ComPPare

1.0.0

Generated by Doxygen 1.14.0

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Concept Index

2.1 Concepts

Here is a list of all concepts with brief descriptions:

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Class Index

4.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Namespace Documentation

6.1 comppare Namespace Reference

ComPPare framework main namespace.

Namespaces

- namespace internal
- namespace plugin

Classes

- class config
- class current_state
- class InputContext

  InputContext class template to hold input parameters for the comparison framework.
- struct spec
- struct spec< spec< Value, Policy >, void >
- struct **spec< Value, Policy >**
- struct spec< Value, void >

Concepts

- concept OutSpec

Typedefs

- template<typename Value, typename Policy>
  using set_policy = spec<Value, Policy>

Functions

- template<typename T>
  void DoNotOptimize (T const &value)

  Prevents the compiler from optimizing away the given value.
- template<typename T>
  void DoNotOptimize (T &value)

  Prevents the compiler from optimizing away the given value.
- template<typename T>
  void DoNotOptimize (T &&value)

  Prevents the compiler from optimizing away the given value.
- void ClobberMemory ()

6.1.1 Detailed Description

ComPPare framework main namespace.

6.1.2 Typedef Documentation

6.1.2.1 set_policy

```
template<typename Value, typename Policy>
using comppare::set_policy = spec<Value, Policy>
```

6.1.3 Function Documentation

6.1.3.1 ClobberMemory()

```
void comppare::ClobberMemory ()  [inline]
```

6.1.3.2 DoNotOptimize() [1/3]

```
template<typename T>
void comppare::DoNotOptimize (
            T && value)  [inline]
```

Prevents the compiler from optimizing away the given value.

Template Parameters

T	The type of the value to protect from optimization.

Parameters

value	The value to protect from optimization.

This implementation is verbatim from Google Benchmark's benchmark::DoNotOptimize(), licensed under Apache2.0. No changes have been made.

6.1.3.3 DoNotOptimize() [2/3]

```
template<typename T>
void comppare::DoNotOptimize (
             T & value)  [inline]
```

Prevents the compiler from optimizing away the given value.

Template Parameters

T	The type of the value to protect from optimization.

Parameters

value	The value to protect from optimization.

This implementation is verbatim from Google Benchmark's benchmark::DoNotOptimize(), licensed under Apache2.0. No changes have been made.

6.1.3.4 DoNotOptimize() [3/3]

```
template<typename T>
void comppare::DoNotOptimize (
            T const & value)  [inline]
```

Prevents the compiler from optimizing away the given value.

Template Parameters

T	The type of the value to protect from optimization.

Parameters

value	The value to protect from optimization.

This implementation is verbatim from Google Benchmark's benchmark::DoNotOptimize(), licensed under Apache2.0. No changes have been made.

6.2 comppare::internal Namespace Reference

Namespaces

- namespace ansi
- namespace concepts
- namespace helper
- namespace policy

6.3 comppare::internal::ansi Namespace Reference

Classes

- class AnsiWrapper

Typedefs

- template<comppare::internal::concepts::Streamable T>
  using AW = AnsiWrapper<std::decay_t<T>>

Functions

- ANSI_DEFINE (RESET, "0", "0")
- ANSI_DEFINE (BOLD, "1", "22")
- ANSI_DEFINE (DIM, "2", "22")
- ANSI_DEFINE (ITALIC, "3", "23")
- ANSI_DEFINE (UNDERLINE, "4", "24")
- ANSI_DEFINE (BLINK, "5", "25")
- ANSI_DEFINE (REVERSE, "7", "27")
- ANSI_DEFINE (HIDDEN, "8", "28")
- ANSI_DEFINE (STRIKE, "9", "29")
- ANSI_DEFINE (BLACK, "30", "39")
- ANSI_DEFINE (RED, "31", "39")
- ANSI_DEFINE (GREEN, "32", "39")
- ANSI_DEFINE (YELLOW, "33", "39")
- ANSI_DEFINE (BLUE, "34", "39")
- ANSI_DEFINE (MAGENTA, "35", "39")
- ANSI_DEFINE (CYAN, "36", "39")
- ANSI_DEFINE (WHITE, "37", "39")
- ANSI_DEFINE (BRIGHT_BLACK, "90", "39")
- ANSI_DEFINE (BRIGHT_RED, "91", "39")
- ANSI_DEFINE (BRIGHT_GREEN, "92", "39")
- ANSI_DEFINE (BRIGHT_YELLOW, "93", "39")
- ANSI_DEFINE (BRIGHT_BLUE, "94", "39")
- ANSI_DEFINE (BRIGHT_MAGENTA, "95", "39")
- ANSI_DEFINE (BRIGHT_CYAN, "96", "39")
- ANSI_DEFINE (BRIGHT_WHITE, "97", "39")
- ANSI_DEFINE (BG_BLACK, "40", "49")
- ANSI_DEFINE (BG_RED, "41", "49")
- ANSI_DEFINE (BG_GREEN, "42", "49")
- ANSI_DEFINE (BG_YELLOW, "43", "49")
- ANSI_DEFINE (BG_BLUE, "44", "49")
- ANSI_DEFINE (BG_MAGENTA, "45", "49")
- ANSI_DEFINE (BG_CYAN, "46", "49")
- ANSI_DEFINE (BG_WHITE, "47", "49")
- ANSI_DEFINE (BG_BRIGHT_BLACK, "100", "49")
- ANSI_DEFINE (BG_BRIGHT_RED, "101", "49")
- ANSI_DEFINE (BG_BRIGHT_GREEN, "102", "49")
- ANSI_DEFINE (BG_BRIGHT_YELLOW, "103", "49")
- ANSI_DEFINE (BG_BRIGHT_BLUE, "104", "49")
- ANSI_DEFINE (BG_BRIGHT_MAGENTA, "105", "49")
- ANSI_DEFINE (BG_BRIGHT_CYAN, "106", "49")
- ANSI_DEFINE (BG_BRIGHT_WHITE, "107", "49")

6.3.1 Typedef Documentation

6.3.1.1 AW

```
template<comppare::internal::concepts::Streamable T>
using comppare::internal::ansi::AW = AnsiWrapper<std::decay_t<T>>
```

6.3.2 Function Documentation

6.3.2.1 ANSI_DEFINE() [1/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_BLACK ,
            "40" ,
            "49" )
```

6.3.2.2 ANSI_DEFINE() [2/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_BLUE ,
            "44" ,
            "49" )
```

6.3.2.3 ANSI_DEFINE() [3/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_BRIGHT_BLACK ,
            "100" ,
            "49" )
```

6.3.2.4 ANSI_DEFINE() [4/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_BRIGHT_BLUE ,
            "104" ,
            "49" )
```

6.3.2.5 ANSI_DEFINE() [5/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_BRIGHT_CYAN ,
            "106" ,
            "49" )
```

6.3.2.6 ANSI_DEFINE() [6/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_BRIGHT_GREEN ,
            "102" ,
            "49" )
```

6.3.2.7 ANSI_DEFINE() [7/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_BRIGHT_MAGENTA ,
            "105" ,
            "49" )
```

6.3.2.8 ANSI_DEFINE() [8/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_BRIGHT_RED ,
            "101" ,
            "49" )
```

6.3.2.9 ANSI_DEFINE() [9/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_BRIGHT_WHITE ,
            "107" ,
            "49" )
```

6.3.2.10 ANSI_DEFINE() [10/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_BRIGHT_YELLOW ,
            "103" ,
            "49" )
```

