HP C/HP-UX Release Notes for HP-UX 11.01

HP 9000 Computers

HP C/HP-UX Version B.11.01.20

HEWLETT® PACKARD

5969-4407

June 2000

Legal Notice

Contents

Contents

1 New and Changed Features

This version of HP C/ANSI C for HP-UX 11.x includes support for the following new features:

- Implementation of C9X standard macros. Macros may now be defined to have a variable number of arguments.
- Predefined identifier__func__, a C9x standard, which provides assistance in debugging code.
- Expansion of the `+ESlit` compiler option to include the `+ESconstlit` and `+ESnolit` compiler options, which cause the HP C compiler to no longer store literals in read-only memory.
- `-Bextern` compiler option, which allows you to specify a separate load module containing symbols, so that you do not need to specify them from the command line.
- `-fast` compiler option, which provides a macro of compiler optimization options to ensure faster compiler speed and build times.
- Declarations within code, functionality similar to the C++ programming semantics of declaring variables anywhere in a block of code.
- WDB “fix and continue”, which allows users to modify a function in a file, and continue debugging with the modified function.
- Shared library performance improvements improvements.
- C HTML Online Help.

C9X standard macros

The C9X standard-compliant version of "variadic" or variable argument (varargs) macro notation has been added to the HP ANSI C compiler. The notation for C9X standard and for the popular GNU version of varargs is very similar.

If you have coded your macros to the GNU standards, you can expect GNU-style behavior using the HP ANSI C compiler. If you have coded your macros to C9X standards, you can expect C9X-style behavior.

IMPORTANT Differences in HP C implementations of GNU and C9X macros are noted in Chapter 4, “Problem Descriptions and Fixes,” on page 25.

Usage differences

In the HP ANSI C compiler, if there is an ellipsis (...) in the macro definition’s argument list, then the trailing arguments (including any separating comma and preprocessing tokens) are merged to form a single item: these are variable arguments. The trailing arguments (including separating commas) are combined and considered as a single argument so that the number of arguments is more than the number of parameters in the macro definition, excluding the ellipsis (...).

There are minor usage differences between how C9X and GNU specify variable argument macros are defined:

Table 1-1 How C9X and GNU define a variable argument macro

C9X	GNU
`#define foo(f, ...) printf (f, __VA_ARGS__)`	`#define foo(f, s...) printf (f, s)`

- In the GNU style, the name of the variable parameter s precedes the ellipsis in the parameter list.
- In the C9X standard, the identifier __VA_ARGS__ refers to the variable arguments.

- The identifier __VA_ARGS__ can only occur only in the replacement-list of a function-like macro that uses the ellipsis notation in the arguments.

Variable names

Variable names are also handled slightly different by C9X and GNU.

Table 1-2 How C9X and GNU refer to varargs in macro replacement text

C9X	GNU
Specified by the identifier `__VA_ARGS__`.	Name appears in the replacement text.

For example:

- `__VA_ARGS__` in the replacement text indicates a variable name in the following C9X-style code:

```
printf(f, __VA_ARGS__)
```

- s in the replacement text indicates a variable name in the following GNU-style code:

```
printf(f, s)
```

Predefined identifier__func__

The HP ANSI C compiler now defines the predefined `identifier__func__`, as specified in the C9X Standard. It has been added to HP C to provide you with additional means of debugging your code.

Syntax

The `__func__` identifier is implicitly declared by the compiler in the following manner:

```
static const char __func__[ ] = "<function-name>";
```

The above declaration is implicitly added immediately after the opening brace of a function which uses the variable `__func__`. The value <function-name> is the name of hte lexically-enclosing function. This name is the unadorned name of the function.

Example

The following code example fragment illustrates the use of predefined identifier`__func__`.

```
#include <stdio.h>
void varfunc(void)
{
     printf("%s\n", __func__);
     /* ... */
}
```

Each time the varfunc function is called, it prints to the standard output stream:

```
varfunc
```

`+ESconstlit` and `+ESnolit` compiler options

The `+ESconstlit` and `+ESnolit` compiler options have been added to HP C. These options, grouped with `+ESlit`, control HP C's storage mechanism for strings in read-only memory.

