traditional UNIX C compilers, without compromising its ANSI C adherence. In particular:

  - Explicit .I file ouput

    Much more attention has been given to the content of .I files produced by the /preprocess_only qualifier, such that compiling a .I file should more closely mimic the effect of compiling the original source. This includes issues such as the following:

- Generation of whitespace to separate tokens only where necessary to prevent accidental token pasting when the .I file is compiled.
- Generation of # num directives and blank lines to keep a better correspondence to the original source.
- Processing of pragmas and directives (including builtins and dictionary) such that compiling the .I file in the absence of the original header files and/or CDD repository will produce the same effect as compiling the original source in its own environment of header files and repositories.
- #pragma message, standard, and nostandard are now also respected under /preprocess_only, so that spurious diagnostics are not produced when making a .I file.

- Token pasting

  More natural treatment of token-pasting operators that do not produce valid tokens: the pasting is simply ignored.

- Preprocessing directives within macro invocations

  More flexible treatment of the appearance of #if, #ifdef, #ifndef, #else, #elif, and #endif directives within the actual parameter list of a function-like macro invocation that spans multiple lines: the directives take effect. There is no ANSI-required behavior for such constructs, and they can easily appear when a function is changed to a function-like macro. Formerly an E-level diagnostic complaining about the syntax of the directive was issued.

- Missing macro parameters

  More natural treatment of function-like macro invocations with missing actual parameters: each missing parameter is treated like an object-like macro with an empty replacement list.

- Macro expansions in #include and #line

  More complete treatment of preprocessing directives like #include and #line, in the cases where a sequence of tokens requiring macro expansion occurs, and the result of the macro expansion is to be matched against one of the "fixed" forms.

- Error recovery

  Better error recovery for preprocessor-generated diagnostics. In some cases the severity of a similar condition diagnosed by the previous version of the preprocessor has been reduced from an error to a warning or informational because the repair is what would be expected at that level. In particular, C preprocessors are sometimes applied to source code that is not really C code - the expectation is that the preprocessor would give at most a warning or informational, and the detection of an error condition resulting from the fixup made by the preprocessor can safely be left to the compiler's syntactic and semantic analysis phases.

- Macro expansions in #pragma

More usual treatment of #pragma directives: the tokens are not subject to macro expansion. For pragmas that already have a well-established and documented behavior under DEC C, macro expansion is still performed. But for new DEC C pragmas and pragmas offering compatibility with other C compilers, macro expansion is not performed (since most other C compilers do not perform it). If an identifier used as the name of a pragma matches the name of a pragma that is defined not to have macro expansion performed, then no expansion will be performed. But if the identifier is not the name of such a pragma, and it is the name of a currently-defined macro, then that macro gets expanded and the resulting token is compared to the following list of pragma names to determine if the rest of the pragma tokens should be macro expanded. This gives maximum compatibility with existing code, but allows the general behavior to be brought more in line with common practice. The pragmas that will continue to be subject to macro expansion are listed below.

- _KAP
- standard
- nostandard
- member_alignment
- nomember_alignment
- dictionary
- inline
- noinline
- module
- message
- extern_model
- builtins
- linkage (Alpha-only)
- use_linkage (Alpha-only)
- define_template (C++ only)

- #pragma environment

  This new pragma allows saving and restoring the state of all pragmas for which the compiler supports a save/restore stack. It has two additional keywords to specify a state that is consistent with the needs of system header files, or to specify a state that is the same as what was established by command line switches at the start of compilation. The primary purpose is to allow the authors of header files describing data structures that will be accessible to library code to establish a consistent compilation environment in which their headers will be compiled, without interfering with the compilation environment of the user. The syntax is:

```
#pragma environment save
#pragma environment restore
#pragma environment header_defaults
#pragma environment command_line
```

The save and restore keywords cause every other pragma that accepts save and restore keywords to perform a save or restore operation. The header_ defaults keyword sets the state of all those pragmas to what is generally desirable in system header files. This corresponds to the state the compiler would be in with no command line options specified and no pragmas processed - except that #pragma nostandard is enabled. The command_line keyword sets the state of all such pragmas to what was specified by the command line options. See the Users's Guide and help files for a description of pragmas that accept save and restore keywords, and for command line options that control behaviors that are also controllable by pragmas.

- __unaligned type qualifier

  A new type qualifier called "__unaligned" has been introduced which can be used in exactly the same contexts as the ANSI C "volatile" and "const" type qualifiers. It can be used either on a declaration or in a cast to tell the compiler specific places where it should not assume natural alignment on a pointer dereference, without making it apply to all dereferences. This qualifier is provided on OpenVMS VAX only for compatibility with the Alpha platforms. On OpenVMS VAX it is parsed but, has no effect upon code generation. However, it is fully functional on OpenVMS Alpha and Digital UNIX.

- New Predefined Macros

  There are two new predefined macros __DECC_VER and __VMS_VER, which map compiler version numbers and VMS version numbers respectively into an unsigned long int. The compiler version number is extracted from the compiler ident and the VMS version macro is obtained by calling sys$getsyiw(SYI$_VERSION), these string values are then changed into an integer in an implementation defined manner. However, newer versions of the compiler and VMS will always have larger values for these macros. If the VMS version string or the compiler's ident string can not be obtained (or analyzed), then the corresponding macro will be defined with the value zero. Please note that pre-5.0 compilers do not define these macros so it is useful to surround any version tests with #ifdef tests.

```
/*__DECC_VER is not defined before V5.0
  to test for a the compiler V5.1 or higher */
#ifdef __DECC_VER
    #if (__DECC_VER >= 50100000)
        / *Code */
    #endif
#endif

/* to test for VMS 6.2 or higher */
#ifdef __VMS_VER
        #if __VMS_VER >= 60200000
                /* code */
        #endif
#endif
```

Defining __VMS_VER to 60100000 (or lower) is useful if you wish to build your application on VMS 6.2 or higher and link your application with pre-6.2 libraries so that the application can be run by customers running pre6.2 systems. If you define __VMS_VER, you may want to compile with the command line option /WARNINGS=DISABLE= MACROREDEF to suppress that warning message.

- Compile Time Performance Improvements

  Some compile-time improvements have been made for V5.0. The most notable improvement is that the preprocessor is now sometimes able to determine that a particular #include file has already been processed once, if the contents were guarded by a #ifndef FILE_SEEN, #define FILE_SEEN sequence, and if the FILE_SEEN macro is still defined. When the compiler detects this case, that include file will not be reopened and reread.

- XPG4 Support

  DEC C V5.0 supports the worldwide portability interfaces described in the X/Open CAE Specifications: System Interfaces and Headers, Issue 4; System Interfaces Definitions, Issue 4; and Commands and Utilities, Issue 4. These interfaces allow an application to be written for international markets from common source code. This model of internationalization is the same as found on many UNIX systems, including Digital's UNIX.