6.3.2.11 ANSI_DEFINE() [11/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_CYAN ,
            "46" ,
            "49" )
```

6.3.2.12 ANSI_DEFINE() [12/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_GREEN ,
            "42" ,
            "49" )
```

6.3.2.13 ANSI_DEFINE() [13/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_MAGENTA ,
            "45" ,
            "49" )
```

6.3.2.14 ANSI_DEFINE() [14/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_RED ,
            "41" ,
            "49" )
```

6.3.2.15 ANSI_DEFINE() [15/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_WHITE ,
            "47" ,
            "49" )
```

6.3.2.16 ANSI_DEFINE() [16/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BG_YELLOW ,
            "43" ,
            "49" )
```

6.3.2.17 ANSI_DEFINE() [17/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BLACK ,
            "30" ,
            "39" )
```

6.3.2.18 ANSI_DEFINE() [18/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BLINK ,
            "5" ,
            "25" )
```

6.3.2.19 ANSI_DEFINE() [19/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BLUE ,
            "34" ,
            "39" )
```

6.3.2.20 ANSI_DEFINE() [20/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BOLD ,
            "1" ,
            "22" )
```

6.3.2.21 ANSI_DEFINE() [21/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BRIGHT_BLACK ,
            "90" ,
            "39" )
```

6.3.2.22 ANSI_DEFINE() [22/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BRIGHT_BLUE ,
            "94" ,
            "39" )
```

6.3.2.23 ANSI_DEFINE() [23/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BRIGHT_CYAN ,
            "96" ,
            "39" )
```

6.3.2.24 ANSI_DEFINE() [24/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BRIGHT_GREEN ,
            "92" ,
            "39" )
```

6.3.2.25 ANSI_DEFINE() [25/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BRIGHT_MAGENTA ,
            "95" ,
            "39" )
```

6.3.2.26 ANSI_DEFINE() [26/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BRIGHT_RED ,
            "91" ,
            "39" )
```

6.3.2.27 ANSI_DEFINE() [27/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BRIGHT_WHITE ,
            "97" ,
            "39" )
```

6.3.2.28 ANSI_DEFINE() [28/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            BRIGHT_YELLOW ,
            "93" ,
            "39" )
```

6.3.2.29 ANSI_DEFINE() [29/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            CYAN ,
            "36" ,
            "39" )
```

6.3.2.30 ANSI_DEFINE() [30/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            DIM ,
            "2" ,
            "22" )
```

6.3.2.31 ANSI_DEFINE() [31/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            GREEN ,
            "32" ,
            "39" )
```

6.3.2.32 ANSI_DEFINE() [32/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            HIDDEN ,
            "8" ,
            "28" )
```

6.3.2.33 ANSI_DEFINE() [33/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            ITALIC ,
            "3" ,
            "23" )
```

6.3.2.34 ANSI_DEFINE() [34/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            MAGENTA ,
            "35" ,
            "39" )
```

6.3.2.35 ANSI_DEFINE() [35/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            RED ,
            "31" ,
            "39" )
```

6.3.2.36 ANSI_DEFINE() [36/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            RESET ,
            "0" ,
            "0" )
```

6.3.2.37 ANSI_DEFINE() [37/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            REVERSE ,
            "7" ,
            "27" )
```

6.3.2.38 ANSI_DEFINE() [38/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            STRIKE ,
            "9" ,
            "29" )
```

6.3.2.39 ANSI_DEFINE() [39/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            UNDERLINE ,
            "4" ,
            "24" )
```

6.3.2.40 ANSI_DEFINE() [40/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            WHITE ,
            "37" ,
            "39" )
```

6.3.2.41 ANSI_DEFINE() [41/41]

```
comppare::internal::ansi::ANSI_DEFINE (
            YELLOW ,
            "33" ,
            "39" )
```

6.4 comppare::internal::concepts Namespace Reference

Concepts

- concept Streamable
- concept FloatingPoint
- concept Integral
- concept Arithmetic
- concept String
- concept Void
- concept RangeOfArithmetic

6.5 comppare::internal::helper Namespace Reference

Functions

- template<typename T>
  T get_arg_value (std::string_view option, const char *nextArg)
- static void parse_args (int argc, char **argv)

6.5.1 Function Documentation

6.5.1.1 get_arg_value()

```
template<typename T>
T comppare::internal::helper::get_arg_value (
            std::string_view option,
            const char * nextArg)
```

6.5.1.2 parse_args()

```
void comppare::internal::helper::parse_args (
            int argc,
            char ** argv)  [inline], [static]
```

6.6 comppare::internal::policy Namespace Reference

Namespaces

- namespace autopolicy

Classes

- struct is_metric_value
- struct is_metric_value< MetricValue< U > >
- class MetricValue

Concepts

- concept MetricValueSpec
- concept ErrorPolicy

Functions

- template<class EP, class V, class Tol>
  void compute_error (EP &ep, const V &a, const V &b, Tol tol)
- template<class EP, class V>
  void compute_error (EP &ep, const V &a, const V &b)
- template<class EP, class Tol>
  bool is_fail (const EP &ep, Tol tol)
- template<class EP>
  bool is_fail (const EP &ep)

Variables

- template<typename M>
  constexpr bool is_metric_value_v = is_metric_value<std::remove_cv_t<M>>::value

6.6.1 Function Documentation

6.6.1.1 compute_error() [1/2]

```
template<class EP, class V>
void comppare::internal::policy::compute_error (
            EP & ep,
            const V & a,
            const V & b)  [inline]
```

6.6.1.2 compute_error() [2/2]

```
template<class EP, class V, class Tol>
void comppare::internal::policy::compute_error (
            EP & ep,
            const V & a,
            const V & b,
            Tol tol)  [inline]
```

6.6.1.3 is_fail() [1/2]

```
template<class EP>
bool comppare::internal::policy::is_fail (
            const EP & ep)  [inline]
```

6.6.1.4 is_fail() [2/2]

```
template<class EP, class Tol>
bool comppare::internal::policy::is_fail (
            const EP & ep,
            Tol tol)  [inline]
```

6.6.2 Variable Documentation

6.6.2.1 is_metric_value_v

```
template<typename M>
bool comppare::internal::policy::is_metric_value_v = is_metric_value<std::remove_cv_t<M>>↩
::value  [inline], [constexpr]
```

6.7 comppare::internal::policy::autopolicy Namespace Reference

Classes

- class ArithmeticErrorPolicy
- struct AutoPolicy
- struct AutoPolicy< T >
- class RangeErrorPolicy
- class StringEqualPolicy

Concepts

- concept SupportedByAutoPolicy

Typedefs

- template<typename T>
  using AutoPolicy_t = typename AutoPolicy<T>::type

6.7.1 Typedef Documentation

6.7.1.1 AutoPolicy_t

```
template<typename T>
using comppare::internal::policy::autopolicy::AutoPolicy_t = typename AutoPolicy<T>::type
```

6.8 comppare::plugin Namespace Reference

Classes

- class Plugin
- class PluginArgParser

Concepts

- concept ValidPlugin

Chapter 7

Concept Documentation

7.1 comppare::internal::concepts::Arithmetic Concept Reference

`#include <concepts.hpp>`

7.1.1 Concept definition

```
template<typename T>
concept comppare::internal::concepts::Arithmetic =  FloatingPoint<T> || Integral<T>
```