- `+ESconstlit` introduces a new default behavior, in which the HP C compiler stores constant-qualified (const) objects and literals in read-only memory.
- `+ESnolit` compiler option disables the default behavior, and causes the HP C compiler to no longer store literals in read-only memory. This restores the compiler's traditional behavior prior to this release.
- Use of the `+ESlit` option places all string literals in read-only memory

Storing const-qualified string literals in read-only memory can cause a program which violates the semantics of const to abnormally terminate with a bus error and core dump. This is because literals, which have been placed in read-only memory may not be modified at runtime. Specifying `+ESnolit` will allow you to specify that literals not be placed in read-only memory.

`-Bextern` compiler option

The `-Bextern` compiler performs the same operation as the existing `+Oextern`=*sym1,sym2,sym3*... option. Unlike `+Oextern`, however, it does not require you to manually specify extern symbols locally on the command line.

Syntax

`-Bextern`:*filename*

When used, `-Bextern` tells the compiler that the symbols reside in a separate load module. `-Bextern`:*filename* fetches the list of symbols from a text file instead of locally. Symbols referenced in the file must be white-space separated within the file.

`-fast` compiler option

The HP C/ANSI C -fast compiler option is a macro command that expands into an existing set of compiler options which, used together, result in optimum speed and build-times. This option is also provided for compatibility in porting other applications to HP-UX.

Do not use this option for programs that depend on IEEE standard floating-point denormalized numbers. Otherwise, different numericaL results may occur.

The `-fast` option expands to the following HP C/ANSI C options:

- `+O3 +nolooptransforms +Olibcalls +FPD +Oentrysched +Ofastaccess`
- `+O3` optimizations include all +O2 optimizations as well as all subprograms within a single file. It also inlines certain subprograms within the input file.
- `+Onolooptransform` disables transformation of eligible loops for improved cache performance.
- `+Olibcalls` increases the runtime performance of code which calls the standard library routines.
- `+FPD` enables a fast underflow mode on PA-7100 architectures or later where denormalized floating-point operands are flushed to zero.
- `+Oentrysched` optimizes instruction scheduling on a procedure's entry and exit sequences.
- `+Ofastaccess` optimizes code for fast access to global data items.

You can override any of the options in `-fast` by specifying a subsequent option after it.

Declarations within code

HP C has added the C9x feature which allows you to declare variables and types inside a block of statements. This also allows declaration of new variables or types, such as `expr_1`, as shown in the for statement below:

```
for(expr_1;expr_2;expr_3) statement_1
```

This new variable or type declared in expr_1 can be used in expr_2, expr_3 and statement_1.

The HP C/ANSI C compiler implementation of declarations within code is similar to, but not identical to, the C++ implementation of declarations within code See Chapter 4, “Problem Descriptions and Fixes,” on page 25 for a description of how HP C uses declarations within code differently than the HP aC++ compiler.

Example

```
int main()
{
  int i=5,j;

  j=i*i;
  printf(*"%d\n",j);

  int k=j;
  /*This is accepted in the new release of HP C*/

for(struct aa {int a;int b}
AA={10,50};AA.a<=AA.b;AA.a++){
     /*This is accepted by the new feature */
     printf("%d\n",AA.a);}
}
```

WDB “fix and continue”

HP’s WDB debugger has implemented a new “fix and continue” feature. This feature allows you to modify a function inside of a file being debugged. Once you modify the function, you can continue debugging the file, which now has a modified function. You are not required to recompile or reload the function: WDB does this in the background.

For more information on fix and continue, see the *WDB Release Notes*.

Shared library performance improvements

Beginning with HP-UX 11.0, users were not allowed to mix shared and archive libraries. This resulted in a slight increase in function overhead, particularly when routine system function calls were required. The libcres.a archive library has been added to the HP C runtime compiler to reduce this overhead.

Libcres.a is a small archive library that contains standard C routines such as string, memory, and other key functions (described in Table 1-3 below). The intent of this library is to provide performance-critical applications with the benefit of static linkage to commonly used libc functions. Libcres.a is built using a high level of optimization (+O3) with the HP C optimizing compiler.