- /NESTED_INCLUDE_DIRECTORY=(PRIMARY_FILE, INCLUDE_FILE, NONE) has been extended.

  It now accepts the NONE keyword. This specifies that the compiler skips the first step of processing #include "file.h" directives. It starts looking for the included file in the /INCLUDE_DIRECTORY directories. It does not start by looking in the directory containing the including file nor in the directory containing the top level source file.

  ```
  Default: /NESTED_INCLUDE_DIRECTORY=INCLUDE_FILE
          (current behavior)

  UNIX equivalents:
   -nocurrent_include  /NESTED_INCLUDE_DIRECTORY=NONE
  ```

- /MMS_DEPENDENCY[=(FILE[=filespec] | [NO]SYSTEM_INCLUDE_FILES ,...)] /NOMMS_DEPENDENCY has been added

  The qualifier /MMS_DEPENDENCY tells the compiler to produce a dependency file. The format of the dependency file is similar to that on Digital UNIX.

  ```
  object_file_name :<TAB><source file name>

  object_file_name :<TAB><path to first include file>
  object_file_name :<TAB><path to second include file>
  ```

  The FILE subqualifier specifies where to save the dependency file. The default file extension for a dependency file is .mms. Other than using the different default extension, this qualifier uses the same procedure as /OBJECT and /LIST for determining the name of the output file.

The SYSTEM_INCLUDE_FILES subqualifier specifies whether or not to include dependency information about system include files. That is, those included with #include <filename>. The default is to include dependency information about system include files.

```
Default: /NOMMS_DEPENDENCY

UNIX equivalents:
    -M              /MMS_DEPENDENCY=SYS$OUTPUT
    -MD             /MMS_DEPENDENCY
    -MM             /MMS_DEPENDENCY=
                         (FILE=SYS$OUTPUT,
                          NOSYSTEM_INCLUDE_FILES)
    default         /NOMMS_DEPENDENCY
```

- /[NO]LINE_DIRECTIVES has been added.

  This qualifier governs whether or not #line directives appear in preprocess output files. Currently, there is no way to specify the form of the line directives. DEC C and C++ always generate the traditional pcc and VAX C form omitting the "line" naming the directive, even in the ANSI modes of the compiler.

```
Default:    /LINE_DIRECTIVES

UNIX equivalents:
    -P                  /NOLINE_DIRECTIVES
    -E                  /LINE_DIRECTIVES
```

- /COMMENTS=(AS_IS,SPACE) /NOCOMMENTS has been added

  This qualifier governs whether or not comments appear in preprocess output files. If they do not appear, it specifies what replaces them.

  AS_IS specifies that the comment appears in the output file. SPACE specifies that a single space replaces the comment in the output file. /NOCOMMENTS specifies that nothing replaces the comment in the output file. This may result in inadvertent token pasting.

  The C and C++ preprocessor may replace a comment at the end of a line or a line by itself with nothing, even if /COMMENTS=SPACE is specified. Doing so cannot change the meaning of the program.

```
Default:
    /NOCOMMENTS in the VAXC and COMMON modes of the
                C compiler
    /COMMENTS=SPACE for C++ and the ANSI modes of
                the C compiler
    An explicit /COMMENTS on the command line
       defaults to /COMMENTS=AS_IS.

UNIX equivalents:
    -oldcomment         /NOCOMMENTS
    -C                  /COMMENTS=AS_IS
    default             /COMMENTS=SPACE
```

- /[NO]VERSION has been added.

  This is a completely new qualifier intended to make it easier for users to report which compiler they are using.

This qualifier causes the compiler to print out via printf the compiler version and platform and to exit without performing any other actions. Because of the DCL syntax for the cc command, it is still necessary to specify a source file name. (The null device, NL:, is adequate for this purpose - it is not necessary to have a real source file).

The compiler version is the same as in the listing file.

```
Defaults:               /NOVERSION

UNIX equivalents
    -V                  /VERSION
    default             /NOVERSION
```

- #dictionary Support

  The #dictionary directive has been enhanced. The syntax and features are the same as for DEC C V4.0 on OpenVMS/VAX, except that V5.0 will also accept two new keywords, text1_to_array and text1_to_char.

```
#[pragma] dictionary "pathname" [null_terminate]
             [name(struct_name)]
             [text1_to_array | text1_to_char]

pathname: a quoted pathname for a CDD record to be
          extracted

null_terminate: an optional keyword which adds an
          additional byte for the null character
          when a data type of text is extracted

name():   an optional keyword to supply an alternate
          tag name or declarator, struct_name, for the
          outer level of a CDD structure.

text1_to_char: an optional keyword which forces the
          CDD type text to be translated to char,
          rather than array of char if the size is 1.
          This is the default, unless null_terminate
          was specified.

text1_to_array: an optional keyword which forces the
          CDD type text to be translated to type array
          of char, even when the size is 1. This is the
          default when null_terminate is
specified
```

7.12 New Features in DEC C V4.0

Since the field test of DEC C V1.3, the following major new features have been added to DEC C T4.0.

- DEC C/C++ Run-Time Components Kit

  DEC C requires the DEC C Run-Time Library. This is part of OpenVMS VAX systems V6.0 and later but is not part of OpenVMS VAX V5.5 systems. The DEC C/C++ Run-Time Components Kit makes it possible to install the DEC C Run-Time Library on OpenVMS VAX V5.5 systems, thus allowing DEC C and DEC C++ built applications to run on OpenVMS VAX V5.5 systems.

  See the documentation for the DEC C/C++ Run-Time Components Kit for more information.

- Builtin Function Support

DEC C supports the non-privileged builtin functions supported by VAX C. These builtins are:

_ADAWI
_BBCCI
_BBSSI
_FFC and _FFS
_INSQHI, _REMQHI, _INSQTI, _REMQTI, _INSQUE, and _REMQUE
_LOCC
_MOVC3 and _MOVC5
_MOVPSL
_PROBER and _PROBEW
_SCANC, _SKPC, and _SPANC

The *DEC C User's Guide for OpenVMS Systems* describes how to use these builtin functions.

- The VAX C to DEC C Migration Guide

  This document gives pointers on migrating applications from VAX C to DEC C.

- DEC C and DEC C++

  DEC C and DEC C++ share a common location for DECC$LIBRARY_ INCLUDE. This director contains the header files used by both DEC C and DEC C++. CXX$LIBRARY_INCLUDE contains the header files used only by DEC C++.

  DEC C and DEC C++ share a common startup command procedure, SYS$STARTUP: CCXX$STARTUP. This startup command procedure replaces the older individual startup command procedures.