7.2 comppare::internal::concepts::FloatingPoint Concept Reference

`#include <concepts.hpp>`

7.2.1 Concept definition

```
template<typename T>
concept comppare::internal::concepts::FloatingPoint =  std::floating_point<std::remove_cvref_t<T»
```

7.3 comppare::internal::concepts::Integral Concept Reference

`#include <concepts.hpp>`

7.3.1 Concept definition

```
template<typename T>
concept comppare::internal::concepts::Integral =  std::integral<std::remove_cvref_t<T»
```

7.4 comppare::internal::concepts::RangeOfArithmetic Concept Reference

`#include <concepts.hpp>`

7.4.1 Concept definition

```
template<typename R>
concept comppare::internal::concepts::RangeOfArithmetic =
        std::ranges::forward_range<std::remove_cvref_t<R» &&
      Arithmetic<std::remove_cvref_t<std::ranges::range_value_t<std::remove_cvref_t<R>»> &&
      (!String<std::remove_cvref_t<R>>)
```

7.5 comppare::internal::concepts::Streamable Concept Reference

`#include <concepts.hpp>`

7.5.1 Concept definition

```
template<typename T>
concept comppare::internal::concepts::Streamable =
        requires(std::ostream &os, T v) { { os « v } -> std::same_as<std::ostream&>; }
```

7.6 comppare::internal::concepts::String Concept Reference

`#include <concepts.hpp>`

7.6.1 Concept definition

```
template<typename T>
concept comppare::internal::concepts::String =  std::same_as<std::remove_cvref_t<T>, std::string>
```

7.7 comppare::internal::concepts::Void Concept Reference

`#include <concepts.hpp>`

7.7.1 Concept definition

```
template<typename T>
concept comppare::internal::concepts::Void =  std::is_void_v<std::remove_cvref_t<T»
```

7.8 comppare::internal::policy::autopolicy::SupportedByAutoPolicy Concept Reference

`#include <policy.hpp>`

7.8.1 Concept definition

```
template<typename T>
concept comppare::internal::policy::autopolicy::SupportedByAutoPolicy =
           comppare::internal::concepts::Arithmetic<T> ||
           comppare::internal::concepts::String<T> ||
           comppare::internal::concepts::RangeOfArithmetic<T>
```

7.9 comppare::internal::policy::ErrorPolicy Concept Reference

`#include <policy.hpp>`

7.9.1 Concept definition

```
template<typename Val, typename EP>
concept comppare::internal::policy::ErrorPolicy =  requires

    {
        { EP::metric_count() } -> std::convertible_to<std::size_t>;
        { EP::metric_name(std::size_t{}) } -> std::convertible_to<std::string_view>;
    } &&

                        (requires(EP ep, const Val &a, const Val &b, double t) { ep.compute_error(a, b,
      t); } || requires(EP ep, const Val &a, const Val &b) { ep.compute_error(a, b); }) &&

                        (requires(EP ep, std::size_t i) {
            { ep.metric(i) } -> MetricValueSpec; } || requires(EP ep, std::size_t i) {
            { ep.metric(i) } -> std::convertible_to<double>; } || requires(EP ep, std::size_t i) {
            { ep.metric(i) } -> std::same_as<std::string>; }) &&

                        (requires(EP ep, double t) {
            { ep.is_fail(t) } -> std::convertible_to<bool>; } || requires(EP ep) {
            { ep.is_fail() } -> std::convertible_to<bool>; })
```

7.10 comppare::internal::policy::MetricValueSpec Concept Reference

`#include <policy.hpp>`

7.10.1 Concept definition

```
template<typename M>
concept comppare::internal::policy::MetricValueSpec =  is_metric_value_v<M>
```

7.11 comppare::OutSpec Concept Reference

`#include <comppare.hpp>`

7.11.1 Concept definition

```
template<typename T>
concept comppare::OutSpec =
        comppare::internal::policy::ErrorPolicy<
            typename spec<T>::value_t,
            typename spec<T>::policy_t>
```

7.12 comppare::plugin::ValidPlugin Concept Reference

`#include <plugin.hpp>`

7.12.1 Concept definition

```
template<template< class, class > class P, class InTup, class OutTup, class Func>
concept comppare::plugin::ValidPlugin =
        requires { { P<InTup, OutTup>::instance() } -> std::same_as<std::shared_ptr<P<InTup, OutTup>>>; }
        &&
        requires(const std::string& name, Func&& user_fn, const InTup& inputs, OutTup& outputs)
        { std::declval<P<InTup, OutTup>&>().register_impl(name, user_fn, inputs, outputs); }
        &&
        std::derived_from<P<InTup, OutTup>, plugin::Plugin<InTup, OutTup»
```

Chapter 8

Class Documentation

8.1 comppare::internal::ansi::AnsiWrapper< T > Class Template Reference

`#include <ansi.hpp>`

Public Member Functions

- AnsiWrapper (const char *on, const char *off, T v)

Private Attributes

- const char * on_
- const char * off_
- T val_

Friends

- std::ostream & operator<< (std::ostream &os, AnsiWrapper const &w)

8.1.1 Constructor & Destructor Documentation

8.1.1.1 AnsiWrapper()

```
template<comppare::internal::concepts::Streamable T>
comppare::internal::ansi::AnsiWrapper< T >::AnsiWrapper (
            const char * on,
            const char * off,
            T v)  [inline]
```

8.1.2 Friends And Related Symbol Documentation

8.1.2.1 operator<<

```
template<comppare::internal::concepts::Streamable T>
std::ostream & operator<< (
            std::ostream & os,
            AnsiWrapper< T > const & w)  [friend]
```

8.1.3 Member Data Documentation

8.1.3.1 off_

```
template<comppare::internal::concepts::Streamable T>
const char* comppare::internal::ansi::AnsiWrapper< T >::off_  [private]
```

8.1.3.2 on_

```
template<comppare::internal::concepts::Streamable T>
const char* comppare::internal::ansi::AnsiWrapper< T >::on_  [private]
```

8.1.3.3 val_

```
template<comppare::internal::concepts::Streamable T>
T comppare::internal::ansi::AnsiWrapper< T >::val_  [private]
```

The documentation for this class was generated from the following file:

- include/comppare/internal/ansi.hpp

8.2 comppare::internal::policy::autopolicy::ArithmeticErrorPolicy< T > Class Template Reference

`#include <policy.hpp>`

Public Member Functions

- MetricValue< T > metric (std::size_t) const
- bool is_fail () const
- void compute_error (const T &a, const T &b)

Static Public Member Functions

- static constexpr std::size_t metric_count ()
- static constexpr std::string_view metric_name (std::size_t)

Private Attributes

- T error_ = T(0)
- std::string err_msg_
- bool valid_ = true

Static Private Attributes

- static constexpr std::array names {"Total|err|"}

8.2.1 Member Function Documentation

8.2.1.1 compute_error()

```
template<typename T>
void comppare::internal::policy::autopolicy::ArithmeticErrorPolicy< T >::compute_error (
            const T & a,
            const T & b)  [inline]
```

8.2.1.2 is_fail()

```
template<typename T>
bool comppare::internal::policy::autopolicy::ArithmeticErrorPolicy< T >::is_fail () const
[inline]
```

