# HP C++ Version A.11.01 Release Notes

## HP 9000 Computers


**5965-4444**
**May 1997**

Printed in: U.S.A.

# Notice



UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

WHILE THE INFORMATION IN THIS PUBLICATION IS BELIEVED TO BE ACCURATE, HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material. Information in this publication is subject to change without notice.

**RESTRICTED RIGHTS LEGEND**

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in sub-paragraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
3000 Hanover Street
Palo Alto, CA 94304 U.S.A.

Rights for non-DoD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19 (c)(1,2).

# Contents


**1. New and Changed Features**

Overview of This Release of HP C++ . . . . . .7
New and Changed Features. . . . . .7

**2. Installation Information**

Migrating to the UNIX System V Release 4 (V.4) File System . . . . . .9

**3. Related Documentation**

Online Documentation . . . . . .11

Online Example Source Files . . . . . .13

Printed Documentation . . . . . .13

Other Printed Documentation . . . . . .14

Information on Exception Handling Features . . . . . .15
Detecting Link Incompatibilities when Using Exception Handling . .15
Detecting Link Incompatibilities in Shared Libraries . . . . . .16
Exception Handling Language Clarifications. . . . . .17

**4. Problem Descriptions and Fixes and Known Limitations**

Known Problems . . . . . .23

Known Limitations . . . . . .23
The setjmp/longjmp and +eh option . . . . . .24
Kernel threads unsupported . . . . . .24

# Contents

# Preface

This document provides the following information:

- new and changed features
- installation information
- related documentation
- problem descriptions and fixes

**NOTE** The software code printed in the release notes title indicates the software product version at the time of release. Some product and operating system changes do not require changes to documentation; therefore, do not expect a one-to-one correspondence between these changes and release notes updates.

Latest printing: May 1997

This document resides online in the file `/opt/CC/newconfig/RelNotes/CXX.release.notes`. You can print the online copy by using an `lp` command like the following:

`lp –d`*`printer_name`* `/opt/CC/newconfig/RelNotes/CXX.release.notes`

# 1 New and Changed Features

This chapter summarizes the new and changed features included in the A.11.01 HP C++ release.

## Overview of This Release of HP C++

New and changed features in this HP C++ release include compiler enhancements and problem fixes, and revised online help documentation.

### New and Changed Features

The new and changed features for A.11.00 are listed below. These items are fully documented in the *HP C++ Online Programmer's Guide* (see Chapter 3 for access instructions.)

- New Compiler Options `+ESlit` and `+ESsfc`.
- Compiler option +a1 changed to be default behavior. This option causes translator mode to produce ANSI C style declarations.
- `+p` option identifies more source code constructs that may be issues when migrating to HP aC++, which supports ANSI C++ syntax.
- The `+DA` option changed so that if you specify `+DA` and not `+DS`, the default instruction scheduling is based the `+DA` option, not the type of system on which you are compiling. Refer to the online help for details about these two options.
- Any programs that use `+eh` and also use Setjmp/Longjmp must change the `#include` from `<setjmp.h>` to `<Setjmp.h>`.
- Header files for the HP Codelibs library moved from `/opt/CC/include/codelibs/` to `/usr/contrib/include/codelibs`. You may see errors that the compiler cannot find certain files.

  The instructions for building the HP Codelibs library are in `/usr/contrib/codelibs/README`.
- For task library users, libV3 is no longer required.

- Changes to the following in HP UX 10.30 may cause incompatibilities with programs created with previous versions of HP C++:
  - The underlying type corresponding to the `typedef size_t` changes from unsigned int to unsigned long. Similarly, `ptrdiff_t` changes from int to long.

    These changes cause compatibility problems when `size_t` is used in a non-extern C interface because the mangled signature is different.

    Because of these changes, if any object files are recompiled or linked, then you must recompile *all* C++ files. This means that third-party libraries in archive form may also need to be recompiled or updated.
  - In HP UX 10.30, `time_t` changes to type long. This change may cause source files that compile without error using HP C++ for HP-UX 10.10 or 10.20 might not compile with the 10.30 release. The example below shows one example of what may occur.