  DEC C V5.0 and later and DEC C++ V1.3 and later may be installed in either order. The kit installation procedure will not replace header files in DECC$LIBRARY_INCLUDE if they are more recent than those in the kit.

- SCA support

  Many problems with DEC C's support of the Source Code Analyzer have been fixed.

- POSIX support

  DEC C supports POSIX for OpenVMS VAX V1.2. See the POSIX documentation for more information.

7.13 Problems fixed in Compaq C V6.4

- token-pasting hex constants

  There was a problem using the preprocessor's token-pasting operator with operands consisting of a pp-number containing the letter E or e. In some case the operand would be treated as two separate tokens in modes other than VAXC or COMMON. E.g. in the following example, the initializer for "a" was correct, while the initializer for "b" was expanded to "0x0E a" instead of "0x0Ea".

```
#define XXX(xxx) 0x##xxx
int a = XXX(0Aa);
int b = XXX(0Ea);
```

- backslash-newline in COMMON mode

Under /MODE=COMMON/PREPROCESS_ONLY, the compiler would sometimes recognize a backslash-newline sequence outside of the context of a string literal, a character constant, or a preprocessing directive, and produce an output file reflecting logical line-splicing instead of leaving the backslash-newline sequence intact in the output. This problem has been corrected as shown:

```
Input:
    char str1[10] = "A"\
                    "Z";
    char str2[10] = 'A'\
                    'Z';
Incorrect output for /stand=common/preprocess_only:
    char str1[10] = "A"
                    "Z";
    char str2[10] = 'A'                    'Z';
Corrected output for /stand=common/preprocess_only:
    char str1[10] = "A"\
                    "Z";
    char str2[10] = 'A'\
                    'Z';
```

- Incorrect behavior of /PREFIX

  If a qualifier either of the form /PREFIX=RTL= or /PREFIX=EXCEPT= was specified on the command line, then the compiler would erroneously omit prefixing any routines (as if /NOPREFIX had been specified). This behavior was present in all previous versions Compaq C for VAX, and has now been corrected.

- Compiler failure for long source lines A source line longer than 32767 characters would trigger an internal compiler error. It now will issue a meaningful (but fatal) diagnostic.

7.14 Problems fixed in Compaq C V6.2

- excessive compile time for long initializer lists

  Extremely long initializer lists (~100K initializer elements) caused apparent looping in the compiler, which was actually an n**2 algorithm introduced in V6.0 along with initializers with designators. This has been corrected.

- compiler failure with /SHOW=EXPANSION

  When compiling with /LISTING/SHOW=EXPANSION or -source_listing -show expansion the compiler could accvio if there was a macro which expandanded to a string of 32737 characters or more. The compiler will now truncate the expanded macro text.

7.15 Problems fixed in DEC C V6.0

- The preprocessor output of wide-string literals explicitly present in the source failed to include the leading L character, turning them into ordinary string literals when the preprocessed output was compiled.

7.16 Problems fixed in DEC C V5.7

- Incorrect optimization comparing an unsigned bit-field consisting of a single bit in the middle of a byte has been fixed. E.g., the following code erroneously printed 1 when compiled with optimization.

```
#include <stdio.h>
static struct {
    unsigned  a : 6;  /* Removing this solves the problem */
    unsigned  b : 1;  /* Changing size to 2 also solves it */
} x = {0};
int badproc(void)
{
    if (x.b > 0)
        return(1);
    else
        return(0);
}
int main(int argc, char **argv)
{
    printf("%d\n", badproc());
    return 0;
}
```

- Case EVT102462 is fixed: SCA files for large compilations generated under the /ANALYSIS_DATA qualifier no longer causes SCA-F-NOTERMINATE from the SCA LOAD command when there are more than 65535 SCA events.

- DEC C now correctly handles array data members of function return results (for functions returning struct or union values). E.g. the following code no longer generates %CC-E-NEEDPOINTER:

```
typedef struct {
    struct {
        int field4;
        int field5[10];
    } field3;
} struct_t;

struct_t return_struct(void);

void foo (struct_t y) {
    int x, z;
    z = y.field3.field5[0];
       /* generate NEEDPOINTER diagnostic below */
    x = return_struct().field3.field5[0];
}
```

- In previous compilers, enumeration constants with values beyond the range of type int were silently truncated to int. These now cause a warning to be issued.

- Certain invalid preprocessing directives could cause an infinite loop in the compiler. E.g. #module 0badident, #include directives with mismatched delimiters.

- Certain uses of the token-pasting operator to form valid hexadecimal constants were incorrectly treated as errors. E.g., the following produced an error but is now handled correctly:

```
#define SVAL(a,b) 0x ## a ## b
int i = SVAL(23a2,0bae);
```

- The compiler no longer issues a pointer mismatch warning when a pointer to an array of unqualified element type is passed to a function declared with a parameter declared as a multiply-dimensioned array type with a type qualifier. The warning was erroneous, as the types are compatible under ANSI C. E.g., the following no longer produces a warning:

```
extern void func1(const double dbl[][3]);
void func2(void) {
  double  adPnts[8][3];

  func1 (adPnts);
}
```

- The compiler now warns when two different names with external linkage in the same compilation are mapped to the same name in the object module (e.g. due to truncation or forcing to monocase).

- The size of conditional operator (i.e. "?:") expressions was not being computed correctly when the values were character string constants. They were being treated as arrays instead of pointers. E.g. the following produced an error instead of evaluating to the size of a pointer-to-char:

```
int is_what;
int b;
void f(void)
{
   b = sizeof(is_what ? "one" : "testing");
}
```

7.17 Problems fixed in DEC C V5.6

- A memory corruption problem within the compiler was fixed. This bug has been present since the first release (V4.0), and could only manifest symptoms under extremely rare circumstances involving the exact number and length of file specifications, the order in which they were processed during early stages of compilation, and the initial state of the runtime heap managed by lib$get_vm. The symptom would typically be a BUGCHECK that would not reproduce when compiling the same source code in a different process environment.

- A compilation containing an ANSI function prototype for a variable argument list using ellipsis syntax, with a mismatched K&R style definition for that function, provoked the warning "%CC-W-FUNCREDECL, In this declaration, function types differ because one has no argument information and the other has ellipsis". But the compiler then access violated after issuing the warning.

- Compiler crash and/or memory exhausted system hang using /debug qualifier with #include <exc_handling.h>, or processing any other struct or union type containing a member that is a pointer to a qualified version of the same struct or union type.

- When compiling a comma-separated list of source files with /list, the listing section containing the values of predefined macros would not be present in the listing files for the second and subsequent compilations. In some cases, the compiler would crash.

