LibSuite++®

THE PLUM HALL VALIDATION SUITE FOR THE STANDARD C++ LIBRARY

VERSION 2025a August 2025

Your Feedback is Valued

Please feel free to contact me with any issues, errors, omissions, thoughts, … concerning the test cases and infrastructure in the Plum Hall test suites. The software is constantly updated with new test cases and infrastructure improvements. A new distribution is released in the month of August every year. Please contact me by email: dougteeple at plumhall2b.com.

New in lvs25a:

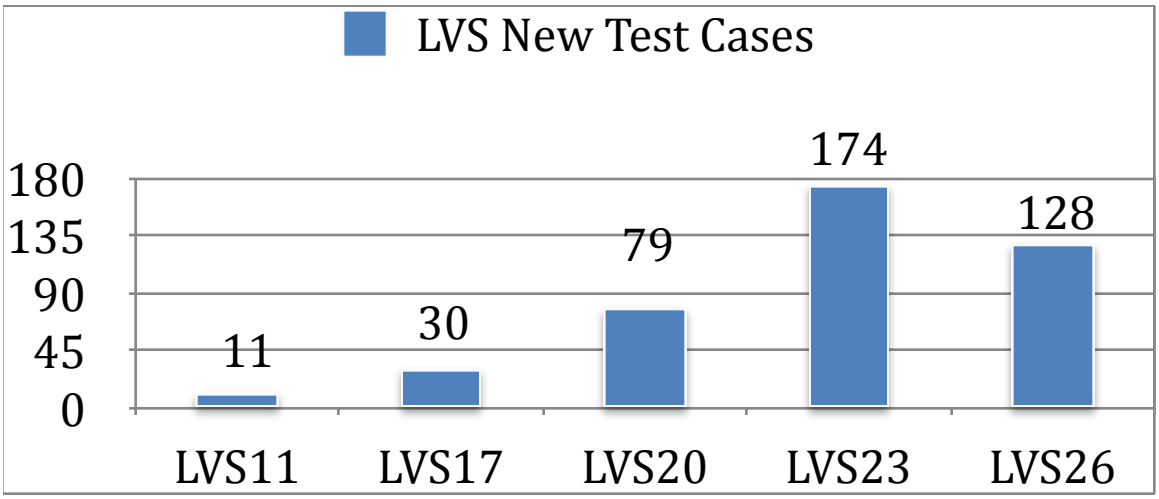
This release moves forward with new test cases for C++26.

Version	ISO Document	__STDC_VERSION__	Comments
CXX11	ISO/IEC 14882:2011	201103L	C++0x
CXX14	ISO/IEC 14882:2014	201402L	C++1y
CXX17	ISO/IEC 14882:2017	201703L	C++1z
CXX20	ISO/IEC 14882:2020	202002L	C++2a
CXX23	ISO/IEC 14882:2023	202311L	C++2b
CXX26			Work In Progress

C++ Releases

There are 128 new LVS test cases, documented in "coverage-c26.html", in multiple directories. These new test cases predominantly pertain to the proposed C++26 standard. The total number of test cases is now more than 6800, including positive, negative and undefined cases.

The file compiler-flags.h contains defines for common compilers. Modify these settings if your compiler is implemented in the list, our add custom settings for your compiler. Also review envsuite in detail. Some suggested settings are included for common compilers. Use these settings to create the appropriate build environment for the version of the compiler that you wish to test.



Running the Test Suite

It is very important that you review envsuite(.bat), flags.h and compiler-flags.h to choose the correct settings for your compiler.

envsuite(.bat) is a script which is basically a large case statement. The cases are settings for different compilers. Common compilers are available in the script. If your compiler is not represented, you can use the existing implementations as a guide. envsuite is called in such a way that it instantiates environment variables used by the build script to run the test cases.

flags.h is a header file included by each test case. It defines flags which determine the standards year to test against and features that should be tested for the corresponding standards year.

compiler-flags.h is another header included by each test case. It defines specific flags for each compiler. If your compiler is not represented, add it using existing cases as a guide. The flags are very restricting as shipped. The reason is that is the only way for the tests actually to return any results, particularly for newer standards years, for which few, if any of the features tested actually compile. So, after an initial run to get basic results, you may see that many test cases are not as skipped. Over time, remove the restraint flags to test newer standards features.

compiler-setup.bat is a Windows-only script that should be run after envsuite.bat to set up specific compiler environment variables. Modify as required (but don’t forget to run this script after envsuite.bat).

VERSION 2024a August 2024

Your Feedback is Valued

Please feel free to contact me with any issues, errors, omissions, thoughts, … concerning the test cases and infrastructure in the Plum Hall test suites. The software is constantly updated with new test cases and infrastructure improvements. A new distribution is released in the month of August every year. Please contact me by email: dougteeple at plumhall2b.com.

New in lvs24a:

This release moves forward with new test cases for C++26.

Version	ISO Document	__STDC_VERSION__	Comments
CXX11	ISO/IEC 14882:2011	201103L	C++0x
CXX14	ISO/IEC 14882:2014	201402L	C++1y
CXX17	ISO/IEC 14882:2017	201703L	C++1z
CXX20	ISO/IEC 14882:2020	202002L	C++2a
CXX23	ISO/IEC 14882:2023	202311L	C++2b
CXX26			Work In Progress

C++ Releases

C++ tests may now be checked as conforming to *16.4.2.5 Freestanding* as opposed to hosted implementations. as well as C++ freestanding tests as denoted in the document ISO/IEC DIS 14882:2023.

There are 113 new LVS test cases, documented in "newcases-lvs23a-lvs24a.txt", in multiple directories. These new test cases predominantly pertain to the C++23 and proposed C++26 standards. This release also contains initial support for testing freestanding C++ in t161.dir/. Directory t01a.dir/ contains some tests for modules, though more detailed testing is found in xvs24a.

C++ also now adds tests for undefined behavior, which were previously only available for the C language. The tests are in conform/undeftests/u*.in. Software, especially FREESTANDING, should not contain any constructs which have undefined behavior as per the standards. ISO 26262 in particular has a focus on dealing with undefined/unspecified behavior of C/C++ and on preventing runtime errors. Obeying language standards is recommended by all current safety standards.

The file compiler-flags.h contains defines for common compilers. Modify these settings if your compiler is implemented in the list, our add custom settings for your compiler. Also review envsuite in detail. Some suggested settings are included for common compilers. Use these settings to create the appropriate build environment for the version of the compiler that you wish to test.

It is very important that you review envsuite(.bat), flags.h and compiler-flags.h to choose the correct settings for your compiler.

Release 24a New Test Cases

This release also adds new test cases addressing:

- Hashing support for std::chrono value classes
- std::is_within_lifetime
- Native handles in file streams
- Interfacing string streams with std::string_view
- Interfacing std::bitset with std::string_view
- More constexpr for <cmath> and <complex>
- Adding the new 2022 SI prefixes on ratios: std::quecto, std::ronto, std::ronna, and std::quetta
- std::copyable_function
- std::submdspan()
- <debugging>: Debugging Support
- <linalg>: A free function linear algebra interface based on the BLAS
- Added tuple protocol to std::complex
- views::concat
- Concatenation of strings and string views
- std::ranges::generate_random
- Printing Blank Lines with std::println()
- std::formatter<std::filesystem::path>

Filename	Test Case
t170.dir/_17_15r1b.cpp	21.3.11 Constant evaluation context - is_constant_evaluated, is_within_lifetime…
t170.dir/_8_4_4a.cpp	8.4.4 Transaction-safe function - implements TS_19841 - CXX26 ...
t182.dir/_183a01a.cpp	std::int_least128_t - implements P3140R0 - C++26 ..
t182.dir/_183a01c.cpp	std::int_least128_t - implements P3140R0 - C++26 ...
t182.dir/_183a01d.cpp	std::atomic_int_least128_t - atomic128_t - implements P3140R0 - C++26 ...
t200.dir/_221422a.cpp	22.14.2.2 Standard format specifiers. - implements P2510R3 "Formatting ...
t200.dir/_221422b.cpp	22.14.2.2 Fix formatting of code units as integers. ...
t200.dir/_221422c.cpp	22.14.6 Formatter - Type-checking format args -- skipped ...
t200.dir/_22144a.cpp	22.14.4 Runtime format strings II. - implements P2918R2 ...
t201.dir/_203a1b_s.cpp	21.4.2 Header <ratio> synopsis - implements P2734R0 "Adding ...
t202.dir/_20_14_1742b.cpp	20.14.17.4.2 Class template move_only_function: Partially Mutable Lambda Captures ...
t202.dir/_20_2_5a.cpp	Utility to check if a pointer is in ...
t202.dir/_22101746a.cpp	22.10.17.4.6 [func.wrap.ref] - implements P0792R14 "function_ref: a type-erased ...
t202.dir/_22102a.cpp	22.10.2 [functional.syn] copyable_function - implements P2548R6 - CXX26 ..
t202.dir/_22132a.cpp	22.13.2 Primitive numeric output conversion. Testing for success ...
t202.dir/_22133a.cpp	22.13.3 Primitive numeric input conversion. Testing for success ...
t203.dir/_20_1454p1b.cpp	Comparisons for reference_wrapper - implements P2944R3 - CXX26 ...
t203.dir/_20_146b.cpp	20.14.6 views::concat - implements P2542R7 - CXX26 ...
t203.dir/_207_132a1c_s.cpp	constexpr STD shared_ptr - implements P3037R1 - CXX26 ...
t203.dir/_2077p3a.cpp	Member visit - implements P2637R3 - 2023 - ...
t203.dir/_27_7_9_1c.cpp	27.7.9, Formatting of chrono Time Values - implements ...
t215.dir/_2332a.cpp	Interfacing stringstreams with string_view - implements P2495R3 - ...
t215.dir/_2681b.cpp	6.8.1 Types General. - P2670R0 "Non-transient constexpr allocation" ...
t217.dir/_214a7a.cpp	to_string or not to_string - implements P2587R3 - ...
t238.dir/_2345f0_14a.cpp	bitset<> template ctor from stringview subrange. - implements ...
t244.dir/_2437a.cpp	24.3.7 [containers.sequences.general] - inplace_vector - A dynamically-resizable vector ...
t244.dir/_24441a.cpp	Better Lookups for map and unordered_map - implements ...
t244.dir/_247226a.cpp	4.7.2.2.6 [views.span] Element access - span.at() - implements ...
t244.dir/_24722a.cpp	24.7.2.2 [views.span] Class template span - CXX20 ...
t244.dir/_24722b.cpp	24.7.2.2 [views.span] STD span over an initializer list ...
t244.dir/_267121b.cpp	views::(take|drop)_exactly - implements P3230R0 - CXX26 ...
t244.dir/_267121c.cpp	views::slice - implements P3216R0 - CXX26 ...