    ```
    1: #include <time.h>
    2: time_t ff (time_t t) { return t; }
    3: time_t ff (long t) { return t; }
    4: int main () { long tt = ff (1L); return 0; }
    ```

    In the example, ff is overloaded to take either a `time_t` or a `long` parameter. On a 10.10 or 10.20 system, where `time_t` is an int, the code compiles. On a 10.30 system, however, where `time_t` is a long, the code fails to compile:

    ```
    CC: “tm.c”, line 4: error: two definitions of ff() (1034)
    ```

# 2 Installation Information

Read this entire document and any other release notes or readme files you may have before you begin an installation.

To install your software, run the SD-UX **`swinstall`** command. It will invoke a user interface that will lead you through the installation. 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 10.x operating system package.

HP C++ requires approximately 46 MB of disk space: 16 MB for the files in `/opt/CC` and 30 MB for DDE, Blink Link, and HP/PAK. Depending on your environment, you may also need documentation for other parts of your system, such as networking, system security, and windowing.

**NOTE**

During the installation, the WARNING and ERROR messages shown below may appear in the files `/var/adm/sw/swmodify.log` and `/var/adm/sw/swagent.log`.

You should ignore these messages, they are not valid.

```
WARNING: Cannot delete the definition for
         "//opt/langtools/lbin/ucomp.tmp" from the fileset
         "Auxiliary-Opt.LANG-AUX".  The file does not exist in
          this fileset.

ERROR:  The selected software was not modified.  All of the
        specified file modifications are invalid.  See the ERROR
         and/or WARNING messages above.
```

## Migrating to the UNIX System V Release 4 (V.4) File System

Two migration tools are provided for users. Either the system Upgrade Tools or the tlink_install script can be used to migrate from an HP-UX 9.x system to an HP-UX 10.x system.

If your system has Upgrade Tools installed (`/usr/sbin/upgrade` exists), transition links are created automatically upon product installation. A method of removing these links is also provided. For more information on automatic transition links refer to your operating system upgrade documentation.

If there are no Upgrade Tools on your system, you can use the C++ `tlink_install` script as a migration aid to create symbolic links for HP C++ product executables and include files when migrating from HP-UX 9.x locations to HP-UX 10.x locations. The script is located in `/opt/CC/newconfig/tlink_install`. Should you want to remove these links, use the script located in `/opt/CC/newconfig/unlink_install`. These scripts must be executed by a super user.

Note that to reverse your migration process, you must use the appropriate uninstall tool. That is, if links were installed using the system Upgrade Tools, they must be uninstalled using the system Upgrade Tools. If links were installed using the `tlink_install` script, they must be uninstalled with the `unlink_install` script.

# 3 Related Documentation

Documentation for HP C++ is described in the following sections.

## Online Documentation

Xwindow users can invoke the *HP C++ Online Programmer's Guide* in any of the following ways:

- Use the `+help` option on the **`CC`** command line.
- Click the ``?'' icon on the HP CDE front panel. Then select the HP C++ icon.
- Execute the **`dthelpview`** command in located in `/usr/dt/bin`:

  ```
  /usr/dt/bin/dthelpview -h /opt/CC/dt/appconfig/help/C/CXX.sdl
  ```

  If your HP compiler is installed on another system or your system is not running HP CDE, this command may be useful.

The following online documentation is included with the HP C++ product:

- *HP-DDE Debugger Online Help*

  Refer to the discussion on basic-style (not advanced-style) debugging of optimized code in the HP/DDE debugger online help.

- To access the *HP C++ Online Programmer's Guide*, use the command:

  ```
  CC +help
  ```

  To access the *HP Linker and Libraries Online User Guide* use the command:

  ```
  ld +help
  ```

  The *HP Linker and Libraries Online User Guide* online guide replaces the manual *Programming on HP-UX*.

**NOTE** Users with character-based terminals or terminal emulators can use the `charhelp` program to view or print the online help provided for C++ and the linker.

To start charhelp enter the full pathname (or just charhelp if /opt/langtools/bin is in your $PATH environment variable), and you will get a usage statement:

```
$ /opt/langtools/bin/charhelp
charhelp: Usage:  charhelp {cc | CC | aCC | f77 | ld | -helpVolume file}
```

For help with C++, for example, enter charhelp CC and follow the menus for further direction. For more information, see the man page for *charhelp*(1) (/opt/langtools/share/man/man1.Z must be in your $MANPATH environment variable).