- Compiler would sometimes crash after emitting %CC-W-NOLINKAGE.

- Under /STAND=VAXC, inform/warn about use of "signed" keyword, which was not supported by VAX C.

- Compiler would incorrectly use filename information from #line directives to name the object module. Only the primary source file name and #module (#pragma module) directives are supposed to affect the object module name.

- Permit array indexing operator to be used on an array member within a struct or union that is a function return value. The compiler now correctly handles array data members of function return results for functions returning struct or union values.

- The /VERSION qualifier no longer creates a 0-length .OBJ file, and it no longer requires a filename to be specified. CC/VERSION (or CC/DECC/VERSION) is sufficient.

- Fixes to VAXC and MS compatibility modes.
- Fixes to /ANA processing.

7.18 Problems fixed in DEC C V5.5

- The globalvalue extern_model could produce a compiler traceback when an external variable of a struct or other type not supported by globalvalue was encountered. The compiler now produces an appropriate warning message and uses the strict_refdef model for such declarations.

- SCA typing information for incomplete arrays declared with a typedef declaration were sometimes incorrect and complex declarations caused compile time access violations. Known instances of these problems have been corrected.

- Placement of certain kinds of pragmas within declarations could cause compiler failures or erroneous .ANA files to be produced uner /ANALYSIS_ DATA.

- Forward declarations involving structure members no longer cause invalid .ANA files to be produced when compiling with /ANALYSIS_DATA

- Invalid .ANA files are no longer produced when compiling with /ANALYSIS_DATA for source files with code beyond column 256.

- Better SCA reference information is output for struct declarations and references.

- Functions with an excessive number of parameters (around 100) will no longer cause the compiler to crash when outputting SCA information. Instead, a warning message is emitted and further parameters will not generate SCA information.

- Debugger support for pointers to functions has been improved.

- Improved initialization for redeclared variables. Previously the compiler would not initialize a in the following code

```
int a;
static int a = 1;
```

- Issue a warning for a cast expression used as an lvalue in strict ANSI mode.

- A #undef or #define preprocessing directive appearing within the arguments to a function-like macro invocation could cause the compiler to crash (e.g. if it undefined or redefined the function-like macro being invoked). This now produces a warning message.

- Source files containing very large numbers of #line preprocessing directives could cause a seemingly infinite loop in the compiler due to an n**2 algorithm. The algorithm is now linear.

- A number of changes have been made to improve compatibility with VAX C, Microsoft C, and UNIX compilers. New messages have been introduced and old ones altered to mimic more accurately the behavior of these other compilers when using the /STANDARD switch. Some of the these compatibility changes are:

  - Do not accept an array of incomplete types.

  - Do not allow constructs like "(z ? x : y).a = 1;"

  - Issue warning for pointer to int assignment.

- Do not allow pointer types as case constants.

- Disable processing of type qualifiers after a comma.

- Handle redeclaration of items with different storage classes.

- Implement VAX C style unnamed struct/union members in VAXC mode, where the struct/union type of the unnamed member is treated as if it occurred outside of any containing struct/union. In MS mode, the same construct is treated like a C++ unnamed member of union type in C++: the members of the nested struct/union are promoted to the containing struct/union, similar to the behavior of VAXC variant_struct and variant_ union constructs.

- Allow "x->y" under /STAND=VAXC where x is an integer constant or variable.

- Under /STAND=VAXC, "sizeof(&array)" now returns "sizeof(array)" (i.e. the size of the array object) instead of the size of a pointer (which is what ANSI C requires in this case).

- Emit meaningful error message for a[b] when both a and b are pointers.

- Emit informational in VAXC mode for declarations with an explicit "extern" storage class which are also initialized.

- Issue a warning message for cases like a->b where a is a pointer to an incomplete struct/union.

- In MS mode, allow an incomplete array type only as the last member in a struct or union. Previously, other incomplete types were accepted, and they were not restricted to the last member.

- Allow ellipsis in old-style function definitions under /STAND=VAXC.

- Emit language extension messages for __restrict and __unaligned in /STAND=PORT

- Improve handling of VAX C storage class modifiers (noshare, readonly, _align) to more closely match the VAX C compiler's behavior.

- In modes other than VAXC, emit additional diagnostics when a function is redeclared with a different type.

- Emit new warning message for the long double type on VAX, which does not support long double with greater precision than double.

7.19 Problems fixed in DEC C V5.3

This kit contains the following bug fixes beyond those contained in DEC C V5.2:

- A problem involving, among other factors, the exact lengths of file names processed by the compiler sometimes caused the compiler to bug check. Immediately preceding the bug check was an error message giving the text of a `#include` directive with no error marker pointing at a location within the directive.

- V5.2 introduced a bug which caused logical name translation to be applied to the names of text library modules.

- The /NESTED_INCLUDE=PRIMARY_FILE qualifier was intended to provide behavior compatible with the VAX C compiler. However, the DEC C implementation of this feature on VAX never handled the "sticky defaults" for top-level source files the way that VAX C did (some of the Alpha versions did provide this behavior). In V5.3, both VAX and Alpha interpret this qualifier to produce behavior that is compatible with VAX C.

- Problems involving the use of variable argument lists (<stdarg.h> or <varargs.h>) were fixed. These involved old-style function parameters of types requiring default argument promotion (now get an E-level diagnostic), and the use of a va_list variable that was not a simple locally-declared variable (now handled correctly).

- Diagnostic messages containing character constants using a hex escape sequence were being printed with the "x" character omitted.

- The preprocessor now allows the "defined" operator to be produced as a result of macro expansion. ANSI C does not specify whether or not this should be allowed. Under /standard=ansi89, a warning message is produced when the preprocessor encounters such a construct. In relaxed ANSI an informational is issued instead.

- Under /stand=vaxc, the type of the sizeof operator is now signed int (matching the behavior of VAX C) rather than unsigned int (which is required by ANSI C).

- A block-scope extern declaration with incomplete type, with linkage to a file-scope declaration with complete type, "blocked" the completed type information from being available within the scope of that block. This could produce an E-level diagnostic for a correct program such as the following:

```
int i[] = {1, 2, 3};
unsigned int f()
{
    extern int i[];
    return sizeof(i);
}
```

- An initialized globalvalue may now be used as a compile-time constant, just as an enumeration value can be.

- For a compilation with multiple source files in a comma-list, the module names in the .ana files produced under the /ana qualifier were incorrect for all but the first compilation.

- Under /standard=vaxc, match the VAX C behavior and accept a sequence of conflicting type specifiers with a warning and use the last one, instead of issuing an E-level diagnostic.

- Statically initialized data declared with the "const" qualifier was sometimes being placed in the $CODE psect even if it contained address constants that could cause $CODE to become non-sharable. Now this data is placed in a new psect $ADDRESS_DATA so that the $CODE psect will remain sharable.