Filename	Test Case
t244.dir/_267121d.cpp	views::transform_join - implements P3211R0 - CXX26 ...
t251.dir/_271017a.cpp	27.10.17 Saturation arithmetic [numeric.sat] - implements P0543R3 - ...
t258.dir/_2533a.cpp	STD basic_const_iterator should follow its underlying type's convertibility ...
t260.dir/_2652b.cpp	26.5.2 Header <bit> synopsis - implements P3103R1: More ...
t260.dir/_2652c.cpp	26.5.2 Header <bit> synopsis - implements P3104R2: Bit ...
t261.dir/_2626Y84.cpp	Add tuple protocol to complex - implements P2819R1 ...
t263.dir/_26572g26b.cpp	26.5.7.2 Vector API for random number generation ...
t263.dir/_26572g26.cpp	26.5.7.2 function template generate_canonical ...
t275.dir/_3174b.cpp	Runtime format strings - implements P2905R2 - CXX26 ...
t275.dir/_3174c.cpp	Header <format> - implements P2093R14 Formatted output, P2216R3 ...
t27k.dir/_31102a.cpp	31.10.2 [filebuf] Class template basic_filebuf - implements P1759R6 ...
t27k.dir/_317635b.cpp	Printing Blank Lines with println - implements P3142R0 ...
t290.dir/_2914a.cpp	Hashing support for STD chrono value classes - ...
t305.dir/_30_33p1b.cpp	Contracts for C++ - contracts are runtime ...
t305.dir/_30_33p1c.cpp	Contracts for C++ - contracts are runtime ...
t305.dir/_30_33p1d.cpp	Contracts for C++ - contracts are runtime requirements ...
t305.dir/_30_33p1e.cpp	Contract testing support - contracts are runtime requirements ...
t305.dir/_30_33p1f.cpp	Contracts for C++ - contracts are runtime requirements ...
t305.dir/_30_33p1g.cpp	Contracts for C++ - Compile-time Evaluation - implements ...
txd2.dir/_3031a.cpp	STD text_encoding: text encodings identification - implements P1885R12
t26a.dir/c2y_09110a.cpp	Make assert() macro user friendly for C and ..
t26a.dir/c2y_1355a.cpp	Concept and variable-template template-parameters...
t26a.dir/c2y_1374a.cpp	Pack Indexing - implements P2662R3 - see also ...
t26a.dir/c2y_1374b.cpp	P2355R2: Postfix fold expressions - implements P2355R2 - ...
t26a.dir/c2y_1374c.cpp	The Oxford variadic comma - implements P3176R0 - ...
t26a.dir/c2y_142o22a.cpp	ADL-proof STD projected. Argument Detected Logic - implements ...
t26a.dir/c2y_17_2_1.cpp	17.2.1 Header <cstddef> synopsis - Make direct-initialization for ...
t26a.dir/c2y_17_3_2.cpp	17.3.2 Header <version> synopsis - Freestanding Feature-Test Macros ..
t26a.dir/c2y_20_11_142b.cpp	Hazard pointers - Hazard Pointers for C++26 - ...
t26a.dir/c2y_20_2_2a.cpp	std::uninitialized<T> - implements P3074R2 - CXX26 - 2023 ...
t26a.dir/c2y_20_3_1_3a.cpp	Mixed comparisons for smart pointers - implements P2249R6 ...
t26a.dir/c2y_21_3_11.cpp	21.3.11 Emitting messages at compile time - P2758R2 ..
t26a.dir/c2y_21_3_7.cpp	21.3.7 Disallow Binding a Returned Glvalue to a ...
t26a.dir/c2y_21_3_8_6.cpp	21.3.8.6 STD constant_wrapper - P2781R4 - C++26 ...
t26a.dir/c2y_21_3_87.cpp	Remove return type deduction in STD apply - ...
t26a.dir/c2y_22_10_14a.cpp	Universal Template Parameters - P1985R3 - C++26 ...
t26a.dir/c2y_22_10_14b.cpp	A Simple Approach to Universal Template Parameters - ...
t26a.dir/c2y_22_10_14.cpp	22.10.14 Function templates bind_front and bind_back - Bind ...
t26a.dir/c2y_22_16_2.cpp	22.16.2 Header <debugging> synopsis - Standard library header ...
t26a.dir/c2y_22_2_2a.cpp	22.2.2 Object relocation in terms of move plus ...
t26a.dir/c2y_22_2_2.cpp	22.2.2 Relax wording to permit relocation optimizations in ...
t26a.dir/c2y_22_4_7.cpp	22.4.7 STD variant_alternative_index and STD tuple_element_index ...
t26a.dir/c2y_22_5_3.cpp	22.5.7 STD optional<T&> - P2988R4 - C++26 ...
t26a.dir/c2y_22_6_4.cpp	22.6.4 STD variant_alternative_index and STD tuple_element_index...
t26a.dir/c2y_23_3_3_2a.cpp	sub-string_view from string - P3044R0 - C++26 ...
t26a.dir/c2y_23_3_3_8.cpp	23.3.3.8 String operations sub-string_view from string - P3044R0 ...
t26a.dir/c2y_237p1a.cpp	Concatenation of strings and string views. - implements ...
t26a.dir/c2y_24_3_3_6a.cpp	24.3.3.6 mdspan of All Dynamic Extents - P2299R4 ...
t26a.dir/c2y_24_3_3_6b.cpp	24.3.3.6 Copy and fill for mdspan - P3242R0 ...
t26a.dir/c2y_24_3_3_6.cpp	24.3.3.6 dextents Index Type Parameter, Better mdspan's CTAD ..
t26a.dir/c2y_24_3_7.cpp	24.3.7 inplace_vector - A dynamically-resizable vector with fixed ...
t26a.dir/c2y_26_2_1_1.cpp	26.2.1.1 maybeview - A view of 0 or ...
t26a.dir/c2y_26_2_1_2.cpp	26.2.1.2 [range.nullable.view] nullable view - P1255R12 - C++26 ...

Filename	Test Case
t26a.dir/c2y_26_5_3a.cpp	26.5.3 View interface - view_interface::at() - P3052R1 - ...
t26a.dir/c2y_26_6_4a.cpp	26.6.4a Add STD views::upto(n) - P3060R1 - C++26 ...
t26a.dir/c2y_274a.cpp	Enabling list-initialization for algorithms - implements P2248R8 - ...
t26a.dir/c2y_27822a.cpp	27.8.2.2 constexpr Stable Sorting - P2562R1 - C++26 ...
t26a.dir/c2y_28_6_2a.cpp	Resolve inconsistencies in begin/end for valarray and braced …
t26a.dir/ c2y_28_9_13_blas1.cpp	28.9.1.3 BLAS 1 algorithms - implements P1673R12 ...
t26a.dir/ c2y_28_9_14_blas2.cpp	28.9.1.4 BLAS 2 algorithms - implements P1673R12 …
t26a.dir/ c2y_28_9_15_blas3.cpp	28.9.1.5 BLAS 3 algorithms - implements P1673R12 ...
t26a.dir/c2y_28_9a.cpp	Fix C++26 by optimizing linalg::conjugated for noncomplex value ..
t26a.dir/c2y_28_9c.cpp	Basic linear algebra algorithms - <linalg>: A free ...
t26a.dir/c2y_28_9.cpp	28.9 Basic linear algebra algorithms - <linalg>: A ...
t26a.dir/c2y_30_4a.cpp	Quantities and units library - implements P3045R0 - ...
t26a.dir/c2y_31_12_4a.cpp	Formatting of std::filesystem::path - implements P2845R6 - CXX26 ...
t26a.dir/c2y_3174a.cpp	Header <print> synopsis - test runtime formatting - ...
t26a.dir/c2y_33_11_2.cpp	33.11.2 Read-copy update (RCU) - Safe reclamation. Read-copy ...
t26a.dir/c2y_33_4_3.cpp	33.4.3 [thread.thread.class.general] - Hassle-free thread attributes
t26a.dir/c2y_33_5_2a.cpp	33.5.2 Header <atomic> synopsis [atomics.syn] - Atomic floating-point ...
t26a.dir/c2y_33_5_2.cpp	33.5.2 Header <atomic> synopsis [atomics.syn] - Expose std::atomic_ref's
t26a.dir/c2y_33_5_7.cpp	33.5.7 Expose std::atomic_ref's object address - P2835R2 - ...
t26a.dir/c2y_6_7_55.cpp	6.7.5.5 Dynamic storage duration - Freestanding Language: Optional ...
t26a.dir/c2y_7_6_18a.cpp	Allowing exception throwing in constant-evaluation...
t26a.dir/c2y_7_6_18b.cpp	Inspecting exception_ptr - implements P2927R2 - C++26 ...
t26a.dir/c2y_9_12_4.cpp	9.12.4 Carries dependency attribute - C++26 ...

New in lvs23a:

This release addresses many defect reports from customers in the 22a release and adds new tests for C++20 and C++23 features. As of this release support for older C++ versions prior to C++11 is dropped.

Version	ISO Document	__STDC_VERSION__	Comments
CXX11	ISO/IEC 14882:2011	201103L	C++0x
CXX14	ISO/IEC 14882:2014	201402L	C++1y
CXX17	ISO/IEC 14882:2017	201703L	C++1z
CXX20	ISO/IEC 14882:2020	202002L	C++2a
CXX23			Work In Progress

C++ Releases

Release 23a New Test Cases

There are 220 new test cases, documented in "newcases-lvs20b-lvs23a.txt", in multiple directories. These new test cases predominantly pertain to the C++20 and C++23 standards. This release also contains initial support for testing freestanding C++ in t161.dir/. Directory t01a.dir/ contains some tests for modules, though more detailed testing is found in xvs23a.

Release 23a Bug Fixes

There were issues in unarchiving negtests, causing partial results and loss of sequencing. These issues have been fixed. The single m19.in test case archive has been split into individual directory archives, A number of files had duplicate main's. A number of test cases were just skeletons, these have been filled in with the actual test case. Test cases with dynamic exceptions have been modified since ISO C++17 does not allow dynamic exception specifications. Test cases involving the *register* keyword ave been modified since *register* has been deprecated. Test cases involving the *volatile* keyword ave been modified since *volatile* has been deprecated in many contexts.

The 23a update release represents 3+ years of test case bug fixing, infrastructure improvements, and new test cases for C++20, and C++23, language and library enhancements. There are many improvements in enhancing the test cases themselves and also enhancing the reporting of the results, through the new html interfaces for reporting coverage, commentary on the intent of the test cases and improved standards conformance reporting.

Release 23a New Test Cases

This release also adds new test cases addressing:

- Char and string types,
- some support for modules,
- the “spaceship” operator <=> and
- char8_t, u8string and u8string_view
- concepts
- coroutines
- version header
- source_location
- format
- span
- ranges and range adapters
- syncstream,
- init-statements and initializers in the range *for* statement
- new attributes: [[no_unique_address]], [[likely]], [[unlikely]]
- pack-expansions in lambda init-captures
- consteval
- constinit
- aggregate initialization using parentheses
- check compiler feature and attribute definitions.

Infrastructure

Installers are available on the PlumHall server for Linux (installPH.sh) and Windows (installPH.bat). These installers greatly simplify installing the PlumHall distributions in a standard layout as described below. Download the installers from the plumhall2b.com server and create the default installations. The installers are customized for each customer:

```
ftp plumhall2b.com
Connected to plumhall2b.com.
220---------- Welcome to Pure-FTPd [privsep] [TLS] ----------
Name (plumhall2b.com:doug): OscarWilde1854
331 User OscarWilde1854 OK. Password required
Password:
passive
get installPH.sh
get installPH.bat
quit

~/installPH.sh --help
Download, check the MD5 hash and install the PlumHall test suites in $HOME/PlumHall/
Options:
  --cvs=<version>       : install CVS version e.g. --cvs=CVS002
  --xvs=<version>       : install XVS version e.g. --xvs=XVS002
  --lvs=<version>       : install CVS version e.g. --lvs=LVS002
  --compiler=<name>     : brief compiler name used to create directory
                           structure, e.g. gcc, edg, clang, etc
  --PW=<zip password>   : zip password for distribution
  --login=<login name>  : login name given in download instructions, e.g. techcontactname
  --username=<username> : suffix to user name given in download instructions,
                           e.g. techcontactname8345
  --keep                : do not delete existing directories before unpacking the distributions.
  --verbose             : chatty
  --help                : help me if you can…

Note: uses scp to securely copy the distributions from the plumhall2b.com server, zip to unpack the
distribution and md5sum to calculate the MD5 hash.
```

Executing the scripts will download your distributions and check the MD5 sums. If the sums do not match the scripts will exit with an error, please contact PlumHall should this occur. If the MD5 sums are correct then compressed files will be expanded and installed in the standard directory structure.

If you have trouble with the install scripts, you may enter the commands:

```
ftp plumhall2b.com
login: OscarWilde1854@plumhall2b.com
passwd: *********
passive
get installPH.sh or get installPH.bat
get lvs23a-LVS000.tar.gz
get lvs23a-LVS000.tar.gz.md5
get lvs23a-LVS000.zip
get lvs23a-LVS000.zip.md5
…
bye
```

If you did a manual download you may then run the installer script with the option *--nodownload* to unpack, check the MD5 signatures and create and populate the standard directory structures. The installer extracts into a directory named ~/PlumHall/ by default. Please ensure that the md5sum utility is available and verify that the MD5 sums compare.

The script will ask for your plumhall2b ftp password as part of the installation process.

The default folder naming convention is:

```
<Test Suite><PlumHall Release Year>-<Compiler Mnemonic>-c<Standards Year>/

e.g. lvs23a-gcc-c20/
```

For example, to create directories for each of the standards years C++17 and C++20, for compilers gcc and clang:

```
installPH.sh --stdyear=17 --stdyear=20 --compiler=gcc --compiler=clang
```

There are three main points of customization:

- flags.h for C/C++ version options,
- compiler-flags.h for compiler-specfifc options and
- envsuite.sh (envsuite.bat) to customize the execution environment.

Some customization is possible by using envsuite command line options. For example: `envsuite.sh cc=g++-latest` sets the version of gcc to use. Type `envsuite.sh -h` for current arguments. Further customization requires editing envsuite.sh(.bat)

The `envsuite` script has been modified to more easily support a standard PlumHall directory structure and multiple compilers on the command line. The standard directory structure is:

```
~/PlumHall/xvs23a-<cc>-c20/          build directory
~/PlumHall/xvs23a                    source directory
~/PlumHall/xvs23a-<cc>-c20-setup/    setup directory updated by script save-setup

~/PlumHall/lvs23a-<cc>-c20/
~/PlumHall/lvs23a/
~/PlumHall/lvs23a-<cc>-c20-setup/

~/PlumHall/cvs23a-<cc>-c20/
~/PlumHall/cvs23a/
~/PlumHall/cvs23a-<cc>-c20-setup/
```

where <*cc*> is gcc or clang, or cl on Windows. The script `createDestination.sh` is available to create and populates these default directories, though the `installPH` scripts do this by default. It takes a command line argument cc=<gcc | clang | cl> to create different build directories for multiple compiler testing. The script `envsuite` and `save-setup` also take the argument cc=. The scripts take arguments cc=gcc or cc=gcc-latest to cc=clang-12 as examples.