8.2.1.3 metric()

```
template<typename T>
MetricValue< T > comppare::internal::policy::autopolicy::ArithmeticErrorPolicy< T >::metric (
            std::size_t ) const  [inline]
```

8.2.1.4 metric_count()

```
template<typename T>
constexpr std::size_t comppare::internal::policy::autopolicy::ArithmeticErrorPolicy< T >↩
::metric_count ()  [inline], [static], [constexpr]
```

8.2.1.5 metric_name()

```
template<typename T>
constexpr std::string_view comppare::internal::policy::autopolicy::ArithmeticErrorPolicy< T
>::metric_name (
            std::size_t )  [inline], [static], [constexpr]
```

8.2.2 Member Data Documentation

8.2.2.1 err_msg_

```
template<typename T>
std::string comppare::internal::policy::autopolicy::ArithmeticErrorPolicy< T >::err_msg_↩
[private]
```

8.2.2.2 error_

```
template<typename T>
T comppare::internal::policy::autopolicy::ArithmeticErrorPolicy< T >::error_ = T(0)  [private]
```

8.2.2.3 names

```
template<typename T>
std::array comppare::internal::policy::autopolicy::ArithmeticErrorPolicy< T >::names {"Total|err|"}
[static], [constexpr], [private]
```

8.2.2.4 valid_

```
template<typename T>
bool comppare::internal::policy::autopolicy::ArithmeticErrorPolicy< T >::valid_ = true  [private]
```

The documentation for this class was generated from the following file:

- include/comppare/internal/policy.hpp

8.3 comppare::internal::policy::autopolicy::AutoPolicy< T > Struct Template Reference

The documentation for this struct was generated from the following file:

- include/comppare/internal/policy.hpp

8.4 comppare::internal::policy::autopolicy::AutoPolicy< T > Struct Template Reference

```
#include <policy.hpp>
```

Public Types

- using type = ArithmeticErrorPolicy<std::remove_cvref_t<T>>
- using type = StringEqualPolicy
- using type = RangeErrorPolicy<T>

8.4.1 Member Typedef Documentation

8.4.1.1 type [1/3]

```
template<typename T>
using comppare::internal::policy::autopolicy::AutoPolicy< T >::type = ArithmeticErrorPolicy<std↩
::remove_cvref_t<T>>
```

8.4.1.2 type [2/3]

```
template<typename T>
using comppare::internal::policy::autopolicy::AutoPolicy< T >::type = StringEqualPolicy
```

8.4.1.3 type [3/3]

```
template<typename T>
using comppare::internal::policy::autopolicy::AutoPolicy< T >::type = RangeErrorPolicy<T>
```

The documentation for this struct was generated from the following file:

- include/comppare/internal/policy.hpp

8.5 comppare::config Class Reference

```
#include <config.hpp>
```

Public Types

- using clock_t = std::chrono::steady_clock
- using time_point_t = std::chrono::time_point<clock_t>

Public Member Functions

- config (const config &)=delete
- config (config &&)=delete
- config & operator= (const config &)=delete
- config & operator= (config &&)=delete

Static Public Member Functions

- static uint64_t warmup_iters ()
- static void set_warmup_iters (uint64_t v)
- static uint64_t bench_iters ()
- static void set_bench_iters (uint64_t v)
- static void reset_roi_us ()
- static void set_roi_us (const time_point_t &start, const time_point_t &end)
- static void set_roi_us (const float start, const float end)
- static void set_roi_us (const double start, const double end)
- template<typename Rep, typename Period>
  static void set_roi_us (std::chrono::duration< Rep, Period > v)
- static void set_roi_us (const double v)
- static void set_roi_us (const float v)
- template<typename Rep, typename Period>
  static void increment_roi_us (std::chrono::duration< Rep, Period > v)
- static void increment_roi_us (const double v)
- static void increment_roi_us (const float v)
- static void set_warmup_us (const time_point_t &start, const time_point_t &end)
- static void set_warmup_us (const float start, const float end)
- static void set_warmup_us (const double start, const double end)
- template<typename Rep, typename Period>
  static void set_warmup_us (std::chrono::duration< Rep, Period > v)
- static void set_warmup_us (const double v)
- static void set_warmup_us (const float v)
- static double get_roi_us ()
- static double get_warmup_us ()
- template<std::floating_point T>
  static T & fp_tolerance ()
- template<std::floating_point T>
  static void set_fp_tolerance (T v)
- static void set_all_fp_tolerance (long double v)

Private Member Functions

- config ()=default

Static Private Member Functions

- static config & instance ()

Private Attributes

- double roi_ {0.0}
- double warmup_ {0.0}
- uint64_t warmup_iters_ {100}
- uint64_t bench_iters_ {100}

8.5.1 Member Typedef Documentation

8.5.1.1 clock_t

```
using comppare::config::clock_t = std::chrono::steady_clock
```

8.5.1.2 time_point_t

```
using comppare::config::time_point_t = std::chrono::time_point<clock_t>
```

8.5.2 Constructor & Destructor Documentation

8.5.2.1 config() [1/3]

```
comppare::config::config (
            const config & )  [delete]
```

8.5.2.2 config() [2/3]

```
comppare::config::config (
            config && )  [delete]
```

8.5.2.3 config() [3/3]

```
comppare::config::config ()  [private], [default]
```

8.5.3 Member Function Documentation

8.5.3.1 bench_iters()

```
uint64_t comppare::config::bench_iters ()  [inline], [static]
```

8.5.3.2 fp_tolerance()

```
template<std::floating_point T>
T & comppare::config::fp_tolerance ()  [inline], [static]
```

8.5.3.3 get_roi_us()

```
double comppare::config::get_roi_us ()  [inline], [static]
```

8.5.3.4 get_warmup_us()

```
double comppare::config::get_warmup_us ()  [inline], [static]
```

8.5.3.5 increment_roi_us() [1/3]

```
void comppare::config::increment_roi_us (
            const double v)  [inline], [static]
```

8.5.3.6 increment_roi_us() [2/3]

```
void comppare::config::increment_roi_us (
            const float v)  [inline], [static]
```

8.5.3.7 increment_roi_us() [3/3]

```
template<typename Rep, typename Period>
void comppare::config::increment_roi_us (
            std::chrono::duration< Rep, Period > v)  [inline], [static]
```

8.5.3.8 instance()

```
config & comppare::config::instance ()  [inline], [static], [private]
```

8.5.3.9 operator=() [1/2]

```
config & comppare::config::operator= (
            config && )  [delete]
```

8.5.3.10 operator=() [2/2]

```
config & comppare::config::operator= (
            const config & )  [delete]
```

8.5.3.11 reset_roi_us()

```
void comppare::config::reset_roi_us ()  [inline], [static]
```

8.5.3.12 set_all_fp_tolerance()

```
void comppare::config::set_all_fp_tolerance (
            long double v)  [inline], [static]
```

8.5.3.13 set_bench_iters()

```
void comppare::config::set_bench_iters (
            uint64_t v)  [inline], [static]
```

8.5.3.14 set_fp_tolerance()

```
template<std::floating_point T>
void comppare::config::set_fp_tolerance (
            T v)  [inline], [static]
```

8.5.3.15 set_roi_us() [1/6]

```
void comppare::config::set_roi_us (
            const double start,
            const double end)  [inline], [static]
```

8.5.3.16 set_roi_us() [2/6]

```
void comppare::config::set_roi_us (
            const double v)  [inline], [static]
```