- SPR HPAQ628CF: Static initialization using a globalvalue as an initializer produced incorrect initialization if the type of object being initialized was not int (e.g. if it was a char or short type).

- The compiler now allows an actual argument to a function call to pass the address of a globalvalue; formerly this produced an E-level diagnostic even in VAX C mode. Also changed the mechanism for passing the address of a constant such that if the callee modifies the value it will not affect subsequent calls. This makes the behavior compatible with VAX C.

- The compiler failed to diagnose an ANSI constraint violation for an old-style function definition with a parameter identifier list, where one of the parameter names matches a visible typedef name.

- The compiler failed to diagnose use of the name of a variant_struct or variant_union member when constructing the name of a contained member. This is a VAX C feature, and VAX C compiler produces an E-level diagnostic if the name is used in this way.

- The compiler was incorrectly producing an E-level "invalid declarator" diagnostic when a label was defined with the same name as a typedef. Labels have a separate namespace and should not conflict with other declarations.

- The compiler would fail if it attempted to output a diagnostic message referring to an unnamed bit field in a struct or union declaration.

- Under /stand=common, hexadecimal escape sequences (\xnnnn) were being recognized within string literals and character constants. This feature was part of VAX C and ANSI C, but it was not present in "pcc" compilers, and recognizing it under /stand=common produced results that differed from pcc compilers.

- SPR HPXQ11CEF: Compiler failure when globaldef storage class is applied to a function. The compiler now gives an appropriate E-level diagnostic when globaldef, globalref, or globalvalue storage class is applied to a function.

- Compiling a function definition that attempts to use a typedef for its function header could either cause a compiler failure or produce an inappropriate diagnostic, e.g.:

  ```
  typedef void VFV();
  VFV f;
  VFV f {}
  ```

  This now produces the correct diagnostic: %CC-E-TYPEDEFFUNC, In this function definition, "f" acquires its type from a typedef.

- If a globaldef declaration specified a psect name, the psect specification would be ignored if the same variable was previously declared in a globalref declaration.

- A block-scope declaration with the extern storage class would cause a compiler failure if the identifier being declared matched a visible typedef name declared at file scope.

- Compiler failure when attempting to output a fatal error message. The following example produced a compiler failure trying to output the fatal error message for include file not found. The root cause involved the specific placement of the pragmas in relation to the declaration:

  ```
  struct my_s {
  #pragma message save
  #pragma message disable portable
     int one;
     int two;

  #pragma message restore
  };
  #include <i_dont_exist.h>
  ```

- When the VAX C "main_program" keyword was used to identify a main program procedure with a name other than "main", the compiler still generated a global procedure named "__MAIN". This made it difficult to put more than one such object module into a single object library. The compiler now makes the "__MAIN" symbol it generates in this case a local symbol.

- Under /STAND=VAXC, an incompatibility between a function prototype and the corresponding function definition produced only a W-level diagnostic, but calls to such a function were silently ignored by the compiler, causing incorrect behavior at runtime. The diagnostic remains at W-level, but now the generated code is correct.

- SPR UVO104030: Compiler failure under /preprocess_only when processing a pragma with incorrect syntax.

- DEC C did not evaluate string literal and character constant hex escape sequences correctly in the /STANDARD=VAXC and /STANDARD=COMMON modes of the compiler. Note that hex escape sequencences are not supported in common mode.

7.20 Problems fixed in DEC C V5.2

This kit contains the following bug fixes beyond those contained in DEC C V5.0-003:

- The 64K source line limitation on debugging C modules, previously identified as a restriction, has now been removed.

- VMS source files with variable length record format could sometimes provoke the message "%CLI-F-READERR, error reading filename, -RMS-W-RTB, n-byte record too large for user's buffer." even for records much smaller than 32K. This problem has been fixed. But note that the advertised maximum source line length of 32767 bytes includes an implicit newline terminator even for file formats that do not explicitly represent the newline terminator charactor, and there are some other boundary conditions with listings and error messages that effectively reduce the actual line length fully supported to a few characters less than 32767.

- /LIST/SHOW=NOSOURCE will no longer cause a compile time accvio

- CC/DECC/PREPROCESS_ONLY=filename SYS$INPUT no longer fails on OpenVMS/VAX.

- SPR HPXQB8B17. Incorrect /OPT on VAX

Previously, DEC C/C++ erroneously put w in a register in the following case

```
unsigned int w = 17;
g(*(struct B *)&w);
```

- DEC C will now Skip parameter checks if we are evaluating an operand to sizeof in common mode as in the example below:

```
j(int a)
{
p(sizeof(j()));
}
```

  The severity of the messages concerning too many and too few parameters is now a warning in VAXC mode (/STAND=VAXC) rather than an error. This behavior is compatible with VAX C.

- Error messages have been relaxed for conflicting extern defs as follows:

  For /STAND=COMMON there is no change in behavior.
  All extern declarations are promoted to file scope. When the compiler encounters a conflicting declaration, it will issue a error as it has always done.

  For /STAND=VAXC the severity of the error message has been reduced to a warning.
  VAX C issues a warning whenever it finds a conflicting extern declaration. It does not matter if the declarations are in the same name scopes or not. In addition to issuing a warning, VAX C replaces the prior declaration with the new declaration from the point of the new declaration onward. DEC C now matches this behavior in VAXC mode.

  For /STAND=ANSI, /STAND=RELAXED
  Errors will be generated, (as they always) if there is a conflict between declarations that are in the same or inner name scopes.

  DEC C will issue a warning if there is a conflict between names that are in disjoint scopes. This will no longer be an E-level message. The standard says that such a case is in error, but that a diagnostic does not have to be issued. We felt that it was better to issue a diagnostic than to silently accept the program.

  For example, in the program shown below:

  /STANDARD=COMMON will result in no diagnostic messages.
  /STANDARD=VAXC will result in a warning message about incompatibile declarations for the second and third declaration of init_color
  /STANDARD=RELAXED, /STANDARD=ANSI will result in an informational diagnostic on the second declaration of init_color because it is implicitly declared and will result in a warning on the third declaration because it is incompatible with the declaration on line 3, even though it is in a different scope.

```
main()
{
  extern void init_color();
}
```

```
Raise_Maps()
{
 init_color();
}

Title_Page()
{
  extern void init_color();
}
```

- DEC C will no longer report a constant overflow if the left shift operator causes the sign bit to change value. The ANSI C definition of the left shift operation renders overflow inapplicable.

- A problem has been corrected which could cause an ACCVIO at compile time when compiling with the qualifier /ANALYSIS_DATA.