---

If the +help option does not work, ensure the environment variable DTHELPSEARCHPATH is set. (It may not be set if you **rlogin** to a system, for example.) If it is not set, use the following command to set it:

```
eval $(dtsearchpath)
```

Ensure the LANG environment variable is set, typically LANG=C.

As a workaround, you can view the linker online help using the ? icon on the HP CDE front panel or by using one of the following commands:

```
/usr/dt/bin/dthelpview -helpVolume linker
```

or

```
/usr/dt/bin/dthelpview -h /opt/langtools/lib/linker/dt/appconfig/help/C/linker.sdl
```

- *HP C++ Templates Technical Addendum* describes template implementation in HP C++. You can access the addendum from within the *HP C++ Online Programmer's Guide*. It is also available in the postscript file, /opt/CC/newconfig/TecDocs/templates.ps and in the ASCII file, /opt/CC/newconfig/TecDocs/templates.ascii.
- *HP C++ Troubleshooting Notes* focuses on methods of diagnosing and solving problems you may encounter. It contains a “troubleshooting matrix” and a list of tools available online in the /opt/CC/contrib/Tools directory.

  The document is available online in the postscript file, /opt/CC/newconfig/TecDocs/tools.ps, and in the ASCII file, /opt/CC/newconfig/TecDocs/tools.ascii. You can access the ASCII file from within the *HP C++ Online Programmer's Guide*.
- *HP C++ Release Notes* is this document. The online ASCII file can be found in /opt/CC/newconfig/RelNotes/CXX.release.notes.

- The *HP PA-RISC Compiler Optimization Technology White Paper* describes the benefits of using optimization. It is available in the postscript file, `/opt/langtools/newconfig/white_papers/optimize.ps`.
- Online manual pages for **`CC`**, **`c++filt`**, **`nm++`**, **`gprof++`**, and the standard libraries (stream, task, complex, codelibs, and standard components) are provided under `/opt/CC/share/man`.

## Online Example Source Files

The HP C++product comes with the source files of examples from the *HP C++ Programmer's Guide*. The example source files reside in the `/opt/CC/contrib/Examples` directory.

## Printed Documentation

- *HP C++ Release Notes* is this document. Release notes are also provided online, as noted above.
- *HP C++ Programmer's Guide* (92501-90005) contains similar, but in some cases less current, information to that of the *HP C++ Online Programmer's Guide*.
- *Quick Reference Card* (B1637-90001)
- *HP/DDE Debugger User's Guide* contains information on debugging programs with the HP Distributed Debugging Environment on the HP 9000. (B3476-90015)

To order printed versions of Hewlett-Packard documents, refer to *manuals*(5).

# Other Printed Documentation

Some of the many available C++ publications are listed here:

- *Codelibs Library Reference* (B2617-90000) complete information on the HP Codelibs class library. This book can be ordered by contacting your local HP sales office or Hewlett-Packard's Support Materials Organization (SMO) at 1-800-227-8164 and providing the above part number. Also see the codelibs(3X) man page. (If you see the message `"Man page could not be formatted"` or `"No manual entry for codelibs"` ensure that the man page is installed and your `MANPATH` variable includes `/opt/CC/share/man`.)
- *The C++ Programming Language*, second edition, by Bjarne Stroustrup (ISBN 0-201-53992-6) is a tutorial on C++ including a complete language reference manual and information about object-oriented design and software development. This book is available at technical bookstores.
- *C++ Primer*, second edition, by Stanley Lippman (ISBN 0-201-54848-8) provides a complete tutorial introduction to C++. This book is available at technical bookstores.
- *The Annotated C++ Reference Manual*, by Margaret Ellis and Bjarne Stroustrup (ISBN 0-201-51459-1) is a complete C++ language reference manual plus annotations and commentary that describe in detail why features are defined as they are. This book is available at technical bookstores.
- The *HP PA-RISC Compiler Optimization Technology White Paper* (5963-7250E) describes the benefits of using optimization. To order a printed copy, contact your local HP sales office or HP DIRECT at 1-800-637-7740. The white paper is also provided online as noted above.
- USL/Novell manuals contain valuable information about C++, some of which is specific to the cfront compiler upon which HP C++ is based.