NOTE There is no functionality impact on applications that use libcres.a.

Libcres.a contains only "leaf functions", i.e., functions that do not call any other functions. Any performance improvement is highly dependent on the application's use of these included functions.

The following symbols are contained in libcres.a:

Table 1-3 libcres.a symbols

libc symbols in libcres.a	Description
`abs()`	return integer absolute value
`bsearch()`	binary search a sorted table
`div()`	integer division and remainder
`ffs()`	find first set bit (BSD)
`insque()`	insert an element in a queue
`labs()`	return long integer absolute value

Table 1-3 **libcres.a symbols**

libc symbols in libcres.a	Description
`ldiv()`	long integer division and remainder
`memchr()`	find first occurrence of byte in memory region
`memcmp()`	compare contents of two memory regions
`memcpy()`	copy memory region to a non-overlapping region
`memmove()`	copy memory region to a possibly overlapping region
`memset()`	fill a memory region with a specified byte
`qsort()`	execute quicker sort
`strcat()`	append string number 2 to string number 1
`strchr()`	get pointer to character in string
`strcmp()`	compare two strings
`strcpy()`	copy string number 2 to string number 1
`strcspn()`	find length of matching substrings
`strlen()`	determine length of string
`strncat()`	append string number 2 to string number 1 (up to a given maximum length)
`strncmp()`	compare two strings (up to a given maximum length)
`strncpy()`	copy string number 2 to string number 1 (up to a given maximum length)
`strrchr()`	get pointer to character in string
`strspn()`	find length of matching substring
`strstr()`	finds a substring in a string
`swab()`	swap bytes

How to use libcres.a

In order to make use of libcres.a, existing makefiles must be modified to include it when linking. Existing applications must also be re-linked in order to make use of libcres.a. The modules of this library are compiled with the HP-UX C compiler using the +O3 optimization flag.

The functions in this library can not be overridden with a user-defined function of the same name, as is the case today with other libc function names.

NOTE If you use `libcres.a`, your other libraries must not contain functions of the same name. Otherwise, unexpected results may occur.

C HTML Online Help

The HP C/ANSI C compiler now includes html online help, which you can invoke using the `+help` option from the command line. See “HP C Online Help” on page 21for a description of how to access this new addition to the HP C/ANSI C compiler.

2 Installation Information

This chapter describes the contents of HP C/ANSI C Developer’s Bundle for HP-UX. This includes the following topics:

- Beginning Installation
- HP C Developer’s Bundle Contents
- Installed Compiler Paths
- Transition Links

Beginning Installation

After loading HP-UX 11.x, you can install your HP C/ANSI C Developer's Bundle. In addition to the C compiler, it contains the HP-UX Developer's Toolkit. To install your software, run the SD-UX swinstall command (see *swinstall*(1M)). It will invoke a user interface that will lead you through the installation.

NOTE After installing the HP C/ANSI C Developer's Bundle, install the latest linker patch (PHSS_16841) or its successor. This patch is required by the +objdebug option. Without this patch, the option is ignored.

For more information about installation procedures and related issues, refer to *Managing HP-UX Software with SD-UX* and other README, installation, and upgrade documentation provided or described in your HP-UX operating system package. Most of this information is also available on the web at http://docs.hp.com.

HP C Developer’s Bundle Contents

The following are the individual components of the HP C Developer’s Bundle:

- Auxiliary-Opt—Auxiliary Optimizer for HP Languages (22,465 Kb)
- C-ANSI-C—HP C/ANSI C Compiler (13,604 Kb)
- C-Analysis-Tools —C Language Analysis Tools (339 Kb)
- C-Dev-Tools—C Language Development Tools (1,382 Kb)
- DDE—Distributed Debugging Environment (27,737 Kb)
- DebugPrg—Debugging Support Tools (341 Kb)
- WDB—HP WDB Debugger (14,217 Kb)
- AudioDevKit—HP Audio Developer Kit (479 Kb)
- CDEDevKit—CDE Developer Kit (10,769 Kb)
- ImagingDevKit—HP-UX Developer’s Toolkit - Imaging (2,216 Kb)
- X11MotifDevKit—HP-UX Developer’s Tool465kit - X11, Motif, and Imake (25,629 Kb)

NOTE Be aware that, if you install all the packages, they occupy approximately 126 megabytes of disk space.