- In all modes, functions declared at block scope will now have their storage class set to extern. A warning is issued if the storage class is register. A warning is also issued if the storage class is auto (except in VAXC mode). If the storage is static, in common and vaxc mode then no warning is issued for the declaration. But a warning will issued later if the function is referrenced and not defined anywhere in the module.

- SPR EVT101335
  Whenever a call causes more than 255 items (longwords on VAX/quadwords on Alpha) to be used in constructing the arg list a warning will be issued. On Both VAX and Alpha an informational will be output warning that the argument list length exceeds maximum specified by the calling standard.

```
/* This program used to ACCVIO on VAX/VMS, now
   it gets a compile-time diagnostic */
struct {
        int i;
        char longer_than_1020[1021];
       } public_domain_sloppy_programmer;
void nothing();
main ()
{
 nothing (public_domain_sloppy_programmer);
}
```

- SPR UVO102632 Formerly the compiler sometimes failed to issue a diagnostic when an assignment was made to a constant array as in the example below:

```
void fred (void)
{
   typedef int A[2][3];
   const A a = {{4, 5, 6}, {7, 8, 9}};

   a[0][0] = 42;
}
```

- The compiler will now accept unnamed structures as members of a struct in VAXC mode.

- The compiler will now issue a warning instead of an error when pointers and ints are compared in common mode.

- The compiler will now issue a warning when preprocessing directives are used in the argument list for a macro

- The compiler will now Allow more than just integers in switch and case expressions in vaxc, and common modes. We now issue a new warning when a float or pointer is used in a switch or case expression.

- A problem has been corrected involving the #dictionary directive when it was nested within a structure declaration. The compiler now correctly generates a member name for the extracted CDD record nested within a struct, not a tag name.

- The macro definitions within a /define=(name[=value],...) list are now processed left to right. Thus /DEFINE=(A=1,A=2) now leaves A defined as 2 instead of 1.

- Problems with the /NESTED= qualifier have been fixed.

- The severity of the NONMULTALIGN message has been reduced to a warning.

- Several problems in computing the value of an integer constant constructed through token-pasting in the preprocessor have been fixed. E.g. the following code formerly resulted in an incorrect message "%CC-W-INVALTOKEN, Invalid token discarded".

```
#define concat(a,b) a ## b
return concat(0x0,1AL) ;
```

  It now is handled correctly.

7.21 Problems fixed in DEC C V5.0

- DEC C in VAX C mode will now do macro subsitution in all three cases below, just as VAX C did. Given the macros:

```
#define str1(arg) "arg"
#define str2(arg) "arg-ber"
#define str3(arg) "go to the arg"
```

- ICA-48945: mixing of old-style and new style function prototypes:

The compiler now allows mixing of new-style function prototypes and old style function definitions where the prototype parameters are not fully promoted integer types (according to default argument promotion rules). With this modification, all integer type combinations are allowed (including signed/unsigned mixing). A warning is issued where we were issuing an E level error in the past (no message is issued if in VAXC mode and the integer types in the old style parameter definition match those in the prototype, as in the code fragment provided in the SPR).

```
void f (char);
void f (p1)
char p1;
{}
```

```
$ cc/stand=vaxc foo.c
$ cc foo.c
char p1;
.....^
%CC-W-PROMOTMATCHW, In the definition of the
function "f", the promoted type of p1 is
incompatible with the type of the corresponding
parameter in a prior declaration.
at line number 3 in file DISK:[dir]FOO.C;1
```

In addition, the following will now correctly compile:

```
extern in (*f1())(int (*f2)());
int (*f1(f2))()
int (*f2)();
{ return 0;}
```

- HPXQ7084C CDD datatype text size 1, can now be converted be converted to an array of char or to a char using the new #dictionary keywords, text1_to_ array, text1_to_char.

- The compiler will now give a diagnostic if C++ style comments are used with /STANDARD= ANSI89.

- ICA49522 In vaxc mode, the compiler will now tolerate declarations which contain redundant use of the type qualifer "volatile".
- DEC C used to issue messages for lexical "errors" appearing within the bodies of macro definitions for macros that were never used. In some cases these should not have been issued according to ANSI C (e.g. warnings for octal constant containing digits 8 or 9), and generally such potential problems do not require an ANSI diagnostics (e.g. the effect of unterminated character constants is undefined). Common practice is to defer such reports until a macro is used, which is what DEC C now does.
- The result of compiling the output of the /PREPROCESS_ ONLY qualifier was not always the same as the result of compiling the original program. Consider the program below.

```
#define A(x) -x
main() {
    int i = 1;
    printf("%d\n", -A(i));
}
```

The output from /PREPROCESS_ONLY used to place the '-' of the body of macro A next to the '-' before the macro invocation, producing:

```
printf("%d\n", --i);
```

Now the output has a space to separate the two '-' characters to prevent this accidental token-pasting unless the compiler is in common or vaxc modes, where this kind of token-pasting is done when compiling the original source directly.

- curses.h

  Changes have been made to improve the functionality of the default curses package.

- float.h

  On OpenVMS Alpha the D_FLOAT definitions of DBL_MAX and LDBL_MAX were corrected.

- fp.h

  The new header file <fp.h> implements some of the features defined by the Numerical C Extensions Group of the ANSI X3J11 committee. Applications making extensive use of floating point functions may find this useful.

  Some of the double precision DEC C RTL functions return the value ±HUGE_VAL (defined in either math.h or <fp.h>) if the result is out of range. The float versions of those functions return the value HUGE_VALF (defined only in <fp.h>) for the same conditions. The long double versions return the value HUGE_VALL (also defined in <fp.h>).

- math.h

  The D_FLOAT definition of HUGE_VAL was corrected on both OpenVMS VAX and OpenVMS Alpha.

- ints.h

  Definitions for (u)int16 and (u)int32 were added for use by DEC C++ programs on OpenVMS VAX. This will allow DEC C programs using (u)int16 or (u)int32 to be portable to DEC C++ on OpenVMS VAX.

- perror.h

  Definitions for decc$ga_sys_errlist and decc$gl_sys_nerr were added for use by DEC C and DEC C++ programs. These are provided for compatibility with VAX C programs that made use of sys_errlist and sys_nerr.

- setjmp.h

  A prototype for decc$setjmp was added.

- stat.h

  Macros defining constants for group and other protection masks were added to match the ones for 'owner'.

- stdarg.h

  A definition for va_count was added.

- stdio.h

  Modifications were made to the definitions of clearerr, feof, ferror such that proper usage of these macros does not give warnings when compiling /WARNING=ENABLE=CHECK.

- unixlib.h

  Prototypes were provided for the following routines on OpenVMS VAX: decc$to_vms, decc$from_vms, decc$match_wild, decc$fix_time, decc$translate_vms.