  To inquire about the latest versions of these manuals, you can contact the following:

  - U.S. customers--phone 1-800-336-5989

- International customers--FAX 1-801-431-4045
- You can also use http://www.stream.com/ to get more information about available documentation.

You need to request the Basic SDK Documentation Set, which includes:

- *Software Development Tools*
- *Programming in Standard C and C++*
- *Debugging and Analyzing C and C++ Programs*
- *Programming with System Calls and Libraries*
- *Porting and Integration Guide*
- *C++ Standard Components Programmer's Guide*
- *Programming with the C++ Standard Libraries*

# Information on Exception Handling Features

Below is some valuable information on exception handling features published in previous release notes.

Exception handling is supported in both compiler mode and translator mode, and such object files can be intermixed. Use the +eh option to enable exception handling for both compiling and linking. There is some performance degradation when using the +eh option in translator mode.

## Detecting Link Incompatibilities when Using Exception Handling

This release of HP C++ supports exception handling when the +eh option is specified. Note that code compiled with +eh is not link compatible with code that has not been compiled with +eh. There are three reasons for this:

1. When +eh is enabled, constructors no longer allocate memory for heap objects; such memory is allocated before the constructor is called. For example, if non +eh code calls a +eh constructor to construct a heap object, memory for the heap object is not allocated.
2. When +eh is enabled, all constructors perform a certain amount of bookkeeping to indicate how far object construction has progressed; this is needed because in the event of an exception, partially constructed objects need to be cleaned up. If +eh code calls a non +eh constructor, this bookkeeping does not take place; thus, in the event of an exception, there is incorrect information about the state of objects in procedures which called non +eh constructors.
3. All +eh procedures perform a certain amount of bookkeeping to save information about the list of objects constructed within each procedure. Since non +eh procedures do not perform this bookkeeping, such procedures do not undergo any object cleanups in the event of an exception.

## Detecting Link Incompatibilities in Shared Libraries

When the CC driver is used to produce a shared library (using –b), link incompatibilities are detected by **`c++patch`** using the same rules described above. When performing a link which involves shared libraries, HP C++ waits until run time to establish that each shared library linked in or explicitly loaded is compatible with the main executable. If any incompatibilities are detected, the default behavior is to print a warning message to stderr. If this default behavior is unacceptable, you can override it by linking in your own version of the routine `__link_incompatibility`.

For example, if you do not wish to have any warning of this kind at all, the following routine can be linked in:

```
extern "C" void __link_incompatibility
     (const char* libname, int lib_mode) {
     //libname is the name of the library
     //lib_mode == 0 for a non +eh library
     //lib_mode == 1 for a +eh library

     //You can provide your own version to override the
     //default behavior
     //This is an empty body which does nothing
}
```

# Exception Handling Language Clarifications

This section lists various exception handling language issues which should be considered clarifications of *The Annotated C++ Reference Manual*. These clarifications represent the behavior of HP's implementation of exception handling.

Issues in this section are organized as follows:

- Throwing an Exception
- Handling an Exception
- Throw Specifications
- `terminate()` and `unexpected()`
- Other Issues

## Throwing an Exception

1. **Can a class with an ambiguous base class be thrown? That is, should the following be legal?**

   ```
   struct A { ... };
   struct B1 : A { ... };
   struct B2 : A { ... };
   struct C : B1, B2 { ... };
   void f()
   {
   C c;
   throw c;  // legal?
   }
   ```

   No, throwing a class with an ambiguous base class is not legal.

2. **Can a class with multiple instances of the same base class be thrown if only one of the base class instances is accessible?**

   No, a class with multiple instances of the same base class cannot be thrown even if only one of the base class instances is accessible.

3. **What happens when a reference is thrown?**

   A temporary is allocated, the object referenced by the throw argument is copied into the temp, and the search for the appropriate handler is begun.

   When the handler is found, if its argument is not a reference type, the local is initialized from the temp. If the handler's local variable is of a reference type, the reference is made to refer to the temp.

The possibly surprising effect of this is that if a reference to a global is thrown, and the handler's local is a reference type, the handler gets a reference to the temporary, not a reference to the global.

4. **Can the name of an overloaded function be thrown?**

   No, the name of an overloaded function (really, its address) cannot be thrown.

5. **What is the precedence of throw?**

   A throw-expression is an assignment-expression.