The file flags.h customizes for C++ standards release version:

```
flags.h   envsuite
folders > 33 > 3kp1b0n50fj_szpj2xxvdgyw0000gn > T > ch.sudo.cyberduck > 075b1092-7ca4-43d5-9a47-bb5e1a094393 > home > doug > PlumHall > xvs22a-gcc-c20 > flags.h
16  /*************************************************************************/
17  /*                  More flexible version defines                        */
18  /*                    See defines in def.h                               */
19  /*                                                                       */
20  /*#define PH_CXX90 1990                                                  */
21  /*#define PH_CXX03 2003                                                  */
22  /*#define PH_CXX11 2011                                                  */
23  /*#define PH_CXX14 2014                                                  */
24  /*#define PH_CXX17 2017                                                  */
25  /*#define PH_CXX20 2020                                                  */
26  /*#define PH_CXX23 2023                                                  */
27  /*#define PH_CXXWP 10000                        i.e. far in the future   */
28  /*                                                                       */
29  /* e.g. #if PH_CXX_VERSION == PH_CXX03                                   */
30  /* e.g. #if PH_CXX_VERSION == PH_CXX20                                   */
31  /* e.g. #if PH_CXX_VERSION >= PH_CXX14                                   */
32  /* e.g. #if PH_CXX_VERSION <= PH_CXX11                                   */
33  /* e.g. #if PH_CXX_VERSION >= PH_CXX17 && PH_CXX_VERSION <= PH_CXX20     */
34  /*************************************************************************/
35  /*************************************************************************/
36  /*                                                                       */
37  /*                                                                       */
38  #define PH_CXX_VERSION PH_CXX20 /* <-- change this line - see defs.h     */
39  /*                                                                       */
40  /*                                                                       */
41  /*************************************************************************/
```

Customization of flags.h

The release numbers in flags.h and envsuite MUST Be kept in sync.

```
Volumes > doug > PlumHall > xvs22a-gcc-c20 > compiler-flags.h > ...
11  /*************************************************************************/
12  /*                                                                       */
13  /*  Part 2 Compiler-specific defines                                     */
14  /*  Set these values to globally enable or disable certain tests.       */
15  /*  Permits more readable and compact test reslts.                      */
16  /*                                                                       */
17  /*************************************************************************/
18
19  #ifdef __cplusplus
20  /*************************************************************************/
21  /*                                                                       */
22  /*  C++ Compilers.                                                       */
23  /*                                                                       */
24  /*************************************************************************/
25
26  #ifdef __clang__
27  /*code specific to clang compiler*/
28  #define DISALLOW_TZDB no tzdb
29  #elif defined(__GNUC__) && !defined(__INTEL_COMPILER)
30  /*code for GNU C compiler */
31  #define DISALLOW_EXECUTION_POLICY noexecpolicy
32  #define DISALLOW_FLUSH_EMIT no flush emit
33  #define DISALLOW_TZDB no tzdb
```

Customization of compiler-flags.h

compiler-flags.h allows for setting flags specific to a particular compiler. These flags are often set to get around compile errors which prevent viewing overall results. For example lang.c and lib.c link in relevant test case object files. If a compile of a particular test fails, none of the results of the other tests can be seen.

It is very important that you review envsuite(.bat), flags.h and compiler-flags.h to choose the correct settings for your compiler.

```
#!/bin/bash
################################################################################
#
# please edit your  save-setup  file to match your choice for PHDST
#
# envsuite --- environment for compiler ---
# $Revision: 26 2022-08-31 Copyright (c) 2086-2022, Plum Hall Inc. $

###format            configure for your target compiler
cc=gcc
compiler=${cc}
suite=lvs
relyear=22
suffix=a
release=${relyear}${suffix}
cvs=cvs
################################################################################
#                                                                              #
#                                                                              #
stdyear=20                  # <-- change this line to build other releases   #
#                                                                              #
#                                                                              #
################################################################################
```

Customization of envsuite.sh

The release numbers in flags.h and envsuite MUST Be kept in sync. Customization of ensuite requires a detailed reading of the source of the script. Usually ensuite is used to set compiler and linker directories as required to build.

The build system itself has been enhanced. In prior releases adding a test case required hand editing multiple different makefiles and scripts. In this release this is no longer required, the makefiles and script automatically adjust to addition/deletion of test cases.

The build system did not adapt well to the new requirements imposed by C++ modules. The t01a.dir directory contains all the module test cases. The makefile and build script are customized to build modules in the style of gcc. At this time there is no support for building modules in the Microsoft cl.exe or clang styles.

In order to test modules and coroutines version 11 of gcc is required on Linux, and c++latest on Windows. For example on Linux:

```
. envsuite cc=g++-11
```

On Windows install the latest version of cl.exe and ensure that STD=c++latest is set in envsuite.bat.

A number of visualization tools have been added. At the end of each buildmax build the following html files are created:

```
coverage-cxx20.html
commentary-cxx20-lib.html
conform-cxx20-lib.html
report-cxx20.html
```

The file conform-cxx-lib.html shows a summary of successful tests and those with issues:

LibSuite++ Conformance cxx20 lvs22a - gcc 11.0.1 20210 - Linux - Sat Jan 28 20:28:25 2023						
negtests | Negative Tests - tests that should fail.						
t007.dir | Threads, wait, notify						
t01a.dir | Modules						
t01b.dir | Coroutines						
t160.dir | Attributes, constexpr						
t170.dir | C/C++ Headers, C Library Functions						
Compile error(s) ... 17_15r1a.cpp						Is_pointer_interconvertible_with_class, 20.15.10 Mem N4878 - CXX17
Compile error(s) ... 17624F20.cpp						Transactional memory - needs syncstream header... o - CXX20
ERROR in ... 178r1a.cpp at line 49 ... 178r1a.cpp						source location. Implements P1208R6 Adopt source_l - skipped if the header is not available - CXX20
ERROR in ... 178r1a.cpp at line 51: "int main(int, char**)" != "main" ... 178r1a.cpp						source location. Implements P1208R6 Adopt source_l - skipped if the header is not available - CXX20
Test Case Failed ... 178r1a.cpp						source location. Implements P1208R6 Adopt source_l - skipped if the header is not available - CXX20
54	93%	2	4	0	0	See results in the output log.
t180.dir | Numeric Limits						
t181.dir | Numeric Limits, Signalling NaN						
t182.dir | New, Delete, cstdint, cstdlib						
t183.dir | alloc, typeinfo, cast, exception, Tables 21-25						
t190.dir | logic_error, domain_error, runtime_error, overflow_error, underflow_error						
t200.dir | Formatting, pair, tuple, memory management tools						
t201.dir | allocators, invoke, bitset, ratio						
t202.dir | type_traits, invoke, hash, decay, make_pair, make_tuple, optional, functional						

conform-cxx-lib.html

The value in the **Expected** column is the number of test cases, where Expected = Actual + Errors + Faults + Aborts. The **Actual** column is the sum of the number of test results that matched expected values/behavior plus the number of skipped test cases. The value in the **Skipped** column is the number of skipped test cases. The value in the **Errors** column is the sum of the number of test cases that meet one of the following conditions:

- One or more unexpected values are returned in the test items.
- A compile error occurred, when the test file was compiled.
- An execution error occurred, when the test was executed.

The value in the **Abort** column is the number of test cases that and abort occurred. The value in the Faults column is the number of tests that meet one of the following conditions:

- An uncaught exception occurred when the test was executed.
- An internal error occurred when the test file was compiled.
- Unknown or unreported test results.

The links to the .out log file and .cpp source file help to quickly find what the issue is and where.

t007.out

Output Log

```
t007.out:
===== _18423a11_s
***** Reached first test *****
_18423a11_s (><) passed
#PASSED:  _18423a11_s
***** 1 individual successful item in _18423a11_s *****
***** 1 successful test case in _18423a11_s *****
***** 0 errors detected in _18423a11_s *****
***** 0 skipped sections in _18423a11_s *****
===== _18511a45_s
***** Reached first test *****
_18511a45_s (><) passed
#PASSED:  _18511a45_s
***** 1 individual successful item in _18511a45_s *****
***** 1 successful test case in _18511a45_s *****
***** 0 errors detected in _18511a45_s *****
***** 0 skipped sections in _18511a45_s *****
===== _18511a46_s
***** Reached first test *****
_18511a46_s (><) passed
#PASSED:  _18511a46_s
***** 1 individual successful item in _18511a46_s *****
***** 1 successful test case in _18511a46_s *****
***** 0 errors detected in _18511a46_s *****
```

t007.out.html

The log filename is a link to the actual output log of test result summaries for the entire test directory. The center column shows errors linked to the compiler log file showing compile errors.

_17624F20.clg

Compiler Log

```
ome/doug/PlumHall//lvs22a/conform/t170.dir/_17624F20.cpp: In function ‘int func()’:
ome/doug/PlumHall//lvs22a/conform/t170.dir/_17624F20.cpp:35:3: error: ‘synchronized’ was not declared in this scope
 35 |   synchronized {
    |   ^~~~~~~~~~~~
ome/doug/PlumHall//lvs22a/conform/t170.dir/_17624F20.cpp:40:1: warning: no return statement in function returning non-void [-W
 40 | }
    | ^
ome/doug/PlumHall//lvs22a/conform/t170.dir/_17624F20.cpp: In function ‘int f()’:
ome/doug/PlumHall//lvs22a/conform/t170.dir/_17624F20.cpp:45:4: error: ‘atomic_noexcept’ was not declared in this scope
 45 |    atomic_noexcept { // begin transaction
    |    ^~~~~~~~~~~~~~
ome/doug/PlumHall//lvs22a/conform/t170.dir/_17624F20.cpp:50:1: warning: no return statement in function returning non-void [-W
 50 | }
    | ^
/home/doug/PlumHall//lvs22a/conform/defs.h
 /usr/lib/gcc/x86_64-linux-gnu/11/include/stddef.h
 /home/doug/PlumHall//lvs22a-gcc-c20/flags.h
. /home/doug/PlumHall//lvs22a-gcc-c20/compiler-flags.h
. /usr/include/stdio.h
.. /usr/include/x86_64-linux-gnu/bits/libc-header-start.h
... /usr/include/features.h
.... /usr/include/x86_64-linux-gnu/sys/cdefs.h
..... /usr/include/x86_64-linux-gnu/bits/wordsize.h
```

t170.dir/_17624F20.clg.html

The source file is linked to browse the test source file.

```
1.
2.  #define MAIN_FILE
3.
4.  #include "defs.h"
5.
6.  /* _17_15r1a is_pointer_interconvertible_with_class, 20.15.10 Member relationships N4878 - CXX17 */
7.
8.  #if defined(SKIP_17_15r1a)
9.  #elif defined(DISALLOW_CXX17)
10. #define SKIP_17_15r1a  _CXX17
11. #elif defined(DISALLOW_CXX14)
12. #define SKIP_17_15r1a  _CXX14
13. #elif defined(DISALLOW_CXX11)
14. #define SKIP_17_15r1a  _CXX11
15. #elif defined(DISALLOW_CXX03)
16. #define SKIP_17_15r1a  _CXX03
17. #endif
18.
19. #if (!defined(SKIP_17_15r1a)&&!defined(SKIP17)&&!defined(ONLY))||defined(CASE_17_15r1a)
20. #include <type_traits>
21.
22. struct A { int a; };
23. struct B { int b; };
24. struct C : public A, public B {};
25. #endif /* CASE_17_15r1a */
26.
27. #include "final_defs.h"
28. int main(int argc, char *argv[])
29. {
30.         if (argc > 1 && argv[1] != 0)
31.                 Debug = TRUE;
32.         Filename = "_17_15r1a.cpp";
```

Top

t170.dir/_17624F20.cpp.html

The `make-commentary` script creates an html file (commentary-cxx.html for example) that shows a brief commentary of the purpose of each test case by folder name and test name:

LibSuite++ cxx20 Commentary lvs22a - gcc 11.0.1 20210 - Linux - Sat Jan 28 20:28:25 2023	
negtests	Negative Tests - tests that should fail.
t007.dir	Threads, wait, notify
t01a.dir	Modules
17p2b_m.cpp	Merging Modules; main - C++20 - implements P1103R3
17p2b_m.cpp	Merging Modules; main - C++20 - implements P1103R3
t01b.dir	Coroutines
t160.dir	Attributes, constexpr
t170.dir	C/C++ Headers, C Library Functions
17_1122p1a.cpp	symmetry for spaceship - CXX20 implements P0905R1 2019
17_1122p1a.cpp	symmetry for spaceship - CXX20 implements P0905R1 2019
17_1122p1a.cpp	symmetry for spaceship - CXX20 implements P0905R1 2019
17_1122p1a.cpp	symmetry for spaceship - CXX20 implements P0905R1 2019
17_1122p1a.cpp	symmetry for spaceship - CXX20 implements P0905R1 2019
17_1122p1a.cpp	symmetry for spaceship - CXX20 implements P0905R1 2019
17_12_14p1a.cpp	bind_front - Simplified partial function application -- CXX20
17_12_14p1a.cpp	bind_front - Simplified partial function application -- CXX20
17_12_14p1a.cpp	bind_front - Simplified partial function application -- CXX20
17_12_14p1a.cpp	bind_front - Simplified partial function application -- CXX20
17_12_14p1a.cpp	bind_front - Simplified partial function application -- CXX20

commentary-cxx-lib.html

The `make-coverage` script generates the html file coverage-xvsxxa.html which shows for each C/C++ release, the Defect Report number, the directory test case file and a brief description of the Defect Report. This is useful to find which directories and test cases address a particular feature introduced by the Defect Report.