8.5.3.17 set_roi_us() [3/6]

```
void comppare::config::set_roi_us (
            const float start,
            const float end)  [inline], [static]
```

8.5.3.18 set_roi_us() [4/6]

```
void comppare::config::set_roi_us (
            const float v)  [inline], [static]
```

8.5.3.19 set_roi_us() [5/6]

```
void comppare::config::set_roi_us (
            const time_point_t & start,
            const time_point_t & end)  [inline], [static]
```

8.5.3.20 set_roi_us() [6/6]

```
template<typename Rep, typename Period>
void comppare::config::set_roi_us (
            std::chrono::duration< Rep, Period > v)  [inline], [static]
```

8.5.3.21 set_warmup_iters()

```
void comppare::config::set_warmup_iters (
            uint64_t v)  [inline], [static]
```

8.5.3.22 set_warmup_us() [1/6]

```
void comppare::config::set_warmup_us (
            const double start,
            const double end)  [inline], [static]
```

8.5.3.23 set_warmup_us() [2/6]

```
void comppare::config::set_warmup_us (
            const double v)  [inline], [static]
```

8.5.3.24 set_warmup_us() [3/6]

```
void comppare::config::set_warmup_us (
            const float start,
            const float end)  [inline], [static]
```

8.5.3.25 set_warmup_us() [4/6]

```
void comppare::config::set_warmup_us (
            const float v)  [inline], [static]
```

8.5.3.26 set_warmup_us() [5/6]

```
void comppare::config::set_warmup_us (
            const time_point_t & start,
            const time_point_t & end)  [inline], [static]
```

8.5.3.27 set_warmup_us() [6/6]

```
template<typename Rep, typename Period>
void comppare::config::set_warmup_us (
            std::chrono::duration< Rep, Period > v)  [inline], [static]
```

8.5.3.28 warmup_iters()

```
uint64_t comppare::config::warmup_iters ()  [inline], [static]
```

8.5.4 Member Data Documentation

8.5.4.1 bench_iters_

```
uint64_t comppare::config::bench_iters_ {100}  [private]
```

8.5.4.2 roi_

```
double comppare::config::roi_ {0.0}  [private]
```

8.5.4.3 warmup_

```
double comppare::config::warmup_ {0.0}  [private]
```

8.5.4.4 warmup_iters_

```
uint64_t comppare::config::warmup_iters_ {100}  [private]
```

The documentation for this class was generated from the following file:

- include/comppare/internal/config.hpp

8.6 comppare::current_state Class Reference

```
#include <config.hpp>
```

Public Member Functions

- current_state (const current_state &)=delete
- current_state (current_state &&)=delete
- current_state & operator= (const current_state &)=delete
- current_state & operator= (current_state &&)=delete

Static Public Member Functions

- static bool using_plugin ()
- static void set_using_plugin (bool v)
- static std::string_view impl_name ()
- static void set_impl_name (std::string_view name)

Private Member Functions

- current_state ()=default

Static Private Member Functions

- static current_state & instance ()

Private Attributes

- bool using_plugin_ {false}
- std::string_view impl_name_

8.6.1 Constructor & Destructor Documentation

8.6.1.1 current_state() [1/3]

```
comppare::current_state::current_state (
            const current_state & )  [delete]
```

8.6.1.2 current_state() [2/3]

```
comppare::current_state::current_state (
            current_state && )  [delete]
```

8.6.1.3 current_state() [3/3]

```
comppare::current_state::current_state ()  [private], [default]
```

8.6.2 Member Function Documentation

8.6.2.1 impl_name()

```
std::string_view comppare::current_state::impl_name ()  [inline], [static]
```

8.6.2.2 instance()

```
current_state & comppare::current_state::instance ()  [inline], [static], [private]
```

8.6.2.3 operator=() [1/2]

```
current_state & comppare::current_state::operator= (
            const current_state & )  [delete]
```

8.6.2.4 operator=() [2/2]

```
current_state & comppare::current_state::operator= (
            current_state && )  [delete]
```

8.6.2.5 set_impl_name()

```
void comppare::current_state::set_impl_name (
            std::string_view name)  [inline], [static]
```

8.6.2.6 set_using_plugin()

```
void comppare::current_state::set_using_plugin (
            bool v)  [inline], [static]
```

8.6.2.7 using_plugin()

```
bool comppare::current_state::using_plugin ()  [inline], [static]
```

8.6.3 Member Data Documentation

8.6.3.1 impl_name_

```
std::string_view comppare::current_state::impl_name_  [private]
```

8.6.3.2 using_plugin_

```
bool comppare::current_state::using_plugin_ {false}  [private]
```

The documentation for this class was generated from the following file:

- include/comppare/internal/config.hpp

8.7 comppare::InputContext< Inputs >::OutputContext< Specs >::Impl Struct Reference

Internal container representing one registered implementation.

Public Member Functions

- template<template< class, class > class Plugin>
  requires comppare::plugin::ValidPlugin<Plugin, InTup, OutTup, Func>
  decltype(auto) attach ()

  Attach a plugin to the output context.

Public Attributes

- std::string name
- Func fn
- InTup * inputs_ptr
- OutputContext * parent_ctx
- std::unique_ptr< OutTup > plugin_output = nullptr

8.7.1 Detailed Description

template<typename... Inputs>
template<OutSpec... Specs>
struct comppare::InputContext< Inputs >::OutputContext< Specs >::Impl

Internal container representing one registered implementation.

Each `Impl` bundles together:

- the *user function* (`fn`) under a given name,
- a pointer to the current input tuple,
- a back-reference to the owning `OutputContext`,
- and optionally, plugin-managed output storage.

This allows the framework to keep track of multiple competing implementations of the same operation (e.g. reference vs optimized), and to attach correctness/performance plugins such as Google Benchmark or NVBench.

8.7.2 Member Function Documentation

8.7.2.1 attach()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
template<template< class, class > class Plugin>
requires comppare::plugin::ValidPlugin<Plugin, InTup, OutTup, Func>
decltype(auto) comppare::InputContext< Inputs >::OutputContext< Specs >::Impl::attach ()
[inline]
```

Attach a plugin to the output context.

Template Parameters

Plugin	The plugin type to attach.

Returns

The result of the plugin attachment.

8.7.3 Member Data Documentation

8.7.3.1 fn

```
template<typename...  Inputs>
template<OutSpec...  Specs>
Func comppare::InputContext< Inputs >::OutputContext< Specs >::Impl::fn
```

8.7.3.2 inputs_ptr

```
template<typename...  Inputs>
template<OutSpec...  Specs>
InTup* comppare::InputContext< Inputs >::OutputContext< Specs >::Impl::inputs_ptr
```

8.7.3.3 name

```
template<typename...  Inputs>
template<OutSpec...  Specs>
std::string comppare::InputContext< Inputs >::OutputContext< Specs >::Impl::name
```

8.7.3.4 parent_ctx

```
template<typename...  Inputs>
template<OutSpec...  Specs>
OutputContext* comppare::InputContext< Inputs >::OutputContext< Specs >::Impl::parent_ctx
```

8.7.3.5 plugin_output