Installed Compiler Paths

Most files related to the HP C compiler are installed in the directories /opt/ansic and /opt/langtools. The installation scripts add the following paths during the installation process:

- /opt/ansic/bin and /opt/langtools/bin to the login file /etc/PATH.
- /opt/ansic/share/man/%L:/opt/ansic/share/man and /opt/langtools/share/man/%L:/opt/langtools/share/man to the login file /etc/MANPATH.

  %L is replaced by the value of the LC_MESSAGES environment variable when the man command is executed. It determines the language used for manpage searches. If LC_MESSAGES is not set, %L defaults to null. See *environ*(5).)

Transition Links

The HP C/ANSI C compiler installation package provides the capability to create and remove transition links from previous HP-UX release locations to HP-UX release 11.*x* locations. The HP C/ANSI C product installs the ISU transition link table specification files on the system.

The Software Distribution update tool tlinstall uses these files to install transition links from previous HP-UX file and directory names to the corresponding HP-UX 11.*x* file and directory names. To remove these transition links, use the update tool tlremove. For more detail, read the update tools manpages. These tools are installed in /opt/upgrade/bin.

3 Documentation Overview

The HP C/ANSI C compiler and related documentation is available for users of the HP C Developer's Bundle. This documentation is available both online, and in printed copy. Online documentation is located at http://docs.hp.com, and is viewable using your favorite web browser.

The HP C documentation consists of:

- *HP C/HP-UX Release Notes*
- *HP C/HP-UX Programmer's Guide*
- *HP C/HP-UX Reference Manual*
- *HP-UX Floating Point Guide*
- *HP C/HP-UX Online Help*

C Compiler Documentation

HP C/HP-UX Release Notes

The *HP C/HP-UX Release Notes* provides release-specific information such as new feature summaries, installation instructions, and known defects. In addition, the Release Notes contains this documentation overview to help you orient yourself regarding available documentation. The release notes are also available online in the text file /opt/ansic/newconfig/RelNotes/ansic.11.11beta.

Printed Documentation

Printed versions of Hewlett-Packard documents are available for ordering. Use the `‘man manuals’` command for details on the documents available for ordering. See also the HP documentation web site http://docs.hp.com and the HP C/HP-UX web site at http://www.hp.com/go/C. Listed below are the documents most closely related to use of the ANSI C Compiler.

- *HP C/HP-UX Reference Manua*l (B3901-90003)

  Provides reference material for HP C as implemented on HP 9000 systems. This document is based on the ANSI C standard 9899-1990, and it discusses the implementations and extensions unique to HP C on HP-UX. It does not replicate the ANSI C standard and you are referred to the standard, for any fine points not covered.

- *HP C/HP-UX Programmer's Guide* (B3901-90002)

  Contains a detailed discussion about selected C topics for HP 9000 systems. Included are discussions of data type sizes and alignment modes, comparisons between HP C and other languages, and information on 64-bit programming, optimization, threads, and parallel processing.

- *HP-UX Floating-Point Guide* (B3906-90006)

  Describes how floating-point arithmetic is implemented on HP 9000 systems and discusses how floating-point behavior affects the programmer. It also provides reference information about the C and Fortran math libraries.

HP C Online Help

The C compiler online help is a series of html files containing a combination of reference and how-to information, including the following:

- Organizing your C programs
- Compiling and running HP C programs
- Data types and declarations
- Expressions and operators
- Statements
- Preprocessing directives
- HP C optimization options
- Calling other languages
- Portable programming
- HP C parallel processing
- Porting non-HP C programs to HP-UX
- Error message descriptions

Before you begin

Before you can begin, you must set the DISPLAY environment variable to a (graphical mode) value that can accommodate the display of an HTML browser.