LibSuite++ Coverage lvs22a - gcc 10.2.0 - Linux - Sun Jan 29 06:46:22 2023			
			LVS11
			LVS14
			LVS17
			LVS20
.dir	p0482r6	_17p2a.cpp	_17p2a reviewing deprecated facilities of C++17 for C++20
.dir	p0718r2	_17p2a.cpp	_17p2a reviewing deprecated facilities of C++17 for C++20
.dir	p0768r1	_17p2a.cpp	_17p2a reviewing deprecated facilities of C++17 for C++20
.dir	p0883r2	_17p2a.cpp	_17p2a reviewing deprecated facilities of C++17 for C++20
.dir	p0966r1	_17p2a.cpp	_17p2a reviewing deprecated facilities of C++17 for C++20
.dir	p0767r1	_17p2a.cpp	_17p2a reviewing deprecated facilities of C++17 for C++20
.dir	p1208r6	_178r1a.cpp	_178r1a source location. Implements P1208R6 Adopt source_location for C-
.dir	p0553r4	_2652a.cpp	_2652a 26.5.2 Header <bit> synopsis - implements P0553R4: Bit operations
.dir	p1956r1	_2652a.cpp	_2652a 26.5.2 Header <bit> synopsis - implements P0553R4: Bit operations
.dir	p0415r1	_2661i1a.cpp	_2661i1a constexpr ctor real imag == !=, CXX14 - implements N3302, N3669
.dir	p1357r1	_20_102k11.cpp	_20_102k11 header type_traits - CXX17 - implements P0006R0, P1357R1 Tra and conditional
.dir	p0771r1	_20_14_1732a.cpp	_20_14_1732a 20.14.17.3.2 Constructors and destructor. STD function move
.dir	p1227r2	_1872p2a.cpp	_1872p2a Signed ssize() functions, unsigned size() functions - implements

coverage-cxx.html

The make-report script generates a table showing all files with the associated commentary:

HTML Report of Commentary from All Sources - cxx20	
lvs-22a	
conform	
t170.dir	
_1723p1a.cpp	char8_t: a type for utf-8 char p1082r0 - 2019 CXX20
7623a61d_s.cpp	_17412Y61d_s _17312X61d_s stdlib Library functions must not be macros
17623a41b.cpp	contents of cname same as name.h
_17647a21.cpp	_17435Y21 may install different handler functions during execution
17624F20b.cpp	coroutines - lazy generation, needs <coroutine> and <range> headers else skipped - CXX20
7623a61a_s.cpp	_17412Y61 _17612i61 ctype Library functions must not be macros
175432p1a.cpp	thou shalt not specialize std function templates - CXX20 - implements p0551r3
_17623a31.cpp	_17412Y31 Table 14: C++ headers for C library
17535nt_31.cpp	allocator requirements, - implements lib 2593 - CXX17
17623a51_s.cpp	_17412Y51_s C Library Macros must be macros
_17655k21.cpp	for a non-virtual member function described in the C++ standard newcase _17655k21 t170.phe 1550
_17p1b.cpp	Adding [nothrow-]swappable traits - implements P0185R1
_17p2a.cpp	reviewing deprecated facilities of C++17 for C++20
7_12_14p1a.cpp	bind_front - Simplified partial function application -- CXX20
_17331p1a.cpp	Make stateful allocator propagation more consistent for operator+(basic_string) - CXX20

report-cxx-lib.html

Again the file names are links for convenient browsing of the test case suite. All of these html documents are produced dynamically from the source as the last steps in the `buildmax` script.

C++23 Library Features

Feature	Paper	Addressed	Test Cases
Stacktrace library	P0881R7 P2301R1		t200.dir/_20213_1.cpp, t200.dir/_20213_2.cpp, t200.dir/_20213_3.cpp, t200.dir/_20213_4.cpp
<stdatomic.h>	P0943R6		negtests/m19.in, t290.dir/_2963p1a.cpp
std::is_scoped_enum	P1048R1		t203.dir/_20_15_5_4a.cpp
basic_string::contains(), basic_string_view::contains()	P1679R3		t231.dir/_237p1a.cpp
std::to_underlying	P1682R3		t203.dir/_20_2_8_1a.cpp
Relaxing requirements for time_point<>::clock	P2212R2		t203.dir/_27_7_9_1a.cpp
DR: std::visit() for classes derived from std::variant	P2162R2		
DR: Conditionally borrowed ranges	P2017R1		t244.dir/_2455g1a.cpp
DR: Repairing input range adaptors and std::counted_iterator	P2259R1		t244.dir/_247161a.cpp
Providing size feedback in the Allocator interface	P0401R6		t232.dir/_20_10_8_1a.cpp
<spanstream> : string-stream with std::span-based buffer	P0448R4		t225.dir/_29_9_1a.cpp, t225.dir/_29_9_3a.cpp
std::out_ptr(), std::inout_ptr()	P1132R8		t203.dir/_20_11_9_1a.cpp
constexpr type_info::operator==()	P1328R1		t183.dir/_1861a31.cpp
Iterator pair constructors for std::stack and std::queue	P1425R4		t235.dir/_22_6_6_3a.cpp
Non-deduction context for allocators in container deduction guides	P1518R2		t201.dir/_20_123p2a.cpp
ranges::starts_with() and ranges::ends_with()	P1659R3		
Prohibiting std::basic_string and std::basic_string_view construction from nullptr	P2166R1		t211.dir/_2132a_101.cpp, t211.dir/_2132a_101_c16, t211.dir/_2132a_101_c8, t211.dir/_2132a_101_c32, t211.dir/_2132a_101_w
std::invoke_r()	P2136R3		t203.dir/_20_142a.cpp
Range constructor for std::basic_string_view	P1989R2		t216.dir/_29_8_4_4_10a.cpp
Default template arguments for pair's forwarding constructor	P1951R1		t200.dir/_2042p1b.cpp
Remove Garbage Collection and Reachability-Based Leak Detection (library support)	P2186R2		t203.dir/_207_137a1a.cpp and negtests
DR: join_view should join all views of ranges	P2328R1		t244.dir/_247111a.cpp
DR: view does not require default_initializable	P2325R3		t244.dir/_2441a.cpp
DR: Range adaptor objects bind arguments by value	P2281R1		t244.dir/_2472a.cpp
DR: constexpr for std::optional and std::variant	P2231R1		
DR: std::format() improvements	P2216R3		t244.dir/_20201r1g.cpp
DR: lazy_split_view and redesigned split_view	P2210R2		t244.dir/_24713a.cpp
zip	P2321R2		t244.dir/_247191a.cpp
Heterogeneous erasure overloads for associative containers	P2077R3		t237.dir/_2252a.cpp
std::byteswap()	P1272R4		t260.dir/_2654a.cpp
Printing volatile T*	P1147R1		t27a.dir/_27751a.cpp
basic_string::resize_and_overwrite()	P1072R10		t212.dir/_21335a11.cpp, t212.dir/_21335a11_w.cpp, t212.dir/_21335a11_c8.cpp, t212.dir/_21335a11_c16.cpp, t212.dir/_21335a11_c32.cpp,
Monadic operations for std::optional	P0798R8		t202.dir/_2065m01.cpp
std::move_only_function	P0288R9		t202.dir/_20_14_1742a.cpp
Add a conditional noexcept specification to std::exchange	P2401R0		t290.dir/_2965g_180c.cpp
Require span & basic_string_view to be *TriviallyCopyable*	P2251R1		t202.dir/_201554.cpp
Clarifying the status of the “C headers”	P2340R1		t170.dir/_17p2a.cpp
DR: Fix ranges::istream_view	P2432R1		t244.dir/_2465g1n.cpp
DR: Add support for non-const-formattable types to std::format	P2418R2		t170.dir/_17624F20c.cpp
DR: What is a view	P2415R2		t244.dir/_24753a.cpp
DR: fixing locale handling in chrono formatters	P2372R3		t200.dir/_2020152r1.cpp
DR: Cleaning up integer-class types	P2393R1		
Contract-based programming	p0542r5		t305.dir/_30_33p1a.cpp

C++20 Library Features

Feature	Paper	Addressed	Test Cases
std::endian	P0463R1		t203.dir/_20_159o1a.cpp
Extending std::make_shared() to support arrays	P0674R1		t203.dir/_20_1136o1a.cpp, t203.dir/_20_1136o1a.cpp
Floating-point atomic	P0020R6		t290.dir/_2963p1a.cpp
Synchronized buffered (std::basic_osyncstream)	P0053R7		t27k.dir/_27_101o1a.cpp, t27k.dir/_27_1021o1a2.cpp, t27k.dir/_27_1021o1a.cpp, t27k.dir/_27_1021o1b.cpp, t27k.dir/_27_1021o1c.cpp, t27k.dir/_27_1021o1d.cpp, t27k.dir/_27_1021o1e.cpp, t27k.dir/_27_1022o1a.cpp, t27k.dir/_27_1023o1a.cpp, t27k.dir/_27_1025o1a.cpp, t27k.dir/_27_102ao1a.cpp, t27k.dir/_27_102o1a.cpp, t27k.dir/_27_1033o11.cpp
constexpr for <algorithm> and <utility>	P0202R3		t251.dir/_2521o11.cpp
More constexpr for <complex>	P0415R1		t260.dir/_2626o1a.cpp
Make std::memory_order a scoped enumeration	P0439R0		t290.dir/_294o11.cpp
String prefix and suffix checking: string(_view) ::starts_with/ends_with	P0457R2		t211.dir/_21426o_21a_c16.cpp, t211.dir/_21426o_21a_c32.cpp, t211.dir/_21426o_21a.cpp, t211.dir/_21426o_21a_w.cpp
Library support for operator<=> <compare>	P0768R1		t170.dir/_17p2a.cpp, t183.dir/_18_101o1a.cpp
std::remove_cvref	P0550R2		t202.dir/_20_152o1a.cpp
[[nodiscard]] in the standard library	P0600R1		t212.dir/_2134a_151b_c16.cpp, t212.dir/_2134a_151b_c32.cpp, t212.dir/_2134a_151b_c8.cpp, t212.dir/_2134a_151b.cpp, t212.dir/_2134a_151b_w.cpp
Using std::move in numeric algorithms	P0616R0		t266.dir/_2641Y11a.cpp
Utility to convert a pointer to a raw pointer	P0653R2		negtests/m19.in, t202.dir/_20_104o1a.cpp
Atomic std::shared_ptr and std::weak_ptr	P0718R2		t170.dir/_17p2a.cpp, t204.dir/_20725g_111.cpp, t204.dir/_20725g_130.cpp, t204.dir/_20725g_141.cpp, t204.dir/_20725g_160.cpp, t204.dir/_20725g_181.cpp, t204.dir/_20725g_200.cpp, t204.dir/_20725g_211a2.cpp, t204.dir/_20725g_211a.cpp, t204.dir/_20725g_230.cpp, t204.dir/_20725g_241.cpp, t204.dir/_20725g_251.cpp, t204.dir/_20725g_270.cpp, t204.dir/_20725g_281a.cpp, t204.dir/_20725g_281b.cpp, t204.dir/_20725g_281c.cpp, t204.dir/_20725g_281d.cpp, t204.dir/_20725g_300.cpp, t204.dir/_20725g_301a.cpp, t204.dir/_20725g_301b.cpp, t204.dir/_20725g_301c.cpp, t204.dir/_20725g_301d.cpp, t204.dir/_20725g30.cpp, t204.dir/_20725g_310.cpp, t204.dir/_20725g_331a.cpp, t204.dir/_20725g_331b.cpp, t204.dir/_20725g_331c.cpp, t204.dir/_20725g_331d.cpp, t204.dir/_20725g_341a.cpp, t204.dir/_20725g_341b.cpp, t204.dir/_20725g_341c.cpp, t204.dir/_20725g_341d.cpp, t204.dir/_20725g60.cpp, t204.dir/_20725g71.cpp, t204.dir/_20725g90.cpp
std::span	P0122R7		t170.dir/_17512p1a.cpp
Calendar and timezone	P0355R7		t204.dir/_20_17p1a.cpp
<version>	P0754R2		t180.dir/_1831p1a.cpp
Comparing unordered containers	P0809R0		t225.dir/_2227p1a.cpp
ConstexprIterator requirements	P0858R0		t231.dir/_2331p1a.cpp
std::basic_string::reserve() should not shrink	P0966R1		t170.dir/_17p2a.cpp, t210.dir/_21324p1a_c16.cpp, t210.dir/_21324p1a_c32.cpp, t210.dir/_21324p1a_c8.cpp, t210.dir/_21324p1a.cpp, t210.dir/_21324p1a_w.cpp
Atomic Compare-And-Exchange with padding bits	P0528R3		t290.dir/_2961p1a.cpp
std::atomic_ref	P0019R8		t290.dir/_298p1a.cpp
contains() member function of associative containers, e.g. std::map::contains()	P0458R2		t237.dir/_22444a.cpp
DR: Guaranteed copy elision for piecewise construction	P0475R1		t201.dir/_20_134p1a.cpp
std::bit_cast()	P0476R2		t215.dir/_216p1a.cpp
Integral power-of-2 operations: std::bit_ceil(), std::bit_floor(), std::bit_width(), std::has_single_bit()	P0556R3 P1956R1		t215.dir/_2162p1a.cpp
Improving the return value of erase-like algorithms	P0646R1		t225.dir/_22391p1a.cpp
std::destroying_delete	P0722R3		t182.dir/_18622m1b_s.cpp
std::is_nothrow_convertible	P0758R1		t203.dir/_20_152p1b.cpp
Add std::shift_left/right to <algorithm>	P0769R2		t258.dir/_256_14p1a.cpp
Constexpr for std::swap() and swap related functions	P0879R0		t258.dir/_16443a.cpp, t258.dir/_2573a.cpp, t258.dir/_28711a.cpp, t258.dir/_2872a.cpp, t258.dir/_28771a.cpp, t258.dir/_2877a.cpp
std::type_identity	P0887R1		t203.dir/_20_153a.cpp
Concepts library	P0898R3		t305.dir/_30_31p1a.cpp
constexpr comparison operators for std::array	P1023R0		t225.dir/_2227p1a.cpp
std::unwrap_ref_decay and std::unwrap_reference	P0318R1		t202.dir/_20_43_4a.cpp, t203.dir/_2042p1a.cpp, t203.dir/_20_4_3a.cpp
std::bind_front()	P0356R5		t170.dir/_17_12_14p1a.cpp
std::reference_wrapper for incomplete types	P0357R3		t203.dir/_20_146a.cpp
Fixing operator>>(basic_istream&, CharT*)	P0487R1		t275.dir/_27611Y11.cpp#if, t276.dir/_276123Y0_12.cpp, t276.dir/_276123Y0_12.cpp#if, t276.dir/_276123Y14.cpp#if, t276.dir/_276123Y21.cpp, t276.dir/_276123Y21.cpp#if, t276.dir/_276123Y22.cpp#if, t27k.dir/_2774p1a.cpp
Library support for char8_t	P0482R6		t160.dir/_1723p1a.cpp, t170.dir/_1723p1a.cpp, t170.dir/_17p2a.cpp
Utility functions to implement uses-allocator construction	P0591R4		t203.dir/_20_1082p1a.cpp
DR: std::variant and std::optional should propagate copy/move triviality	P0602R4		t190.dir/_1963p1a.cpp
A sane std::variant converting constructor	P0608R3		t203.dir/_20733p1a.cpp
std::function's move constructor should be noexcept	P0771R1		t202.dir/_20_14_1732a.cpp
The One Ranges Proposal	P0896R4		t305.dir/_30_32p1a.cpp
Heterogeneous lookup for unordered containers	P0919R3 P1690R1		t180.dir/_1854p1a.cpp
<chrono> zero(), min(), and max() should be noexcept	P0972R0		t300.dir/_27_2_1a.cpp
constexpr in std::pointer_traits	P1006R1		t203.dir/_20_1032p1a.cpp