```
template<typename...  Inputs>
template<OutSpec...  Specs>
std::unique_ptr<OutTup> comppare::InputContext< Inputs >::OutputContext< Specs >::Impl↩
::plugin_output = nullptr
```

The documentation for this struct was generated from the following file:

- include/comppare/comppare.hpp

8.8 comppare::InputContext< Inputs > Class Template Reference

InputContext class template to hold input parameters for the comparison framework.

`#include <comppare.hpp>`

Classes

- class OutputContext
  OutputContext class template to hold output parameters and manage implementations.

8.8.1 Detailed Description

template<typename... Inputs>
class comppare::InputContext< Inputs >

InputContext class template to hold input parameters for the comparison framework.

Template Parameters

Inputs	

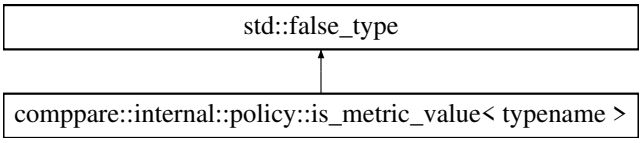
The documentation for this class was generated from the following file:

- include/comppare/comppare.hpp

8.9 comppare::internal::policy::is_metric_value< typename > Struct Template Reference

`#include <policy.hpp>`

Inheritance diagram for comppare::internal::policy::is_metric_value< typename >:



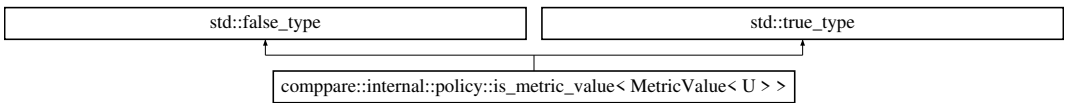
The documentation for this struct was generated from the following file:

- include/comppare/internal/policy.hpp

8.10 comppare::internal::policy::is_metric_value< MetricValue< U > > Struct Template Reference

`#include <policy.hpp>`

Inheritance diagram for comppare::internal::policy::is_metric_value< MetricValue< U > >:



The documentation for this struct was generated from the following file:

- include/comppare/internal/policy.hpp

8.11 comppare::internal::policy::MetricValue< T > Class Template Reference

`#include <policy.hpp>`

Public Member Functions

- MetricValue (T v)
- MetricValue (T v, bool is_fail)
- MetricValue (T v, bool is_fail, bool valid, std::string_view msg)

Private Attributes

- T value_
- bool is_fail_ {false}
- bool valid_ {true}
- std::string_view err_msg_

Friends

- std::ostream & operator<< (std::ostream &os, MetricValue< T > const &mv)

8.11.1 Constructor & Destructor Documentation

8.11.1.1 MetricValue() [1/3]

```
template<comppare::internal::concepts::Streamable T>
comppare::internal::policy::MetricValue< T >::MetricValue (
            T v)  [inline]
```

8.11.1.2 MetricValue() [2/3]

```
template<comppare::internal::concepts::Streamable T>
comppare::internal::policy::MetricValue< T >::MetricValue (
            T v,
            bool is_fail)  [inline]
```

8.11.1.3 MetricValue() [3/3]

```
template<comppare::internal::concepts::Streamable T>
comppare::internal::policy::MetricValue< T >::MetricValue (
            T v,
            bool is_fail,
            bool valid,
            std::string_view msg)  [inline]
```

8.11.2 Friends And Related Symbol Documentation

8.11.2.1 operator<<

```
template<comppare::internal::concepts::Streamable T>
std::ostream & operator<< (
            std::ostream & os,
            MetricValue< T > const & mv)  [friend]
```

8.11.3 Member Data Documentation

8.11.3.1 err_msg_

```
template<comppare::internal::concepts::Streamable T>
std::string_view comppare::internal::policy::MetricValue< T >::err_msg_  [private]
```

8.11.3.2 is_fail_

```
template<comppare::internal::concepts::Streamable T>
bool comppare::internal::policy::MetricValue< T >::is_fail_ {false}  [private]
```

8.11.3.3 valid_

```
template<comppare::internal::concepts::Streamable T>
bool comppare::internal::policy::MetricValue< T >::valid_ {true}  [private]
```

8.11.3.4 value_

```
template<comppare::internal::concepts::Streamable T>
T comppare::internal::policy::MetricValue< T >::value_  [private]
```

The documentation for this class was generated from the following file:

- include/comppare/internal/policy.hpp

8.12 comppare::InputContext< Inputs >::OutputContext< Specs > Class Template Reference

OutputContext class template to hold output parameters and manage implementations.

```
#include <comppare.hpp>
```

Classes

- struct Impl

  Internal container representing one registered implementation.

Public Member Functions

- template<typename... Ins>
  OutputContext (Ins &&...ins)
- OutputContext (const OutputContext &other)=delete
- OutputContext & operator= (const OutputContext &other)=delete
- OutputContext (OutputContext &&other)=delete
- OutputContext & operator= (OutputContext &&other)=delete
- template<typename F>
  requires std::invocable<F, const Inputs &..., val_t<Specs> &...>
  Impl & set_reference (std::string name, F &&f)
- template<typename F>
  requires std::invocable<F, const Inputs &..., val_t<Specs> &...>
  Impl & add (std::string name, F &&f)
- const OutPtr get_reference_output () const
- const OutPtr get_output (const size_t idx) const
- const OutPtr get_output (const std::string_view name) const
- template<typename U = void>
  requires (sizeof...(Specs) > 0)
  void get_reference_output (val_t< Specs > *...outs) const
- template<typename U = void>
  requires (sizeof...(Specs) > 0)
  void get_output (const size_t idx, val_t< Specs > *...outs) const
- template<typename U = void>
  requires (sizeof...(Specs) > 0)
  void get_output (const std::string_view name, val_t< Specs > *...outs) const
- void run (int argc=0, char **argv=nullptr)

Private Types

- template<typename S>
  using val_t = typename spec<S>::value_t
- template<typename S>
  using pol_t = typename spec<S>::policy_t
- using Func = std::function<void(const Inputs &..., val_t<Specs> &...)>
- using InTup = std::tuple<Inputs...>
- using OutTup = std::tuple<val_t<Specs>...>
- using PolicyTup = std::tuple<pol_t<Specs>...>
- using OutPtr = std::shared_ptr<OutTup>
- using OutVec = std::vector<OutPtr>

Private Member Functions

- void register_plugin (const std::shared_ptr< plugin::Plugin< InTup, OutTup > > &p)

  Register a plugin for the output context.
- void print_header () const
- void compute_errors (PolicyTup &errs, const OutTup &test, const OutTup &ref)
- bool any_fail (const PolicyTup &errs) const
- void print_metrics (const PolicyTup &errs) const
- OutPtr get_output_by_index_ (const size_t idx) const
- OutPtr get_output_by_name_ (const std::string_view name) const
- void unpack_output_ (const OutTup &outtup, val_t< Specs > *...outs) const

Static Private Member Functions

- template<std::size_t I>
  static constexpr auto spec_metric_count ()
- template<std::size_t I>
  static constexpr auto spec_metric_name (std::size_t m)

Private Attributes

- InTup inputs_
- OutVec outputs_
- PolicyTup policies_ref_
- std::shared_ptr< plugin::Plugin< InTup, OutTup > > plugins_
- std::vector< Impl > impls_

Static Private Attributes

- static constexpr size_t NUM_OUT = sizeof...(Specs)
- static constexpr int PRINT_COL_WIDTH = 20

8.12.1 Detailed Description

template<typename... Inputs>
template<OutSpec... Specs>
class comppare::InputContext< Inputs >::OutputContext< Specs >

OutputContext class template to hold output parameters and manage implementations.

Template Parameters

Specs	

8.12.2 Member Typedef Documentation

8.12.2.1 Func