6. **Can a throw appear in a conditional expression? For example, is the following legal?**

```
void f()
{
int x;
x ? throw : 12;
}

void g()
{
int x;
x ? 12 : throw;
}
```

   Yes, a throw can appear in a conditional expression.

7. **Are nested throws allowed?**

   Yes. When a nested throw occurs, processing of the previous exception is abandoned and the new exception is processed.

8. **What happens if a rethrow occurs outside the dynamic context of a handler?**

   The behavior of a rethrow outside the dynamic context of a handler is undefined.

9. **What happens if an exception is thrown in a signal handler?**

   Throwing an exception in a signal handler is not supported. There is no way to predict when a signal handler will execute, consequently the signal handler could be called when the exception handling structures are in an inconsistent state.

10. **What happens if a longjmp is issued in a signal handler?**

This is not recommended for the same reason that throwing an exception in a signal handler is not supported. The signal handler interrupts processing of the code resulting in undefined data structures with unpredictable results.

## Handling an Exception

1. **Should the implementation warn or generate a hard error for the appearance of a masked catch clause?**

   The appearance of a masked catch clause is an error.

2. **Does the presence of a linkage specification affect the handlers that can catch (the address of) a function?**

   No, the type of a function is not affected by a linkage specification.

   For example, this throw:

   ```
   extern "C" {
   void f(int);
   };

   void g()
   {
   throw f;
   }
   ```

   is catchable by:

   ```
   catch (void (*)())
   ```

3. **Can an incomplete type appear in a catch clause?**

   No, an incomplete type cannot appear in a catch clause.

4. **When is an exception considered handled?**

   An exception is considered handled when one of the following occurs:

   - a handler for the exception is invoked
   - terminate is invoked
   - unexpected is invoked

## Throw Specifications

1. **Must all throw specifications on the definition and declarations for a given function agree?**

   Yes, all throw specifications on the definition and declarations for a given function must agree.

2. **Can a class with ambiguous base classes be on a specification list? That is, is the following throw specification on bar legal?**

```
struct A { ... };
struct B1 : A { ... };
struct B2 : A { ... };
struct C : B1, B2 { ... };

void foo (C* cp)
{
w *cp;   //error according to ANSI
}

void bar () throw(C);  // legal?
```

   No, a class with an ambiguous base class cannot appear in a throw specification.

3. **Can a derived class of a class on a throw specification list also appear in that same throw specification list?**

   Yes, a derived class of a class on a throw specification list can also appear in that same throw specification list.

4. **Can a function that lists a pointer to a base class in its throw specification list also throw a pointer to a derived class of that class?**

   Yes, a function that lists a pointer to a base class in its throw specification list can throw a pointer to a derived class of that class.

5. **Can a reference appear in a throw specification list?**

   Yes, a reference can appear in a throw specification list.

6. **Can a type appear more than once in a throw specification list?**

   **That is, is the following declaration legal?**

```
void baz() throw(A,A,A);     // legal?
```

   Yes, duplicate types are allowed in throw specification type lists.

7. **Can an incomplete type appear in a throw specification list? For example, should the following be legal?**

```
struct A;
void f() throw(A) { }
```

   Yes, an incomplete type can appear in a throw specification list.

8. **Where can a throw specification appear?**

A throw specification can appear only in a function declaration or a function definition and only for the function being declared or defined. In particular, it can not appear within an argument list nor in a typedef.

## terminate() and unexpected()

1. **What should be done when a thrown exception is not handled?**

   No cleanups should take place; terminate should be called.

   If an unhandled exception occurs while constructing static objects, call terminate. If terminate then calls exit, any fully constructed or partially constructed statics should be destroyed.

   If an unhandled exception occurs while destroying static objects, call terminate. If terminate then calls exit, try to destroy any remaining static objects. Do not try again to destroy the object that caused the exception.

2. **Can terminate() call exit()?**

   Yes, terminate() can call exit().

3. **Can unexpected() return?**

   No, unexpected() cannot return.

4. **Can unexpected() throw or rethrow?**

   Yes, unexpected() can throw or rethrow.

5. **What does unexpected() rethrow?**

   A rethrow in unexpected() rethrows the exception that caused unexpected() to be called.

## Other Issues

1. **Are transfers of control into try blocks and handlers legal?**

   No, transfers of control into try blocks and handlers are not legal.