std::assume_aligned()	P1007R3		t200.dir/_20_10p1a.cpp
Smart pointer creation with default initialization (e.g. make_unique_for_overwrite)	P1020R1 P1973R1		t202.dir/_20_11_142a.cpp
Misc constexpr bits	P1032R1		t160.dir/_7_7_1a.cpp, t200.dir/_2042p1a.cpp
Remove comparison operators of std::span	P1085R2		t180.dir/_1872p1a.cpp
Make stateful allocator propagation more consistent for operator+(basic_string)	P1165R1		t170.dir/_17331p1a.cpp
Consistent container erasure, e.g. std::erase(std::vector), or std::erase_if(std::map)	P1209R0 P1115R3		t200.dir/_20p1b.cpp, t215.dir/_2132p1b.cpp
Standard library header units	P1502R1		t021a.dir/_16_5_1_2m.cpp
polymorphic_allocator<> as a vocabulary type	P0339R6		t203.dir/_20_1232a.cpp
std::execution::unseq	P1001R2		t204.dir/_20_18_2a.cpp, t204.dir/_20_182k1a.cpp, t205.dir/_209p1a.cpp, t266.dir/_268_11p1a.cpp
std::lerp() and std::midpoint()	P0811R3		t251.dir/_251016a.cpp
Usability enhancements for std::span	P1024R3		t190.dir/_19536r1a.cpp
DR: Make create_directory() intuitive	P1164R1		t27k.dir/_27_10_154m1a.cpp, t27k.dir/_27_11_14_39p1a.cpp, t27k.dir/_27_11_143p1b.cpp, t27k.dir/_27_11_144p1a.cpp, t27k.dir/_27_11_147o11.cpp, t27k.dir/_27_11_147p1a.cpp, t27k.dir/_27_111o1a.cpp
std::ssize() and unsigned extent for std::span	P1227R2		t180.dir/_187202a.cpp
Traits for (un)bounded arrays	P1357R1		t202.dir/_20_102k11.cpp, t202.dir/_2092g1a_s.cpp
std::to_array()	P0325R4		t244.dir/_248k11a.cpp
Efficient access to std::basic_stringbuf's buffer	P0408R7		t27k.dir/_27_1023o2a.cpp
Layout-compatibility and pointer-interconvertibility traits	P0466R5		t203.dir/_20_157p1a.cpp
Bit operations: std:: rotl(), rotr(), countl_zero(), countl_one(), countr_zero(), countr_one(), popcount()	P0553R4		t260.dir/_2652a.cpp
Mathematical constants	P0631R8		t240.dir/_24r1a.cpp
Text formatting	P0645R10		t200.dir/_201r1e.cpp
std::stop_token and std::jthread	P0660R10		t300.dir/_32321a.cpp
constexpr std::allocator and related utilities	P0784R7		t215.dir/2S9, 681a.cpp
constexpr std::string	P0980R1		t201.dir/_20_12_13k1a.cpp, t210.dir/_2131m1b.cpp
constexpr std::vector	P1004R2		t200.dir/_201r1b.cpp
Input range adaptors	P1035R7		t244.dir/_247101a.cpp, t244.dir/_247121a.cpp, t244.dir/_247141a.cpp, t244.dir/_247151a.cpp, t244.dir/_247161a.cpp, t244.dir/_247171a.cpp, t244.dir/_247181a.cpp, t244.dir/_24721a.cpp, t244.dir/_24781a.cpp, t244.dir/_24791a.cpp
constexpr std::invoke() and related utilities	P1065R2		t201.dir/_20_14_5a.cpp
Atomic waiting and notifying, std::counting_semaphore, std::latch and std::barrier	P1135R6		t290.dir/_321821a.cpp, t290.dir/_32832a.cpp, t290.dir/_32731a.cpp
std::source_location	P1208R6		t160.dir/_161p4b.cpp, t170.dir/_178r1a.cpp, t170.dir/_178r1a.cpp t170.dir/_178r1a.cpp, t170.dir/_178r1a.cpp
Adding <=> to the standard library	P1614R2		t280.dir/_28_1212g20.cpp, t280.dir/_28_1222g20.cpp
constexpr default constructor of std::atomic and std::atomic_flag	P0883R2		t170.dir/_17p2a.cpp
constexpr for numeric algorithms	P1645R1		t205.dir/_2681k09.cpp, t266.dir/_2641Y11a.cpp, t266.dir/_2642Y11a.cpp, t266.dir/_2643Y21a.cpp, t266.dir/_2644Y41a.cpp, t266.dir/_268_10k11a.cpp, t266.dir/_268_12m11.cpp, t266.dir/_2683k11.cpp, t266.dir/_2684k11.cpp, t266.dir/_2687k11.cpp, t266.dir/_2689k11.cpp
Safe integral comparisons	P0586R2		t201.dir/_2027a.cpp

C++17 Library Features

Feature	Paper	Addressed	Test Cases
std::void_t	N3911		negtests/m19.in, t202.dir/_20_1076k1_22.cpp
std::uncaught_exceptions()	N4259		t170.dir/_17p2a.cpp, t170.dir/_17p2a.cpp#include, t183.dir/_1874a11b.cpp, t183.dir/_1874a11.cpp
std::size(), std::empty() and std::data()	N4280		t170.dir/_17512p1a.cpp, t170.dir/_17512p1a.cpp#if, t170.dir/_17512p1a.cpp#include, t180.dir/_1872p1a.cpp, t180.dir/_1872p1a.cpp, t180.dir/_1872p1a.cpp#include, t180.dir/_1872p1a.cppvoid, t190.dir/_19536r1a.cpp, t190.dir/_19536r1a.cpp//, t225.dir/_22731p1a.cpp
Improving std::pair and std::tuple	N4387		negtests/m19.in, t200.dir/_2032k51.cpp, t200.dir/_2042k51.cpp, t202.dir/_20521k1a.cpp, t202.dir/_20521k1b.cpp, t202.dir/_20521k1c.cpp
std::bool_constant	N4389		t202.dir/_20_102k02.cpp
std::shared_mutex (untimed)	N4508		t300.dir/_304141k10.cpp, t300.dir/_304141k_122a_s.cpp, t300.dir/_304141k_122b_s.cpp, t300.dir/_304141k_123a.cpp, t300.dir/_304141k_131a.cpp, t300.dir/_304141k_131b.cpp, t300.dir/_304141k_141a.cpp, t300.dir/_304141k_151.cpp, t300.dir/_304141k22.cpp, t300.dir/_304141k52a_s.cpp, t300.dir/_304141k52b_s.cpp, t300.dir/_304141k52c_s.cpp, t300.dir/_304141k52d_s.cpp, t300.dir/_304141k53a.cpp, t300.dir/_304141k53b.cpp, t300.dir/_304141k61a.cpp, t300.dir/_304141k71a.cpp, t300.dir/_304141k81.cpp, t300.dir/_304151k10.cpp, t300.dir/_304151k_122a_s.cpp, t300.dir/_304151k_122b_s.cpp, t300.dir/_304151k_123a.cpp, t300.dir/_304151k_131a.cpp, t300.dir/_304151k_141a.cpp, t300.dir/_304151k_151.cpp, t300.dir/_304151k21a.cpp, t300.dir/_304151k21b.cpp, t300.dir/_304151k21c.cpp, t300.dir/_304151k21d.cpp, t300.dir/_304151k21h.cpp, t300.dir/_304151k22.cpp, t300.dir/_304151k52a_s.cpp, t300.dir/_304151k52b_s.cpp, t300.dir/_304151k52c_s.cpp, t300.dir/_304151k52d_s.cpp, t300.dir/_304151k53a.cpp, t300.dir/_304151k53b.cpp, t300.dir/_304151k61a.cpp, t300.dir/_304151k71a.cpp, t300.dir/_304151k81.cpp, t300.dir/_30421k01c.cpp, t300.dir/_30421k01d.cpp, t300.dir/_30421k01e.cpp, t300.dir/_30421k01f.cpp, t300.dir/_30421k10e_s.cpp, t300.dir/_30421k10f_s.cpp, t301.dir/_304141k10.cpp, t301.dir/_304141k_122a_s.cpp, t301.dir/_304141k_122b_s.cpp, t301.dir/_304141k_131a.cpp, t301.dir/_304141k_131b.cpp, t301.dir/_304141k_141a.cpp, t301.dir/_304141k_151.cpp, t301.dir/_304141k22.cpp, t301.dir/_304141k52a_s.cpp, t301.dir/_304141k52b_s.cpp, t301.dir/_304141k52c_s.cpp, t301.dir/_304141k52d_s.cpp, t301.dir/_304141k53a.cpp, t301.dir/_304141k53b.cpp, t301.dir/_304141k61a.cpp, t301.dir/_304141k71a.cpp, t301.dir/_304141k81.cpp, t301.dir/_304151k10_x.cpp, t301.dir/_304151k_122a_s_x.cpp, t301.dir/_304151k_122b_s_x.cpp, t301.dir/_304151k_123a_x.cpp, t301.dir/_304151k_131a_x.cpp, t301.dir/_304151k_141a_x.cpp, t301.dir/_304151k21a_x.cpp, t301.dir/_304151k21b_x.cpp, t301.dir/_304151k21c_x.cpp, t301.dir/_304151k21d_x.cpp, t301.dir/_304151k21h_x.cpp, t301.dir/_304151k22_x.cpp, t301.dir/_304151k52a_s_x.cpp, t301.dir/_304151k52b_s_x.cpp, t301.dir/_304151k52c_s_x.cpp, t301.dir/_304151k52d_s_x.cpp, t301.dir/_304151k53a_x.cpp, t301.dir/_304151k53b_x.cpp, t301.dir/_304151k61a_x.cpp, t301.dir/_304151k71a_x.cpp, t301.dir/_30421k01c_s.cpp, t301.dir/_30421k01d_s.cpp, t301.dir/_30421k01e_s.cpp, t301.dir/_30421k01f_s.cpp
Type traits variable templates	P0006R0		t190.dir/_195k2_99.cpp//, t200.dir/_2041k21.cpp, t202.dir/_20_102k11.cpp, t202.dir/_209_10k11.cpp, t203.dir/_20_152k11.cpp, t203.dir/_20771k11.cpp
Logical operator type traits	P0013R1		t201.dir/_20_138k1a.cpp
Standardization of Parallelism TS	P0024R2		t204.dir/_20_182k1a.cpp, t205.dir/_209_12k01_s.cpp, t205.dir/_209_12k02.cpp, t205.dir/_209_12k03.cpp, t205.dir/_209_12k04.cpp, t205.dir/_253k0_100.cpp, t205.dir/_253k0_101.cpp, t205.dir/_253k0_10a.cpp, t205.dir/_253k0_11.cpp, t205.dir/_253k0_12.cpp, t205.dir/_253k0_13.cpp, t205.dir/_253k0_14.cpp, t205.dir/_253k0_15.cpp, t205.dir/_253k0_16a.cpp, t205.dir/_253k0_17b.cpp, t205.dir/_253k0_18.cpp, t205.dir/_253k0_19.cpp, t205.dir/_253k01.cpp, t205.dir/_253k0_20.cpp, t205.dir/_253k0_21.cpp, t205.dir/_253k0_22.cpp, t205.dir/_253k0_23.cpp, t205.dir/_253k0_24.cpp, t205.dir/_253k0_25.cpp, t205.dir/_253k0_26.cpp, t205.dir/_253k0_27.cpp, t205.dir/_253k0_28.cpp, t205.dir/_253k0_29.cpp, t205.dir/_253k02a.cpp, t205.dir/_253k0_30.cpp, t205.dir/_253k0_31.cpp, t205.dir/_253k0_32.cpp, t205.dir/_253k0_33.cpp, t205.dir/_253k0_34.cpp, t205.dir/_253k0_35.cpp, t205.dir/_253k0_36.cpp, t205.dir/_253k0_37.cpp, t205.dir/_253k0_38.cpp, t205.dir/_253k0_39.cpp, t205.dir/_253k03a.cpp, t205.dir/_253k0_40.cpp, t205.dir/_253k0_41.cpp, t205.dir/_253k0_42.cpp, t205.dir/_253k0_43.cpp, t205.dir/_253k0_44.cpp, t205.dir/_253k0_45.cpp, t205.dir/_253k0_46.cpp, t205.dir/_253k0_47.cpp, t205.dir/_253k0_48.cpp, t205.dir/_253k0_49.cpp, t205.dir/_253k04.cpp, t205.dir/_253k0_50.cpp, t205.dir/_253k0_51.cpp, t205.dir/_253k0_52.cpp, t205.dir/_253k0_53.cpp, t205.dir/_253k0_54.cpp, t205.dir/_253k0_55.cpp, t205.dir/_253k0_56.cpp, t205.dir/_253k0_57.cpp, t205.dir/_253k0_58.cpp, t205.dir/_253k0_59.cpp, t205.dir/_253k05b.cpp, t205.dir/_253k0_60.cpp, t205.dir/_253k0_61.cpp, t205.dir/_253k0_62.cpp, t205.dir/_253k0_63.cpp, t205.dir/_253k0_64.cpp, t205.dir/_253k0_65.cpp, t205.dir/_253k0_66.cpp, t205.dir/_253k0_67.cpp, t205.dir/_253k0_68.cpp, t205.dir/_253k0_69.cpp, t205.dir/_253k06.cpp, t205.dir/_253k0_70.cpp, t205.dir/_253k0_71.cpp, t205.dir/_253k0_72.cpp, t205.dir/_253k0_73.cpp, t205.dir/_253k0_74.cpp, t205.dir/_253k0_75.cpp, t205.dir/_253k0_76.cpp, t205.dir/_253k0_77.cpp, t205.dir/_253k0_78.cpp, t205.dir/_253k0_79.cpp, t205.dir/_253k07.cpp, t205.dir/_253k0_80.cpp, t205.dir/_253k0_81.cpp, t205.dir/_253k0_82.cpp, t205.dir/_253k0_83.cpp, t205.dir/_253k0_84.cpp, t205.dir/_253k0_85.cpp, t205.dir/_253k0_86.cpp, t205.dir/_253k0_87.cpp, t205.dir/_253k0_88.cpp, t205.dir/_253k0_89.cpp, t205.dir/_253k08a.cpp, t205.dir/_253k08.cpp, t205.dir/_253k0_90.cpp, t205.dir/_253k0_91.cpp, t205.dir/_253k0_92.cpp, t205.dir/_253k0_93.cpp, t205.dir/_253k0_94.cpp, t205.dir/_253k0_95.cpp, t205.dir/_253k0_96.cpp, t205.dir/_253k0_97.cpp, t205.dir/_253k0_98.cpp, t205.dir/_253k0_99.cpp, t205.dir/_253k09.cpp, t205.dir/_2681k0_10.cpp, t205.dir/_2681k0_11.cpp, t205.dir/_2681k0_13.cpp, t205.dir/_2681k0_14.cpp, t205.dir/_2681k0_15.cpp, t205.dir/_2681k0_16.cpp, t205.dir/_2681k0_17.cpp, t205.dir/_2681k0_18.cpp, t205.dir/_2681k0_19.cpp, t205.dir/_2681k01.cpp, t205.dir/_2681k02.cpp, t205.dir/_2681k03.cpp, t205.dir/_2681k04.cpp, t205.dir/_2681k06.cpp, t205.dir/_2681k07.cpp, t205.dir/_2681k08.cpp, t205.dir/_2681k09.cpp, t251.dir/_253k05a.cpp
std::clamp()	P0025R0		t258.dir/_2558k21.cpp
Hardware interference size	P0154R1		t183.dir/_1864k11.cpp, t183.dir/_1864k11.cpp
(nothrow-)swappable traits	P0185R1		t170.dir/_17p1b.cpp
File system library	P0218R1		t27k.dir/_27_10_10k1a.cpp, t27k.dir/_27_10_12k1a.cpp, t27k.dir/_27_10_13k1a.cpp, t27k.dir/_27_10_14k1a.cpp, t27k.dir/_27_107k1a.cpp, t27k.dir/_27_108k1a.cpp, t27k.dir/_27_108m1a.cpp, t27k.dir/_27_109k1a.cpp, t27k.dir/_27_111o1a.cpp
std::string_view	N3921 P0220R1		negtests/m19.in, t170.dir/_17612kt_14.cpp, t201.dir/_20_12_13k1a.cpp, t201.dir/_20_129k31.cpp, t201.dir/_20_138k1a.cpp, t201.dir/_20_15_87.cpp, t201.dir/_2062k1a.cpp, t201.dir/_2063k1a.cpp, t201.dir/_207k1a.cpp, t201.dir/_2093k1a.cpp, t202.dir/_20525k11.cpp, t216.dir/_214k1a_c16.cpp, t216.dir/_214k1a_c32.cpp, t216.dir/_214k1a_c8.cpp, t216.dir/_214k1a.cpp, t216.dir/_214k1a_w.cpp, t251.dir/_253_13k81.cpp, t251.dir/_254_12k21.cpp