7.22 Problems Fixed in V4.0-01

1. DEC C now recognizes comment terminators which span multiple lines through the use of backslash, newline line continuation.

   ```
   /* comment *\
   /
   ```

2. The DEC C kit now installs properly on OpenVMS VAX V5.5-2.
3. The DEC C kit now supplies the correct header files on systems on which DEC C++ V1.2 was previously installed.
4. The DEC C kit now ships CMA$DEF.H. Previous kits generated CMA$DEF.H from SYS$LIBRARY:STARLETSD.TLB, and the generated version was not correct.
5. If DEC C encounters a bad multibyte character in a compilation source, it attempts to skip past the character and continue compilation.
6. In addition to `#pragma inline <function name>`, you can suggest inlining of a function with `__inline`. Below, both func1 and func2 are candidates for inlining:

   ```
   __inline void func1(void} {}
   void func2 (void) {}
   #pragma inline func2
   ```

7. Functions declared volatile or const via a typedef:

   ```
   typedef int (F)();
   const volatile F abs;
   ```

   now produce a CONSTFUNC or VOLATILEFUNC diagnostic.

8. Redundant type specifiers (e.g. `int int x;`) now produce warning diagnostics, not error diagnostics. Warning diagnostics may be suppressed with /WARNING=DISABLE; error diagnostics may not be suppressed.
9. Similarly, improper use of register variables, such as:

   ```
   register int x;
   int *p = &x;
   ```

   produces a warning diagnostic, not an error diagnostic.

10. The redundant use of a type qualifier of a pointer (e.g. `int * const const p;`) produces a warning diagnostic.
11. #include "'file.h'" no longer crashes the compiler.
12. DEC C supports C++ style comments in all modes except /STANDARD=ANSI89. DEC C cannot support C++ style comments in this mode because they are not allowed by the C standard.
13. DEC C for OpenVMS VAX supports the /NESTED_INCLUDE_DIRECTORY command line qualifier. The default behavior is /NESTED_INCLUDE_ DIRECTORY=INCLUDE_FILE. This matches the behavior of VAX C. See the

DEC C User's Guide for OpenVMS Systems for more information about this qualifier.

14. #pragma __nostandard and #pragma __standard work correctly. In some cases, the pair of pragmas were not suppressing some messages that should have been suppressed, and leaving other messages suppressed that should not have been suppressed.

15. Most of the header files supplied with this kit or created during installation compile with no diagnostics in all modes of the compiler and with all diagnostics enabled. Some exceptions are:

    - Headers files whose names contain $ give a DOLLARID diagnostic when compiled with /WARNINGS=ENABLE=ALL. An identifier containing the name of the file is used before the #pragma nostandard occurs.
    - The generated CVT$ROUTINES.H, new in OpenVMS VAX V6.1, does not compile in any mode of the compiler.

16. The DEC C kit contains the header file, builtins.h.

17. DEC C no longer requires definitions for unreferenced declared objects. The following program no longer produces the linker diagnostic %LINK-W-UNDFSYMS:

```
extern int x;
#include <stdio.h>
int main(void) {
        printf("Hello world\n");
}
```

18. DEC C no longer inlines functions that use va_start.

19. In the /STANDARD=VAXC, /STANDARD=RELAXED_ANSI89, and /STANDARD=COMMON modes of the compiler, a redeclaration of a function with an empty argument list is now compatible with previous declarations containing an ellipse. The following function declarations are now compatible:

```
int x(int first, ...);
int x();
```

    As required by the C Standard, DEC C does gives a FUNCREDECL diagnostic for this in the /STANDARD=ANSI89 mode of the compiler. DEC C also gives this diagnostic when you specify /WARNING=ENABLE=PORTABLE since such redeclarations are not compatible in all implementations of C.

20. Passing a pointer to an array of pointers to const chars to a function expecting a pointer to a pointer to an array of pointers to non-const chars now results in the following diagnostic:

```
static void f(char *argv[]) {}
static void g(const char *argv[])
{
    f(argv);
%CC-W-PTRMISMATCH, (1) In this statement, the referenced
 type of the pointer value "argv" is "pointer to const
 char", which is not compatible with "pointer to char".
}
```

21. DEC C supports the __int32 and __int16 datatypes.

7.23 Restrictions and known bugs

This is a list of some known compiler restrictions and bugs.

- The source file names and line numbers put into the object module by the compiler for use by the debugger always reflect the actual filespecs used by the compiler to access files during compilation, and do not reflect any filenames or line numbers specified explicitly by `#line` directives within the source. This is a permanent restriction on the VAX.
- The core "Transform" code of the MD5 message-digest algorithm, for which there are public-domain versions and copyrighted versions from RSA Data Security, cannot be compiled with optimization. This code involves computation of a 16-byte hash value given an old 16-byte hash value and 64 bytes of new data. The code represents the 16-byte hash value as 4 unsigned ints, and proceeds in 64 steps, where each step involves three computations and assignments to one of the 4 unsigned ints constituting the hash value. The resulting straight-line sequence of 192 assignments involves many complex expressions with values flowing through the assignments. This triggers a pathological case in the optimizer that exhibits n**2 behavior in the number of steps to be optimized, which effectively appears to be an infinite loop at compile time. The code can be correctly compiled only using `/optimize=nodisjoint`.

  Some implementations of the algorithm do not mention the name MD5, but the code can be recognized as follows. Given 4 integer variables (h1, h2, h3, h4) holding the hash value, after macro expansion the first of the 64 steps looks like:

  ```
  h1 += (h4 ^ (h2 & (h3 ^ h4))) + input[0] + 0xd76aa478;
  h1 = h1 << 7 | h1 >> 25;
  h1 += h2;
  ```

- Under /NESTED_INCLUDE_DIRECTORY=PRIMARY_FILE, the sticky-default processing for included files ignores filespec information from top-level source files that are empty.
- There is a limit of 16 total text libraries that can be searched in a compilation. This is a limit in the LBR$ utility routines, and is a permanent restriction.
- There is a permanent restriction that header file searches involving DECnet file specifications may not correctly interpret an access string with password. This is because an expanded filespec is actually used to open the files, and expanded filespecs are stored without the password information for security reasons. Thus an attempt to open the file using the expanded name will generally fail. The DECnet access should be made without the need for a password (e.g. through the default account or through a proxy).
- The _LOCC builtin function does not always work correctly if an optional fourth argument is specified and if compiled /NOOPTIMIZE. In the following example, DEC C is not updating p_c after the second call to _LOCC. Thus, the program prints out `node-+` instead of `node-:`.

  ```
  #include <stdio.h>
  #pragma builtins

  int main(void) {
      char   name[]= "node::          ", *p_c;
      char **pp_c = &p_c;
  ```

```
    _LOCC ( ':', sizeof(name), &name, &p_c );
    *p_c++ = '-';
    _LOCC ( '+', sizeof(name), &name, pp_c );
    *p_c++ = '+';

    printf(name);
}
```

- DEC C does not warn about C++ style comments with /WARNINGS=ENABLE=PORTABLE.
- If your code includes assert.h multiple times and uses the text library form of inclusion, `#include assert`, the first include will work correctly. But, the second include causes the compiler to issue an error message about an invalid include file name. This is because assert have been defined as a macro within assert.h, so the compiler is looking a the macro expanded file name, which does not exist. The assert macro may be used elsewhere without any problems.