```
template<typename...  Inputs>
template<OutSpec...  Specs>
using comppare::InputContext< Inputs >::OutputContext< Specs >::Func = std::function<void(const
Inputs &..., val_t<Specs> &...)>  [private]
```

8.12.2.2 InTup

```
template<typename...  Inputs>
template<OutSpec...  Specs>
using comppare::InputContext< Inputs >::OutputContext< Specs >::InTup = std::tuple<Inputs...>
[private]
```

8.12.2.3 OutPtr

```
template<typename...  Inputs>
template<OutSpec...  Specs>
using comppare::InputContext< Inputs >::OutputContext< Specs >::OutPtr = std::shared_ptr<OutTup>
[private]
```

8.12.2.4 OutTup

```
template<typename...  Inputs>
template<OutSpec...  Specs>
using comppare::InputContext< Inputs >::OutputContext< Specs >::OutTup = std::tuple<val_t<Specs>...>
[private]
```

8.12.2.5 OutVec

```
template<typename...  Inputs>
template<OutSpec...  Specs>
using comppare::InputContext< Inputs >::OutputContext< Specs >::OutVec = std::vector<OutPtr>
[private]
```

8.12.2.6 pol_t

```
template<typename...  Inputs>
template<OutSpec...  Specs>
template<typename S>
using comppare::InputContext< Inputs >::OutputContext< Specs >::pol_t = typename spec<S>↩
::policy_t  [private]
```

8.12.2.7 PolicyTup

```
template<typename...  Inputs>
template<OutSpec...  Specs>
using comppare::InputContext< Inputs >::OutputContext< Specs >::PolicyTup = std::tuple<pol_t<Specs>...>
[private]
```

8.12.2.8 val_t

```
template<typename...  Inputs>
template<OutSpec...  Specs>
template<typename S>
using comppare::InputContext< Inputs >::OutputContext< Specs >::val_t = typename spec<S>↩
::value_t  [private]
```

8.12.3 Constructor & Destructor Documentation

8.12.3.1 OutputContext() [1/3]

```
template<typename...  Inputs>
template<OutSpec...  Specs>
template<typename...  Ins>
comppare::InputContext< Inputs >::OutputContext< Specs >::OutputContext (
            Ins &&...  ins)  [inline], [explicit]
```

8.12.3.2 OutputContext() [2/3]

```
template<typename...  Inputs>
template<OutSpec...  Specs>
comppare::InputContext< Inputs >::OutputContext< Specs >::OutputContext (
            const OutputContext< Specs > & other)  [delete]
```

8.12.3.3 OutputContext() [3/3]

```
template<typename...  Inputs>
template<OutSpec...  Specs>
comppare::InputContext< Inputs >::OutputContext< Specs >::OutputContext (
            OutputContext< Specs > && other)  [delete]
```

8.12.4 Member Function Documentation

8.12.4.1 add()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
template<typename F>
requires std::invocable<F, const Inputs &..., val_t<Specs> &...>
Impl & comppare::InputContext< Inputs >::OutputContext< Specs >::add (
            std::string name,
            F && f)  [inline]
```

8.12.4.2 any_fail()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
bool comppare::InputContext< Inputs >::OutputContext< Specs >::any_fail (
            const PolicyTup & errs) const  [inline], [private]
```

8.12.4.3 compute_errors()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
void comppare::InputContext< Inputs >::OutputContext< Specs >::compute_errors (
            PolicyTup & errs,
            const OutTup & test,
            const OutTup & ref)  [inline], [private]
```

8.12.4.4 get_output() [1/4]

```
template<typename...  Inputs>
template<OutSpec...  Specs>
const OutPtr comppare::InputContext< Inputs >::OutputContext< Specs >::get_output (
            const size_t idx) const  [inline]
```

8.12.4.5 get_output() [2/4]

```
template<typename...  Inputs>
template<OutSpec...  Specs>
template<typename U = void>
requires (sizeof...(Specs) > 0)
void comppare::InputContext< Inputs >::OutputContext< Specs >::get_output (
            const size_t idx,
            val_t< Specs > *...  outs) const  [inline]
```

8.12.4.6 get_output() [3/4]

```
template<typename...  Inputs>
template<OutSpec...  Specs>
const OutPtr comppare::InputContext< Inputs >::OutputContext< Specs >::get_output (
            const std::string_view name) const  [inline]
```

8.12.4.7 get_output() [4/4]

```
template<typename...  Inputs>
template<OutSpec...  Specs>
template<typename U = void>
requires (sizeof...(Specs) > 0)
void comppare::InputContext< Inputs >::OutputContext< Specs >::get_output (
            const std::string_view name,
            val_t< Specs > *...  outs) const  [inline]
```

8.12.4.8 get_output_by_index_()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
OutPtr comppare::InputContext< Inputs >::OutputContext< Specs >::get_output_by_index_ (
            const size_t idx) const  [inline], [private]
```

8.12.4.9 get_output_by_name_()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
OutPtr comppare::InputContext< Inputs >::OutputContext< Specs >::get_output_by_name_ (
            const std::string_view name) const  [inline], [private]
```

8.12.4.10 get_reference_output() [1/2]

```
template<typename...  Inputs>
template<OutSpec...  Specs>
const OutPtr comppare::InputContext< Inputs >::OutputContext< Specs >::get_reference_output
() const  [inline]
```

8.12.4.11 get_reference_output() [2/2]

```
template<typename...  Inputs>
template<OutSpec...  Specs>
template<typename U = void>
requires (sizeof...(Specs) > 0)
void comppare::InputContext< Inputs >::OutputContext< Specs >::get_reference_output (
            val_t< Specs > *...  outs) const  [inline]
```

8.12.4.12 operator=() [1/2]

```
template<typename...  Inputs>
template<OutSpec...  Specs>
OutputContext & comppare::InputContext< Inputs >::OutputContext< Specs >::operator= (
            const OutputContext< Specs > & other)  [delete]
```

8.12.4.13 operator=() [2/2]

```
template<typename...  Inputs>
template<OutSpec...  Specs>
OutputContext & comppare::InputContext< Inputs >::OutputContext< Specs >::operator= (
            OutputContext< Specs > && other)  [delete]
```

8.12.4.14 print_header()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
void comppare::InputContext< Inputs >::OutputContext< Specs >::print_header () const  [inline],
[private]
```

8.12.4.15 print_metrics()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
void comppare::InputContext< Inputs >::OutputContext< Specs >::print_metrics (
            const PolicyTup & errs) const  [inline], [private]
```

8.12.4.16 register_plugin()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
void comppare::InputContext< Inputs >::OutputContext< Specs >::register_plugin (
            const std::shared_ptr< plugin::Plugin< InTup, OutTup > > & p)  [inline], [private]
```

Register a plugin for the output context.

Parameters

p	The shared pointer to the plugin to register.

8.12.4.17 run()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
void comppare::InputContext< Inputs >::OutputContext< Specs >::run (
            int argc = 0,
            char ** argv = nullptr)  [inline]
```