std::any	P0220R1		negtests/m19.in, t170.dir/_17612kt_14.cpp, t201.dir/_20_12_13k1a.cpp, t201.dir/_20_129k31.cpp, t201.dir/_20_138k1a.cpp, t201.dir/_20_15_87.cpp, t201.dir/_2062k1a.cpp, t201.dir/_2063k1a.cpp, t201.dir/_207k1a.cpp, t201.dir/_2093k1a.cpp, t202.dir/_20525k11.cpp, t216.dir/_214k1a_c16.cpp, t216.dir/_214k1a_c32.cpp, t216.dir/_214k1a_c8.cpp, t216.dir/_214k1a.cpp, t216.dir/_214k1a_w.cpp, t251.dir/_253_13k81.cpp, t251.dir/_254_12k21.cpp
std::optional	P0220R1		negtests/m19.in, t170.dir/_17612kt_14.cpp, t201.dir/_20_12_13k1a.cpp, t201.dir/_20_129k31.cpp, t201.dir/_20_138k1a.cpp, t201.dir/_20_15_87.cpp, t201.dir/_2062k1a.cpp, t201.dir/_2063k1a.cpp, t201.dir/_207k1a.cpp, t201.dir/_2093k1a.cpp, t202.dir/_20525k11.cpp, t216.dir/_214k1a_c16.cpp, t216.dir/_214k1a_c32.cpp, t216.dir/_214k1a_c8.cpp, t216.dir/_214k1a.cpp, t216.dir/_214k1a_w.cpp, t251.dir/_253_13k81.cpp, t251.dir/_254_12k21.cpp
Polymorphic memory resources	P0220R1		negtests/m19.in, t170.dir/_17612kt_14.cpp, t201.dir/_20_12_13k1a.cpp, t201.dir/_20_129k31.cpp, t201.dir/_20_138k1a.cpp, t201.dir/_20_15_87.cpp, t201.dir/_2062k1a.cpp, t201.dir/_2063k1a.cpp, t201.dir/_207k1a.cpp, t201.dir/_2093k1a.cpp, t202.dir/_20525k11.cpp, t216.dir/_214k1a_c16.cpp, t216.dir/_214k1a_c32.cpp, t216.dir/_214k1a_c8.cpp, t216.dir/_214k1a.cpp, t216.dir/_214k1a_w.cpp, t251.dir/_253_13k81.cpp, t251.dir/_254_12k21.cpp
Mathematical special functions	P0226R1		t266.dir/_26_10_10k11.cpp, t266.dir/_26_10_11k11.cpp, t266.dir/_26_10_12k11.cpp, t266.dir/_26_10_13k11.cpp, t266.dir/_26_10_14k11.cpp, t266.dir/_26_10_15k11.cpp, t266.dir/_26_10_16k11.cpp, t266.dir/_26_10_17k11.cpp, t266.dir/_26_10_18k11.cpp, t266.dir/_26_10_19k11.cpp, t266.dir/_26_101k11.cpp, t266.dir/_26_10_20k11.cpp, t266.dir/_26_10_21k11.cpp, t266.dir/_26_102k11.cpp, t266.dir/_26_103k11.cpp, t266.dir/_26_104k11.cpp, t266.dir/_26_105k11.cpp, t266.dir/_26_106k11.cpp, t266.dir/_26_107k11.cpp, t266.dir/_26_108k11.cpp, t266.dir/_26_109k11.cpp
C++17 should refer to C11 instead of C99	P0063R3		t200.dir/_20_10_10m1a.cpp, t203.dir/_20_10_11m1a.cpp
Splicing Maps and Sets	P0083R3		t237.dir/_23571m0_99.cpp
std::variant	P0088R3		negtests/m19.in, t203.dir/_2073m1a.cpp
std::make_from_tuple()	P0209R2		t202.dir/_20535m21.cpp
std::has_unique_object_representations	P0258R2		t202.dir/_20_1543mt_42.cpp
std::gcd() and std::lcm()	P0295R0		t266.dir/_268_13m11.cpp, t266.dir/_268_14m11.cpp
std::not_fn	P0005R4 P0358R1		t201.dir/_20_129k31.cpp
Elementary string conversions, including FP (Floating-Point) values support	P0067R5		t202.dir/_2028m1a.cpp, t202.dir/_2029m1a.cpp
std::shared_ptr and std::weak_ptr with array support	P0414R2		
std::scoped_lock	P0156R2		t300.dir/_304141k_123a.cpp, t301.dir/_304141k10.cpp, t301.dir/_304141k_122a_s.cpp, t301.dir/_304141k_122b_s.cpp, t301.dir/_304141k_131a.cpp, t301.dir/_304141k_131b.cpp, t301.dir/_304141k_141a.cpp, t301.dir/_304141k_151.cpp, t301.dir/_304141k22.cpp, t301.dir/_304141k52a_s.cpp, t301.dir/_304141k52b_s.cpp, t301.dir/_304141k52c_s.cpp, t301.dir/_304141k52d_s.cpp, t301.dir/_304141k53a.cpp, t301.dir/_304141k53b.cpp, t301.dir/_304141k61a.cpp, t301.dir/_304141k71a.cpp, t301.dir/_304141k81.cpp, t301.dir/_304151k10_x.cpp, t301.dir/_304151k_122a_s_x.cpp, t301.dir/_304151k_122b_s_x.cpp, t301.dir/_304151k_123a_x.cpp, t301.dir/_304151k_131a_x.cpp, t301.dir/_304151k_141a_x.cpp, t301.dir/_304151k21a_x.cpp, t301.dir/_304151k21b_x.cpp, t301.dir/_304151k21c_x.cpp, t301.dir/_304151k21d_x.cpp, t301.dir/_304151k21h_x.cpp, t301.dir/_304151k22_x.cpp, t301.dir/_304151k52a_s_x.cpp, t301.dir/_304151k52b_s_x.cpp, t301.dir/_304151k52c_s_x.cpp, t301.dir/_304151k52d_s_x.cpp, t301.dir/_304151k53a_x.cpp, t301.dir/_304151k53b_x.cpp, t301.dir/_304151k61a_x.cpp, t301.dir/_304151k71a_x.cpp, t301.dir/_30421k01c_s.cpp, t301.dir/_30421k01d_s.cpp, t301.dir/_30421k01e_s.cpp, t301.dir/_30421k01f_s.cpp, t301.dir/_30421k31c_s.cpp, t301.dir/_30442m01a.cpp, t301.dir/_30442m01b.cpp, t301.dir/_30442m10a_s.cpp, t301.dir/_30442m10b_s.cpp, t301.dir/_30442m10c_s.cpp, t301.dir/_30442m10d_s.cpp, t301.dir/_30442m20.cpp, t301.dir/_30442m31.cpp, t301.dir/_30442m52.cpp, t301.dir/_30442m81.cpp
std::byte	P0298R3		t180.dir/_1821m1a.cpp
std::is_aggregate	LWG2911		t202.dir/_20_152mt_42_s.cpp