  Digital recommends that you avoid the text library form of inclusion for assert.h. Use `#include <assert.h>` instead.
- In some rare cases DEC C will enter an infinite loop trying to optimize control flow involving constant expressions. The simplest example occurs below.

  ```
  void f() {
      if (0) {}
      while (1) if (0);
  }
  ```

  Small changes to the source that do not affect the sense of the program allow the compiler to successfully compile the code. In the above example, replacing the `{}` with `;` allows the compiler to compile the code successfully. Removing the `if (0)` pieces of the code altogether also fixes the problem.
- DEC C cannot handle extremely long types when compiling with /DEBUG. The types have to be something on the order of:

  ```
  long ************************************************
       ************************************************
       ************************************************
       ************************************************
       ************************************************
       ******************************************a;
  ```

  to exhibit the problem.
- In some rare cases, DEC C does not inline functions correctly. The only known instance of incorrect inlining is for the following program:

  ```
  #include <stdio.h>

  static int called(int);
  /* Adding #pragma noinline called makes this work */

  int main (void) {
      volatile int zz = 1;
      printf("Hello World!\n");
      printf("Answer is: %d should be (12)\n",caller(zz));
    }

  int caller( int sw) {
      switch(sw) {
          case 0: return called(3);   break;
          case 1: return called(3);   break;
      }
  }
  ```

```
int called(int k) {
    struct {int table[4000];} *p=0;
    return (int)&(p->table[k]);
}
```

Instead of an answer of 12, the answer is garbage.

- DEC C produces the wrong debugging information for automatic doubles, making it impossible to examine their values with the debugger.

```
main(){
    double db = 34.439;         /* can't be examined */
    static double sb = 45.54;   /* can be examined */
}
```

- The /SHOW=TRANSLATION qualifier does not work. The information it places into the listing file is not correct.
- The default for the /PREFIX_LIBRARY_ENTRIES is ALL, even in the /STANDARD=ANSI89 and /STANDARD=RELAXED_ANSI89 modes of the compiler. If you are defining functions whose names conflict with names in the DEC C RTL, e.g. open, you will get multiply defined symbol warnings from the linker. If so, use /PREFIX_LIBRARY_ENTRIES=ANSI_C89_ ENTRIES.
- Certain programs with a loop containing a goto that goes to a label outside the loop, which in turn contains a goto back into the loop, will run incorrectly when compiled with optimization turned on. They run correctly with optimization turned off. An example of such a program is:

```
extern sub1();
extern sub2();
extern sub3();

main()
 {
  int i, j, k;
  for (i=0; i<1; i++)
    for (j=0; j<1; j++)
     {
      goto label2;
     label1:
      sub4(); sub5(); sub6();
     }
  return;

  label2:
    for (k=0; k<1; k++)
     {
      sub1(); sub2(); sub3();
     }
    goto label1;
 }
```

- The diagnostic for array declarations with more than INT_MAX elements is misleading. It should state that there are too many elements in the array.
- Listing file line numbers wrap to 0 after reaching 64k.
- If a file includes itself, the compilation eventually fails with the error that it could not open the include file because the file could not be found or had the wrong protection. It should fail with an error that it reached the maximum number of include files that it could handle.

- Extremely large, sparse switch statements cause the compiler to generate code that when linked is far larger than necessary. In the case reported, a switch statement whose range (the difference between the highest case label and the lowest case label) was about 35000 and only had 2000 case labels inside the switch was over 20 megabytes large when linked. The problem only seems to appear for switch statements whose range is in the tens of thousands and where the number of case labels is slightly more than 5% of the range of the case.

  One workaround for this problem is to add an extra case label that increases the range of the switch so that the number of case labels falls below 5% of the range (for example, add case 100000: just before the case that handles bad switch indexes). This will cause the compiler effectively to turn the switch statement into a long sequence of if-then else-if statements. The best workaround is to rewrite the switch into a series of smaller, more densely packed switch statements or to choose an alternate implementation altogether (for example, a table search that returns an index into an array of functions).

- There is a memory leak in the compiler noticed when compiling a comma-separated list of source files. If you get an error about memory limits exceeded when using comma lists, recompile each file separately.

- DEC C does not always diagnose function argument lists longer than 255 arguments. Compiling the code causes a COMPILERBUG diagnostic unless DEC C inlines the function.

- On OpenVMS VAX systems prior to V6.1, there was a problem with the relaxed_refdef external model. Under the external model relaxed_refdef, an uninitialized declaration of an external variable that left off the keyword `extern` was not associated with either a definition of the global variable under strict_refdef, or an initialized definition of the global variable under relaxed_refdef. For example, assume module A.C was compiled /EXTERN_MODEL=RELAXED_REFDEF and contained the declaration:

  ```
  /* relaxed_refdef --> tentative definition */
  int global_var;
  ```

  and module B.C was compiled /EXTERN_MODEL=STRICT_REFDEF and contained the declaration:

  ```
  /* strict_refdef --> definition */
  int global_var;
  ```

  or the module B.C was compiled /EXTERN_MODEL=RELAXED_REFDEF and contained the declaration:

  ```
  /* relaxed, strict_refdef --> definition */
  int global_var=1;
  ```

  When modules A and B were linked together on a pre-V6.1 VAX system, the linker failed to treat the declaration of `global_var` in A.C as a reference to the definition of `global_var` in B.C. Instead, references to `global_var` in A.C used the address zero. No linker messages were produced, and an executable would ACCVIO when `global_var` was referenced.

  This problem can be worked around by using one of the following techniques:

  1. Upgrade the system to OpenVMS V6.1 or higher (preferably V6.2 or higher).

2. When using /EXTERN_MODEL=RELAXED_REFDEF (this is the default), declare all external references using the keyword `extern`, and only have a single declaration as the definition that lacks the keyword `extern` or initializes the variable.
3. Use /EXTERN_MODEL=STRICT_REFDEF for all modules. This produces the most portable source code.
4. Use /EXTERN_MODEL=COMMON for all modules.