You can set the BROWSER environment variable to point to the location of your HTML browser. If you do not set the BROWSER environment variable, the compiler will automatically run the browser located in `/opt/ns-navgold/bin/netscape` or `/opt/ns-communicator/netscape`.

You may set the CROOTDIR environment variable to set the root directory of the online help source. If CROOTDIR is not set, the URL of the online help will be `file:/opt/ansic/html/guide/${LOCALE}/c_index.html`; this is assuming that compiler binaries are located in /opt/ansic/bin.

Accessing HP C Online Help

To access the online help, on a system where the HP C compiler is installed, enter the following:

```
/opt/ansic/bin/cc +help
```

This command will launch a web browser, displaying the index file for the HP C online help system. The actual file location of the html help is `file:/${CROOTDIR}/html/guide/${LOCALE}/c_index.html`.

If the environment variable CROOTDIR is not set, path will be formed relative to the compiler's root directory; this is usually `/opt/ansic/`. See the previous section "Before you begin" on page 21 for more information on setting the CROOTDIR environment variable.

NOTE If the browser path set by the BROWSER environment variable does not exist, or if the default browser paths `/opt/ns-navgold/bin/netscape` or `/opt/ns-communicator/netscape` do not exist, a message will be displayed telling you that the BROWSER environment variable must be set properly.

X-Motif CDE Help is obsolete

Previous versions of the HP C compiler, when installed on the X-Motif CDE environment, included a CDE version of the online help. This and the accompanying text-based 'charhelp' will no longer be updated with the ANSI C compiler. If you want to view online help, please use the HP C HTML Online Help.

Related Documentation

This documentation is available on the HP-UX 11.0 Instant Information CD-ROM and on the web site http://docs.hp.com.

- *Parallel Programming Guide for HP-UX Systems*

  Describes efficient parallel programming techniques available using HP Fortran 90, HP C, and HP aC++ on HP-UX. This document is also available online at http://docs.hp.com.

- *HP-UX 64-bit Porting and Transition Guide*

  Describes the changes you need to make to compile, link, and run programs in 64-bit mode. This document is also available online at http://docs.hp.com and in the Postscript file `/opt/ansic/newconfig/RelNotes/64bitTrans.bk.ps`.

- *HP PA-RISC Compiler Optimization Technology White Paper*

  Describes the benefits of using optimization. This white paper is available online in the PostScript file /opt/langtools/newconfig/white_papers/optimize.ps.

- *HP-UX Linker and Libraries Online User Guide*

  Replaces the manual *Programming on HP-UX*. To access the *HP Linker and Libraries Online User Guid*e, use the `ld +help` command, or visit http://docs.hp.com.

- *HP Assembler Reference Manual*

  Describes the use of the Precision Architecture RISC (PA-RISC) Assembler on HP 9000 computers. Describes PA-RISC Assembler directives, pseudo-operations, and how to run the Assembler on HP-UX.

- *HP-UX Reference Manual*

  The reference manual pages, or man pages, are available online (use the command man man for more information), and are also available on the CD-ROM. You may also access this manual online by visiting http://docs.hp.com.

- HP-UX Software Transition Kit (STK)

  Enables the application developer to easily transition software from HP-UX 10.*x* to either the 32-bit or the 64-bit version of HP-UX 11.0. The kit is available free of charge on the HP-UX 11.0 Application Release CD-ROM, or from the web at http://www.software.hp.com/STK/index.html.

- HP WDB Debugger documentation

  HP WDB is the HP-supported implementation of the GDB debugger. Refer to the README file in the directory /opt/langtools/wdb/doc for information on the documentation provided with the debugger. See also the web site http://www.hp.com/go/debuggers.

4 Problem Descriptions and Fixes

This section details known defect fixes, and workarounds for the BETA version of the HP C/ANSI C compiler.

Problems corrected in the final release of the HP C/ANSI C compiler will be referenced in the *Software Status Bulletin*.