C++14 Library Features

Feature	Paper	Addressed	Notes
constexpr for <complex>	N3302		t260.dir/_2661i1a.cpp
Transparent operator functors	N3421		t202.dir/_2094i81a.cpp
std::result_of and SFINAE	N3462		t202.dir/_20_1076i09.cpp
constexpr for <chrono>	N3469		t203.dir/_20_122i1a.cpp
constexpr for <array>	N3470		t230.dir/_23321i1b.cpp
constexpr for <initializer_list>, <utility> and <tu	N3471		negtests/m19.in, t200.dir/_2022i1_s.cpp, t200.dir/_2023i18a.cpp, t200.dir/_2023i18b.cpp, t200.dir/_2023i18c.cpp, t200.dir/_2023i1a.cpp, t200.dir/_2032i21a.cpp, t200.dir/_2032i21b.cpp, t200.dir/_2032i21.cpp, t202.dir/_20427i1a.cpp
Improved std::integral_constant	N3545		t202.dir/_20_103i07.cpp
User-defined literals for <chrono> and <string>	N3642		t203.dir/_20_1258i1a.cpp, t217.dir/_217i1a.cpp
Null forward iterators	N3644		
std::quoted	N3654		t27e.dir/_2776i1a_s.cpp
std::make_unique	N3656		t203.dir/_20814i11.cpp
Heterogeneous associative lookup	N3657		t203.dir/_2094i81b.cpp
std::integer_sequence	N3658		t202.dir/_2051i01.cpp
std::shared_timed_mutex	N3659		t300.dir/_304141k10.cpp, t300.dir/_304141k_122a_s.cpp, t300.dir/_304141k_122b_s.cpp, t300.dir/_304141k_123a.cpp, t300.dir/_304141k_131a.cpp, t300.dir/_304141k_131b.cpp, t300.dir/_304141k_141a.cpp, t300.dir/_304141k_151.cpp, t300.dir/_304141k22.cpp, t300.dir/_304141k52a_s.cpp, t300.dir/_304141k52b_s.cpp, t300.dir/_304141k52c_s.cpp, t300.dir/_304141k52d_s.cpp, t300.dir/_304141k53a.cpp, t300.dir/_304141k53b.cpp, t300.dir/_304141k61a.cpp, t300.dir/_304141k71a.cpp, t300.dir/_304141k81.cpp, t300.dir/_304151k10.cpp, t300.dir/_304151k_122a_s.cpp, t300.dir/_304151k_122b_s.cpp, t300.dir/_304151k_123a.cpp, t300.dir/_304151k_131a.cpp, t300.dir/_304151k_141a.cpp, t300.dir/_304151k_151.cpp, t300.dir/_304151k21a.cpp, t300.dir/_304151k21b.cpp, t300.dir/_304151k21c.cpp, t300.dir/_304151k21d.cpp, t300.dir/_304151k21h.cpp, t300.dir/_304151k22.cpp, t300.dir/_304151k52a_s.cpp, t300.dir/_304151k52b_s.cpp, t300.dir/_304151k52c_s.cpp, t300.dir/_304151k52d_s.cpp, t300.dir/_304151k53a.cpp, t300.dir/_304151k53b.cpp, t300.dir/_304151k61a.cpp, t300.dir/_304151k71a.cpp, t300.dir/_304151k81.cpp, t300.dir/_30421k01c.cpp, t300.dir/_30421k01d.cpp, t300.dir/_30421k01e.cpp, t300.dir/_30421k01f.cpp, t300.dir/_30421k10e_s.cpp, t300.dir/_30421k10f_s.cpp, t301.dir/_304141k10.cpp, t301.dir/_304141k_122a_s.cpp, t301.dir/_304141k_122b_s.cpp, t301.dir/_304141k_131a.cpp, t301.dir/_304141k_131b.cpp, t301.dir/_304141k_141a.cpp, t301.dir/_304141k_151.cpp, t301.dir/_304141k22.cpp, t301.dir/_304141k52a_s.cpp, t301.dir/_304141k52b_s.cpp, t301.dir/_304141k52c_s.cpp, t301.dir/_304141k52d_s.cpp, t301.dir/_304141k53a.cpp, t301.dir/_304141k53b.cpp, t301.dir/_304141k61a.cpp, t301.dir/_304141k71a.cpp, t301.dir/_304141k81.cpp, t301.dir/_304151k10_x.cpp, t301.dir/_304151k_122a_s_x.cpp, t301.dir/_304151k_122b_s_x.cpp, t301.dir/_304151k_123a_x.cpp, t301.dir/_304151k_131a_x.cpp, t301.dir/_304151k_141a_x.cpp, t301.dir/_304151k21a_x.cpp, t301.dir/_304151k21b_x.cpp, t301.dir/_304151k21c_x.cpp, t301.dir/_304151k21d_x.cpp, t301.dir/_304151k21h_x.cpp, t301.dir/_304151k22_x.cpp, t301.dir/_304151k52a_s_x.cpp, t301.dir/_304151k52b_s_x.cpp, t301.dir/_304151k52c_s_x.cpp, t301.dir/_304151k52d_s_x.cpp, t301.dir/_304151k53a_x.cpp, t301.dir/_304151k53b_x.cpp, t301.dir/_304151k61a_x.cpp, t301.dir/_304151k71a_x.cpp, t301.dir/_30421k01c_s.cpp, t301.dir/_30421k01d_s.cpp, t301.dir/_30421k01e_s.cpp, t301.dir/_30421k01f_s.cpp
std::exchange	N3668		t200.dir/_2023i11.cpp
fixing constexpr member functions without cons	N3669		t202.dir/_2062i_381.cpp, t202.dir/_2092g1a_s.cpp//, t230.dir/_23321i1a.cpp, t260.dir/_2661i1a.cpp
std::get<T>()	N3670		negtests/m19.in, t200.dir/_2032i21a.cpp, t200.dir/_2032i21b.cpp, t200.dir/_2032i21.cpp, t202.dir/_20426i_101a.cpp, t202.dir/_20426i_101.cpp, t202.dir/_20426i81a.cpp, t202.dir/_20426i81.cpp
Dual-Range std::equal, std::is_permutation, std	N3671		t251.dir/_252_12g11a.cpp, t251.dir/_252_12g11b.cpp, t251.dir/_252_12i1a.cpp, t251.dir/_252_12i1b.cpp

C++ 11

Feature	Paper	Addressed	C++ standard	Comments
Type Traits				
Garbage Collection and Reachability-Based Leak Detection (library support)	N1836 N2240 N2244 N2255 N2342 N2984 N3142			
Money, Time and hex/float manipulators	N2071 N2072			
Disallowing COW (copy on write) string	N2688			

There is a new script `runtest.sh(.bat)` which given a test identifier will find that file in the source directory and execute just that test. It is useful for debugging test cases. Here is an example of usage in Visual Studio:

9_2_7a.cpp

3kp1b0n50fj_szpj2xxvdgyw0000gn > T > ch.sudo.cyberduck > 075b1092-7ca4-43d5-9a47-bb5e1a094393 > home > doug > PlumHall > xvs22a > conform > t09d.dir > 9_2_7a.cpp

```cpp
/* 9_2_7a 9.2.7 The constinit specifier. - implements P1143R2 - CXX20 */

#if defined(SKIP9_2_7a)
#elif defined(DISALLOW_CXX17)
#define SKIP9_2_7a  _CXX17
#endif

#if (!defined(SKIP9_2_7a)&&!defined(SKIP7)&&!defined(ONLY))||defined(CASE9_2_7a)

    extern thread_local constinit int x_;
    const char * f_() { return "dynamic init"; }    // ill-formed
    constexpr const char *g_(bool p) { return p ? "constant init" : f_(); }
    int h_() { return x_; }                         // no check of a guard variable required

    constinit const char *c_ = g_(true);

    constexpr int constexprVal = 1000;
    constinit int constinitVal = 1000;

    int incr_tinkr(int val){ return ++val;}

#endif /* CASE9_2_7a */
```

PROBLEMS TERMINAL OUTPUT DEBUG CONSOLE bash - douglasteeple

```
doug@doug:~/PlumHall/xvs22a-gcc-c20$ runtest.sh 9_2_7a
Found the conform directory t09d.dir
Found a B-File 9_2_7a_b.cpp
9_2_7a:
===== 9_2_7a
***** Reached first test *****
9_2_7a (><) passed
#PASSED:  9_2_7a
***** 5 individual successful items in 9_2_7a *****
***** 1 successful test case in 9_2_7a *****
***** 0 errors detected in 9_2_7a *****
***** 0 skipped sections in 9_2_7a *****
doug@doug:~/PlumHall/xvs22a-gcc-c20$
```

Please let me know your thoughts and suggestions: doug@plumhall2b.com.

Historical Versions

New in lvs19a:

Each subdirectory, such as t170.dir, provided one large file, such as t170.cpp, which collected together all or most of the tests in this section. We have dropped support for the LibSuite++ feature of providing files such as t170.cpp. We have reluctantly concluded that we have no way of implementing this feature.

New in lvs18a:

In `flags.h`, we have added a new choice: choose between CXX03, CXX11, CXX14, CXX17, and CXXWP ("working paper"). Contrary to our expectations a few years ago, we have to maintain CXX17 and CXXWP, because there are changes that only apply to CXXWP.

New in lvs17a:

The requirements of ISO/IEC editors have caused the chapters ("clauses") in the C++ Standard to be re-numbered. What was originally clause 17 is now clause 20, etc. Fortunately, the offset is a constant (three).

For now, only the members of the C++ standards committee are affected by this change, but eventually everyone will see this offset.

Plum Hall has not changed the testcase numbering system; those of you consulting the most recent drafts will need to subtract three from the clause number.

WG21 has changed the name `result_of` to `invoke_result` (and `result_of_t` to `invoke_result_t`). The new testcases (the "m" cases) use the new name, but we have otherwise changed the name in "`flags.h`".

In view of the complications running `dst-win/buildmax.bat`, Plum Hall .has put `t007` last in `buildmax.bat.`

New in lvs16a:

We have placed all of the 2016 target-dirs into a folder called "`ph16`".

In `flags.h`, we have added a new choice: choose between CXX03, CXX11, CXX14, and CXXWP ("working paper"). In a few years, CXXWP will become CXX17.

In the various `makefiles`, we have added the section number (for documentation only), just so you can see it as the commands go past on the screen. Also, `buildmax[.bat]` will build the `t007.dir` tests first. The Unix/Linux scripts in dst-ix seem to work fine, but the Windows scripts may require some manual intervention. Here at Plum Hall we wait a minute or two before concluding that a test is hung, kill the batch file, and (in a separate window) edit the batch file to "remark" the lines that passed already. Sometimes, six or seven iterations are needed to complete the `t007.dir` tests.

In `dst-win`, the `envsuite.bat` file calls a separate `compiler-setup.bat` file.

Approximately 180 pages of new specifications were added during the Jacksonville 2018 meeting: Special Math, Library Fundamentals, Parallelism, and File System.

New in lvs15a:

This release, lvs15a, implements all of C++14.

All the negtests (from whatever clause they originally appear in) have been put into `m19.in`.

New in lvs14a:

If there are no "dots" in the filename, the `txtchk` command will expect to find its checksums in a "`.txtchk`" file, so it can now be invoked as simply

```
txtchk  -f  lvs18a
```

The lvs14a release is mostly a bug-fix release; no new cases have been added above lvs13a. In "`flags.h`" there is now a choice between CXX03, CXX11, and CXX14 (but don't use CXX14 yet).

We have revised the scoring method in `conform/util.c:report()`, so that for the "big file" ("`tnnn.cpp`"), it reports only one testcase: either there were one or more cases skipped (recorded as a SKIP), one or more cases failed (recorded as a FAIL), or all cases passed (recorded as a PASS).

We have added a new `make-summary` command, which will produce the appropriate `.sum` file. Also, we re-named `buildboth-all` to `buildmax`.

The testcases in `t007.dir` are all "difficult" in one way or another; they try to exhaust all free space or are prone to run forever on multi-threading failures, etc. In the `dst-win` (Windows) environment, each execution will be prefixed with a "`time /T`" command which will at least permit visual inspection of the elapsed time. One must manually kill the batch file, manually edit it, and start it running again. In the `dst-ix` (Linux/UNIX) environment, we can do slightly better: Each testcase is started as a background task. Then after a configurable elapsed time, the `lvsclgo2` script kills the background task, and if it was still running, we add it (with a times-ten configurable elapsed time) to a `build-again` script.

Using these methods, we have obtained a successful alpha-testing of each testcase in `t007.dir` (except for one exceptionally-difficult case).

Plum Hall Inc., 67-1185 Mamalahoa Hwy #D104, PMB #372 Kamuela HI 96743

1.: Historical Overview

NOTE: SOME OF THE INFORMATION HERE IS RETAINED AS A HISTORICAL REFERENCE.

For example, while CXX03 may be referenced, it is no longer supported on Windows because Microsoft's cl compiler only supports versions as far back as CXX14, the suites do not support any version prior to CXX11.

LibSuite++®, the Plum Hall Validation Suite for C++, is a set of C++ programs for testing and evaluating a C++ library implementation.

This manual will explain how each section of the suite works, how to configure the tests for your system, and what assumptions are made about previous sections. The examples will illustrate the use of ***LibSuite++***, and also demonstrate how some of the sections work.

1.1 License

Please take the time to read the license that your organization has signed. It is a legal document, and the restrictions apply to any persons using the product.

Here is a brief summary:

- You may use ***LibSuite++*** on any machine within a 2-mile radius of your Designated Site.
- Your Management Contact person, or anyone designated by the Management Contact, may call Plum Hall for consultation and advice.
- You need to notify us if you designate a new Management Contact, or plan to change your Designated Site, or plan to change your company's name.
- ***LibSuite++*** is proprietary, confidential, copyrighted software. You must protect its confidentiality with the same procedures you use to protect your own company's confidential information.
- You may not disclose the detailed results of running ***LibSuite++***, except as permitted in the License.
- You may not take any form of copies of ***LibSuite++*** away from the Designated Site.

1.2 Technical Overview

This distribution of ***LibSuite++*** covers Chapters 17 through 30, and Normative Annex D, of the Standard for C++. The normative tests of ***LibSuite++*** are found underneath one directory named **`CONFORM`**; these are positive tests for basic conformance with the Standard. (This section provides coverage for C++ analogous to the **`LIB`** tests of the Plum Hall C Validation Suite.)

Tools for use in different "destination" directories are provided in the directory trees named **`dst-win`** and **`dst-ix`**. Each of these contains a subtree that matches the structure of the source directories in **`CONFORM`**. Each subtree contains subdirectories for the various specialized tests of ***LibSuite++***, named **`t170.dir`**, **`t180.dir`**, etc. Thus, the components of ***LibSuite++*** are arranged in a directory tree something like this:

```
            |
   +---------+------------+------------+
   |         |            |            |
conform    dst-win                   dst-ix
   |         |            |            |
```

```
  ...     conform        ...          ...
             |
  +----------+------------+--------+
  |          |            |        |
t170.dir  t180.dir ... t27q.dir  txd2.dir
```

All the configurable files are now found in the "destination root" directory. Your compile scripts need to use **$PHDST** (or **%PHDST%**) in their search-path for header files in order for the compiler to find the configurable headers.

Also, we define the compiler's name as **$PHCC** (or **%PHCC%**) in the **envsuite** scripts. Configure this to the name of your compiler's executable file (e.g. **mycc**).

A useful feature of ***LibSuite*++** allows you to record the reasons for each compile-time skipped case failure. In your **flags.h** file, you can add a definition to some compile-time flags, such as

```
#define SKIP525Y1_11 our parser error
#define FAIL_261Y11  Plum Hall bug?
```

Once you've categorized your skips and fails in this way, the strings you defined will show up in the execution output, something like this:

```
#SKIPPED 525Y1_11 (>our parser error<)
#FAILED _261Y11  (>Plum Hall bug?<)
```

And the "unexpected" skips and fails will show up with the distinctive string **"(><)"** attached to each "unexpected" skip or fail. This makes it much easier to re-run the test suite after you've made compiler changes, because you can quickly search for the **"><"** string in the output to see if any new failures have appeared.