2. **Is it correct to consider an object constructed when its last statement is reached, while a destructor is considered complete just before its first statement is reached?**

An object is not considered fully constructed until everything in the constructor is finished. An object is considered partially destroyed before anything happens in the destructor.

3. **Should the EH run-time delete memory allocated by a new-with-placement?**

   No, the EH run-time should not delete memory allocated by a new-with-placement.

4. **Should locals and globals be cleaned up when an unhandleable exception is thrown?**

   No, locals and globals are not to be cleaned up when an unhandleable exception is thrown.

5. **Should an object for which a destructor has been called still be cleaned up by the EH run-time?**

   A destructor should not be called explicitly on an object for which a destructor is called implicitly. Thus the EH run-time should not have to worry about whether an explicit destructor call has been issued for an object.

6. **Should exit() throw a standard exception to ensure that automatics are cleaned up?**

   No, exit() should not throw an exception.

7. **What should happen when an exception is thrown from a function registered with atexit()?**

   When an exception is thrown from a function registered with atexit(), terminate() should be called.

8. **What should happen if the user program calls alloca()?**

   You can only use alloca() in translator mode. However, it is recommended that you avoid this function.

# 4 Problem Descriptions and Fixes and Known Limitations

This chapter summarizes the known problems and limitations of the current version of HP C++, except as otherwise noted.

HP-UX 10.10 is the last supported OS for PA-RISC 1.0 architecture machines. HP-UX 10.30 no longer supports execution of PA-RISC 1.0 code, and 10.30 compilers no longer support the compilation of PA-RISC 1.0 code.

For system level binary compatibility information, see the *Release Notes for HP-UX 10.30*.

## Known Problems

Customers on support can use the product number to assist them in finding SSB and SRB reports for HP C++. The product number you can search for is HPCPLUSPLUSA.

To verify the product number and version for your HP C++ compiler, execute the following HP-UX commands:

```
what /opt/CC/lbin/cfront
```

```
what /opt/CC/bin/CC
```

## Known Limitations

Some of these limitations with possible workarounds are discussed in detail elsewhere in this document. Please be aware that some of these limitations are platform-specific.

## The setjmp/longjmp and +eh option

Code compiled in compiler mode with the +eh option should not use setjmp/longjmp. To use setjump/longjmp with +eh in translator mode, replace all setjmp/longjmp calls with Setjmp/Longjmp. You must also must change the `#include` from `<setjmp.h>` to `<Setjmp.h>`.

## Kernel threads unsupported

The thread-safe level of the code generated by HP C++ depends on which libC routines are called with the possible exception of static constructors for function scope statics or +eh code.

Thread-safe levels depends on which particular interface and the type of threads.

**Table 4-1 Thread-safe Levels**

| | Kernel Threads | User Threads |
|---|---|---|
| **Generated Code:** | | |
| Function-scope statics | Thread-Restricted C. User owns local variable. | Thread-Restricted C. User owns the local variable. |
| File-scope static and globals | Thread-Restricted C for dynamic loading of shared libraries. Ordering of initialization may be more of a problem. | Thread-Restricted C for dynamic loading of shared libraries. Ordering of initialization may be more of a problem. |
| `+eh` code | Thread-Restricted A. Thread-Unusable unless other threads are just C. | Thread-Safe Performance Constrained. |
| **`libC` interfaces:** | | |
| +eh ([re]throw) | Thread-Restricted A. Thread-Restricted B if *only one* thread is written in C++. | Thread-Safe Performance Constrained. |
| I/O (`iostreams`, `strstream`, etc.) | Thread-Restricted C. | Thread-Safe Performance Constrained. Tuned if using predefined streams: **`cin`**, **`cout`**, **`cerr`**, **`clog`**. Otherwise Thread-Restricted C. |

| | Kernel Threads | User Threads |
|---|---|---|
| vec `new/delete` | Thread-Restricted C. | Thread-Safe<br>Performance Constrained. |
| `cxxshl_load` and `cxxshl_unload` | Thread-Restricted C. | Thread-Restricted C. |
| Others | Probably references no statics/ globals so completely safe. `libC` is not fork-safe. It assumes no cancellations are possible. | Probably references no statics/ globals so completely safe. `libC` is not fork-safe. It assumes no cancellations are possible. |