Users with support contracts may access these bulletins and patch information from the HP SupportLine database on the World Wide Web located at one of the following URLs:

- `http://us-support.external.hp.com/`
- `http://europe-support.external.hp.com/`

Defect fixes

The following defect fixes are included with the BETA version of the HP C/ANSI C compiler:

- `+Onoinline` has been corrected so that, when specified, inlining does not occur.
- When compiled with -O +DA1.0 +DS1.1, the compiler occasionally terminated. This has been corrected.
- swverify is properly running for all directories installed with the ANSI C Developer's Bundle.
- The compiler has been corrected so that it no longer terminates when compiling a program with two or more functions at +O.
- Inlining previously dropped the register storage class of a variable. This has been corrected.
- A previous release of the compiler filesets did not specify the attributes `architecture`, `machine_type`, `os_name`, `os_release`, and `os_version`. This has been corrected.
- HP C implements enumeration types and bit fields as signed by default, as specified by the ANSI C standard. As of HP-UX 11.0, HP C/ANSI C in 64-bit mode only incorrectly implemented enumeration types and bit fields as unsigned by default. This defect has been fixed in version A.11.01.00 of HP C/ANSI C. That is, in 64-bit mode, HP C/ANSI C now correctly implements enumeration types and bit fields as signed by default.

Workarounds

The following are workaround solutions to previous problems with the HP C/ANSI C compiler:

- In the 64-bit environment, using the `+ESlit` option with the +DA2.OW option results in an error message. The combination of these options tells the code generator to locate the const data in the readonly code segment so that it becomes an error in the 64-bit environment. In the 32-bit environment, however, this error occurs silently. The linker will only detect and display error information for this inconsistency on 64-bit addressing.
- `+Onomoveflops` should always be used with the `+FPZ` and `+FPI` floating point options. `+Onomoveflops` prevents floating point instructions from being moved, and and replaces integer division by floating point multiply by the inverse.
- If you intend to use GNU style variable argument macros in HP C, note that you can make the concatenation operator '##' prevent syntax errors from occurring when the variable argument comes in as empty (the null string). However, you can also insert whitespace to the left of the left operand of '##' to more accurately specify the intended left operand.

  For example, if you use

  ```
  #define foo(f, s...) printf(f, s)
  ```

  Then the macro "call"

  ```
  foo("Hello world.\n");
  ```

  results in the expansion

  ```
  printf("Hello world.\n",);
  ```

  (note the comma ",") causing a syntax error.

  GNU provides the following workaround for this kind of a situation. If you use:

  ```
  #define foo(f, s...) printf(f, ## s)
  ```

If the variable parameter 's' is non-null, if for example, you use:

```
foo("%s %d\n", "Cycles", "1024");
```

the result is

```
printf("%s %d\n", "Cycles", "1024");
```

as the expansion as you would expect.

However, if 's' is null, this erases the comma to the left of the '##' in the macro definition and resulting expansion is:

```
printf("Hello world.\n");
```

Note that the comma is gone.

In order to get the same behavior in HP C, you must insert a space to the left of the comma to make it clear to the preprocessor that the comma is the left operand of the '##' operator. Thus your definition for the macro 'foo' is:

```
#define foo(f, s...) printf(f , ## s)
```

(Note the space to the left of the '##' operator in the macro definition.)

If the space is not inserted, the left operand of the '##' operator is understood to be:

```
printf(f,
```

Because there is no parameter by that name for 'foo', it is erased.

- When specifying declarations within code in the HP C/ANSI C compiler, *do not* expect the same behavior in HP aC++. For the example:

  ```
  for(int i = 0; i < j; i ++) int i;
  ```

  Note the lack of a new block opening for the "for" statement. The C++ compiler accepts this form, with warnings, but the C compiler does not. The difference in the way the stack is handled causes the difference in behavior.

  Previously, the C compiler did not emit the source file information for the global typedefs. To correct this, use `-y` option along with `-g` when debug info is generated. You can generate debug information by compiling with `+Oobjdebug`.

- The `+Olibcalls` transformation in the HP C compiler has been changed so that the following information in the *HP C/HP-UX Programmer's Guide* is no longer valid:

  “Calls to `setjmp()` and `longjmp()` may be replaced by their equivalents `_setjmp()` and `_longjmp()`, which do not manipulate the process's signal mask.”

  Note that all other tranformations of `+Olibcalls` are unaffected by this change.