You probably will need to put some **SKIP** flags into your **flags.h** file to skip test cases that prevent you from building and executing the **CONFORM** programs.

In ***LibSuite*++**, we have also provided a simpler way of determining the **SKIP** and **FAIL** flags. Each subdirectory, such as **t170.dir**, but the subdirectory also provides individual files, **_17Y11.cpp**, **_17Y12.cpp**, etc., each of which contains only one specific test case. Therefore, you can compile and run the smaller files individually. Each subdirectory contains a **build** script that performs this logic automatically.

We define the compiler's name as **$PHCC** (or **%PHCC%**) in the **envsuite** scripts. Configure this to the name of your compiler's executable file (e.g. **mycc**).

You can record the reasons for each compile-time skipped case or run-time failure. In your **flags.h** file you can add a definition to some compile-time flags, such as

```
#define SKIP_171Y1_11 our parser error
#define FAIL_261Y11  Plum Hall bug?
```

Once you've categorized your skips and fails in this way, the strings you defined will show up in the execution output, something like this:

```
#SKIPPED _171Y1_11 (>our parser error<)
#FAILED _261Y11  (>Plum Hall bug?<)
```

And the "unexpected" skips and fails will show up with the distinctive string **"(><)"** attached to each "unexpected" skip or fail. This makes it much easier to re-run the test suite after you've made compiler changes, because you can quickly search for the **"><"** string in the output to see if any new failures have appeared.

2.: Configuration

2.1 What You Need to Know and Do

In order to install and run ***LibSuite++***, there are several things you need to know, and several things you need to be able to do. If you don't have this knowledge yourself, then you need to locate someone who knows these things and is able to provide you with the information.

- You need to know how to use a text editor on each system you will be using.
- You need to know the basics of how to write and execute "script" (or "batch") files on each system.
- You need to know how much free disk space is available on each system. Fifty megabytes (50 MB) is often enough, if you remove each executable file after gathering its output. If you have less, refer to the Resources section later in this chapter for details.
- You need to know some C++ programming, to customize certain files and to understand the general meaning of the compiler diagnostics that may be produced by some of the nastier test cases.
- You need to know which compiler and library you are supposed to test, and what commands, arguments, environment settings, etc., are needed in order to invoke the compiler you're testing. (The compiler you're testing is called the target compiler.) You may also need to use a different compiler to compile the tool programs themselves. This is known as the host compiler, and it may have its own commands, arguments, environment settings, etc.
- Similarly, you need to know how to invoke the target linker and the host linker, to link the object-files produced by the compilers together with the appropriate libraries.
- Once the target compiler and target linker have produced an executable program to be tested, you need to know how to execute that executable program. On some systems this is almost trivial; on others it involves downloading from one machine to another, capturing output, networking the output back to the host machine, etc.

2.2 Running LibSuite++

There are many different modes in which you can use the Plum Hall Suites:

- Script (or "batch"') command files for compiler, linker, etc, or "line-by-line" individual commands.
- Host compiling (host and target compiler are the same), or cross compiling (host and target are different).
- UNIX platform, or Windows platform, or some other platform.

We have packaged the ***LibSuite++*** so that any set of these choices can be chosen.

2.3 Scripts

Using scripts (or "batch"' files) for compiler, linker, etc., simplifies many aspects of running the suite in varying environments. For example, many QA departments will need to routinely re-execute ***LibSuite++*** using dozens of different compiler flags and options. Using an unchanging set of compiler scripts, and just changing the flags and options in one script, or just setting the flags into environment variables, allows routine re-running of ***LibSuite++***.

In ***LibSuite++***. there is only one script to perform compile-link-and-go:

lvsclgo *pgm* [*output-file-name*] [**bfile**]

Compile *pgm*, taking source and headers from the appropriate directories. Put diagnostic messages into *pgm.clg.* Put output into *output-file-name,* if specified, otherwise send output to standard output. If the third argument is *bfile*, *pgm.cpp* will be linked with *pgm_b.cpp*.

envsuite

The envsuite script requires hand-configuration of environment variables for host and target compilers. You must examine it line-by-line. Here are a few of the environment variables it defines:

`PHCC`	the name of the target compiler
`PHCFLAGS`	compiler flags (for target compiler)
`OLDPATH`	original value of PATH variable before starting
`PATH`	command search path, including compilers, linkers, etc.

UNIX CONSIDERATIONS

If you are on a UNIX platform, you may need to execute the **chmodall** script:

```
sh chmodall
```

in order to mark all your script files as executable files. (It can't hurt, whether needed or not.)

DOS CONSIDERATIONS

The scripts and makefiles need three commands which are common on UNIX but not standard on Windows: **cat, rm** and **cp**. We have written work-alike C source files named **phcat.c** (for ``Plum Hall cat"), **phcp.c** (for ``Plum Hall cp"), and **phrm.c** (for "Plum Hall rm"). The **makefile** in **dst.1** and **dst.2** will compile these to produce exe files (**phcat.exe, phcp.exe, phrm.exe**). After building each of these exe files, the **makefile** invokes a "setup" script (**setup-cat.bat, setup-cp.bat, setup-rm.bat**). Using "cat" as an example, the setup script determines whether a command named **cat** is already available on this system. If not, it copies **phcat.exe** to be named **cat.exe**, so that any further invocation of **cat** will invoke this exe file.

2.4 buildmax

When you have configured for your choices of environment, you should be ready to run all the tests

The **buildmax** command runs **lvsclgo** upon each of the source files in the **conform** directory.

The **buildmax** command also builds the summary files (`.sum`,`.det`,`.html` files), using the appropriate file of expected results (`.exp` file).

Besides the scripts, you will need to configure these other files that are in the destination directory:

flags.h	configurable parameters, including **SKIP** and **FAIL** flags
hocompil.h	characteristics of host-compiler (if different from target compiler)

homachin.h	characteristics of host-machine (if different from target machine)
hodefs.h	flags for hosted compilation (if different from **defs.h**)

SETTING THE ENVSUITE ENVIRONMENT

Each time begin a testing session it is important to "source"' the **envsuite** script to establish all the necessary environment variables. This operation exports the environment variables into your interactive shell.

You do this in different ways depending on your host system's command processor or "shell":

MS-DOS	simply type **envsuite**.
Bourne shell	use the "dot"' command: **". ./envsuite"**

2.5 Installing A Release

We try to accommodate our customers' wide variety of environments, operating systems, and purposes for the suite. Also, we try to use update procedures which will be reasonably efficient for those who make no changes to the distributed Suite, while still being flexible enough for those of you who make local changes.

Some of you are primarily interested in the quality assurance process of running the suites, exactly as distributed, in a reliable fashion that takes a minimum of your time. Others of you are developing compilers that change daily, tracking the latest Standard, with numerous local changes and **SKIP** flags to accommodate unimplemented features.

We always welcome ideas and suggestions for improvement, so please let us know if you see a better way of doing something.

Minimal and Complete Installation Choice

The original packaging of ***LibSuite***++ contained only a few dozen source files. Although the small number of compilations was convenient, the downside was the iterative manual process of determining the **SKIP** flags for the **flags.h** header, to skip language features yet unimplemented in the compiler.

We provide an alternative packaging of ***LibSuite***++ to include "small" files, each containing one test. This packaging is described later in this manual. The hundred or so new subdirectories under conform each contain a traditional "big" file, as well as corresponding "small" files. Each directory's **build** script builds the "standalone" cases, then tries to build the "big" file. Then, if trying to build the "big" file fails, it tries to build each of the small files. Thus, you get full test results on the first run through the suite, with no manual setting of **SKIP** flags required.

Installing the Distribution

Most importantly, install to an empty directory. Installing over the old directory structure will cause no end of chaos. (Also, removing the prior release will help you fulfill your license requirement to maintain source-control of previous versions.)

Verifying Your Files

No matter which method you used for updating—diskette, tape, or patch from diffs—you can check your resulting updated files by compiling the **txtchk** program, and then using it to test the checksum of all your file contents:

```
cd ~/PlumHall/lvs22a-gcc-c20  (or whatever your source root is named)
txtchk -f lvs22a
```

3.: CONFORM

The CONFORM section provides thousands of C++ programs, each covering part of a clause in the Library section of the Standard:

t170 *Clause 17;*
t180 *First part of clause 18;*
t181 *Second part of clause 18;*
etc.

Each program writes a report to its **.out** file in a form very similar to the output of the **LANG** program in the C Suite. That is, **t170** reports that it has executed the first test with the output

```
***** Reached first test *****
```

t170 reports errors using messages of the form

```
ERROR in t170 at line 656: (4) != (5)
```

and prints a summary of the form

```
***** 18 individual successful items in t170
***** 11 successful tests in t170
***** 0 errors detected in t170
***** 0 skipped sections in t170
```

An "individual successful item" is the successful outcome of one individual test function (**ieq**, **chk**, etc.). A "successful test" is the completion of a **begin_case-end_case** sequence with no errors in its individual items.

3.1 Compiling and Executing LibSuite++ CONFORM

In the distribution, C++ source files have a **.cpp** extension and headers have a **.h** extension. The **.cpp** files may be renamed to, say, **.cxx** files to suit your compiler, but the **.h** files should not be renamed.

The **buildmax** script specifies all the steps for building and executing each program. Or you can create the executables by invoking your compiler and linker directly from a command line.

For example, to create the executable for **t170**, compile and link the following files: **t170.cpp** and **util.c**. The compiler command line is typically of the form

```
CC -ot170 t170.cpp util.c
```

where **CC** is the C++ compiler command, **t170.cpp** supplies the main function for the program, and **util.c** contains utility functions used in the test cases.

Alternatively, if in your **flags.h** file you place a definition like

```
#define UTIL_SHOULD_BE_INCLUDED
```

then the compilation will **#include "util.c"** and you need not link with it. This simplifies the compilation process somewhat, and if the compiler supports precompiled headers there is not much overhead in the method.

3.2 Selective Enabling/Disabling with flags.h

In the destination directory, you should create a file named **flags.h**. This header is **#included** by each ***LibSuite++*** file, so that you can record specific enable/disable flags for the tests being made in this directory tree.

If your C++ compiler cannot compile a particular test case, you can use a **SKIP** flag to disable that case. For example, to prevent compilation of test case **_17312Y21** in **t170.c**, add

```
#define SKIP_17312Y21   because some reason
```

to the file **flags.h**. Then recompile and relink **t170**. The line

```
#SKIPPED: _17312Y21 (>because some reason<)
```

will appear in the output when you execute **t170**. The total number of skipped cases appears at the end of the output.

You can also define **DISALLOW** flags in **flags.h** to globally disable certain language features that your compiler may not be able to handle.

For example

```
#define DISALLOW_MEMBER_TEMPLATES
```

compiles alternative code for some cases to accommodate the absence of member templates. See the **flags.h** file for description of each flag.

Similarly, each Library issue in C++17 status has at least one test case for the specified behavior. Using

```
#define DISALLOW_CXX17
```

causes the set of all those test cases to be disabled.

The "disputed cases", described in the following section, are excluded by default.

3.3 Controversial Cases

We strive to make ***LibSuite++*** test the C++ library as commonly understood by the worldwide C++ community. The purpose of the ongoing standard is to capture that understanding. However, there will probably always be specific issues in the language which evoke differing interpretations, and hence there will probably be specific tests in ***LibSuite++*** which evoke differing opinions from ***LibSuite++*** users about the expected results. ***LibSuite++*** accommodates controversial tests in the category of "disputed cases", which are disabled by

```
#define DISALLOW_DISPUTED
```

A "disputed case" is a test for which some ***LibSuite++*** users have expressed the view that, although the test reflects the words in the Standard, the words in the Standard do not reflect common practice, or that the words in the Standard are in the process of being revised within the C++ committee.

We hope to eventually resolve all the disputed cases and convert them to agreeable tests in the **CONFORM** sections of ***LibSuite++***. We welcome your feedback regarding our judgments in these areas.

3.4 Running the CONFORM Programs

Once configuration is completed, you are ready to compile and execute the **CONFORM** programs. Any compile errors reported may represent currently-unimplemented syntactic features, or bugs in your compiler, or bugs in ***LibSuite++***. Or, don't forget, sometimes a compile error means that the compiler wasn't properly installed, or that you weren't told the proper command-line options to use, or that the compilation environment wasn't properly set up. You have to investigate all these possibilities.

If you are unable to trace the cause of any compile errors whilst building **CONFORM**, you should telephone, fax or email Plum Hall for assistance.

Some testcases in `t007.dir` will intentionally keep allocating until the heap is exhausted, in order to test the out-of-memory responses; see SR01131. We have not found any portable way to reduce the time that this takes. On some Unix/Linux systems you may be able to use **ulimit –v nnnn** and/or **ulimit –d nnnn** to reduce the heap size. In some cases, you can reduce the heap size in a virtual machine. If you are really lucky, the allocation library can accept a request to pretend that heap is exhausted. We are interested in hearing any suggestions.

Users have asked us to help speed up testing on parallel multiprocessor systems. We have provided a **makefile** in the **dst*/conform** folder, which can be executed with

```
make -k --jobs N all
```

so that users with N processors should see an N-fold speedup.

When you have tried all the components of the CONFORM section (or at any time you like), you can compare your obtained results against the expected results by executing, in the destination root directory, the command `make-summary`.