8.12.4.18 set_reference()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
template<typename F>
requires std::invocable<F, const Inputs &..., val_t<Specs> &...>
Impl & comppare::InputContext< Inputs >::OutputContext< Specs >::set_reference (
            std::string name,
            F && f)  [inline]
```

8.12.4.19 spec_metric_count()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
template<std::size_t I>
constexpr auto comppare::InputContext< Inputs >::OutputContext< Specs >::spec_metric_count ()
[inline], [static], [constexpr], [private]
```

8.12.4.20 spec_metric_name()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
template<std::size_t I>
constexpr auto comppare::InputContext< Inputs >::OutputContext< Specs >::spec_metric_name (
            std::size_t m)  [inline], [static], [constexpr], [private]
```

8.12.4.21 unpack_output_()

```
template<typename...  Inputs>
template<OutSpec...  Specs>
void comppare::InputContext< Inputs >::OutputContext< Specs >::unpack_output_ (
            const OutTup & outtup,
            val_t< Specs > *...  outs) const  [inline], [private]
```

8.12.5 Member Data Documentation

8.12.5.1 impls_

```
template<typename... Inputs>
template<OutSpec... Specs>
std::vector<Impl> comppare::InputContext< Inputs >::OutputContext< Specs >::impls_ [private]
```

8.12.5.2 inputs_

```
template<typename... Inputs>
template<OutSpec... Specs>
InTup comppare::InputContext< Inputs >::OutputContext< Specs >::inputs_ [private]
```

8.12.5.3 NUM_OUT

```
template<typename... Inputs>
template<OutSpec... Specs>
size_t comppare::InputContext< Inputs >::OutputContext< Specs >::NUM_OUT = sizeof...(Specs)
[static], [constexpr], [private]
```

8.12.5.4 outputs_

```
template<typename... Inputs>
template<OutSpec... Specs>
OutVec comppare::InputContext< Inputs >::OutputContext< Specs >::outputs_ [private]
```

8.12.5.5 plugins_

```
template<typename... Inputs>
template<OutSpec... Specs>
std::shared_ptr<plugin::Plugin<InTup, OutTup> > comppare::InputContext< Inputs >::OutputContext<
Specs >::plugins_ [private]
```

8.12.5.6 policies_ref_

```
template<typename... Inputs>
template<OutSpec... Specs>
PolicyTup comppare::InputContext< Inputs >::OutputContext< Specs >::policies_ref_ [private]
```

8.12.5.7 PRINT_COL_WIDTH

```
template<typename... Inputs>
template<OutSpec... Specs>
int comppare::InputContext< Inputs >::OutputContext< Specs >::PRINT_COL_WIDTH = 20 [static],
[constexpr], [private]
```

The documentation for this class was generated from the following file:

- include/comppare/comppare.hpp

8.13 comppare::plugin::Plugin< InTup, OutTup > Class Template Reference

`#include <plugin.hpp>`

Public Member Functions

- virtual ~Plugin ()=default
- virtual void initialize (int &, char **)
- virtual void run ()

8.13.1 Constructor & Destructor Documentation

8.13.1.1 ~Plugin()

```
template<class InTup, class OutTup>
virtual comppare::plugin::Plugin< InTup, OutTup >::~Plugin ()  [virtual], [default]
```

8.13.2 Member Function Documentation

8.13.2.1 initialize()

```
template<class InTup, class OutTup>
virtual void comppare::plugin::Plugin< InTup, OutTup >::initialize (
            int & ,
            char ** )  [inline], [virtual]
```

8.13.2.2 run()

```
template<class InTup, class OutTup>
virtual void comppare::plugin::Plugin< InTup, OutTup >::run ()  [inline], [virtual]
```

The documentation for this class was generated from the following file:

- include/comppare/plugin/plugin.hpp

8.14 comppare::plugin::PluginArgParser Class Reference

`#include <plugin.hpp>`

Public Member Functions

- PluginArgParser (std::string header, bool strict_missing_value=false)
- PluginArgParser (const PluginArgParser &)=delete
- PluginArgParser & operator= (const PluginArgParser &)=delete
- PluginArgParser (PluginArgParser &&)=default
- PluginArgParser & operator= (PluginArgParser &&)=default
- ~PluginArgParser ()=default
- std::pair< int, char ** > parse (int argc, char **argv)
- int argc () const
- char ** argv ()
- const std::vector< std::string > & args () const

Static Private Member Functions

- static bool starts_with (const std::string &s, const std::string &prefix)
- static std::vector< std::string > split_shell_like (const std::string &s)
- static void append_tokens (std::vector< std::string > &dst, const std::string &value)

Private Attributes

- std::string header_
- bool strict_
- std::vector< std::string > args_
- std::vector< char * > cargv_

8.14.1 Constructor & Destructor Documentation

8.14.1.1 PluginArgParser() [1/3]

```
comppare::plugin::PluginArgParser::PluginArgParser (
            std::string header,
            bool strict_missing_value = false)  [inline], [explicit]
```

8.14.1.2 PluginArgParser() [2/3]

```
comppare::plugin::PluginArgParser::PluginArgParser (
            const PluginArgParser & )  [delete]
```

8.14.1.3 PluginArgParser() [3/3]

```
comppare::plugin::PluginArgParser::PluginArgParser (
            PluginArgParser && )  [default]
```

8.14.1.4 ~PluginArgParser()

```
comppare::plugin::PluginArgParser::~PluginArgParser ()  [default]
```

8.14.2 Member Function Documentation

8.14.2.1 append_tokens()

```
void comppare::plugin::PluginArgParser::append_tokens (
            std::vector< std::string > & dst,
            const std::string & value)  [inline], [static], [private]
```

8.14.2.2 argc()

```
int comppare::plugin::PluginArgParser::argc () const  [inline], [nodiscard]
```

8.14.2.3 args()

```
const std::vector< std::string > & comppare::plugin::PluginArgParser::args () const  [inline],
[nodiscard]
```

8.14.2.4 argv()

```
char ** comppare::plugin::PluginArgParser::argv ()  [inline], [nodiscard]
```

8.14.2.5 operator=() [1/2]

```
PluginArgParser & comppare::plugin::PluginArgParser::operator= (
            const PluginArgParser & )  [delete]
```

8.14.2.6 operator=() [2/2]

```
PluginArgParser & comppare::plugin::PluginArgParser::operator= (
            PluginArgParser && )  [default]
```

8.14.2.7 parse()

```
std::pair< int, char ** > comppare::plugin::PluginArgParser::parse (
            int argc,
            char ** argv)  [inline], [nodiscard]
```

8.14.2.8 split_shell_like()

```
std::vector< std::string > comppare::plugin::PluginArgParser::split_shell_like (
            const std::string & s)  [inline], [static], [private]
```

8.14.2.9 starts_with()

```
bool comppare::plugin::PluginArgParser::starts_with (
            const std::string & s,
            const std::string & prefix)  [inline], [static], [private]
```

8.14.3 Member Data Documentation

8.14.3.1 args_

```
std::vector<std::string> comppare::plugin::PluginArgParser::args_  [private]
```

8.14.3.2 cargv_

```
std::vector<char*> comppare::plugin::PluginArgParser::cargv_  [private]
```

8.14.3.3 header_

```
std::string comppare::plugin::PluginArgParser::header_  [private]
```

8.14.3.4 strict_

```
bool comppare::plugin::PluginArgParser::strict_  [private]
```

The documentation for this class was generated from the following file:

- include/comppare/plugin/plugin.hpp

8.15 comppare::internal::policy::autopolicy::RangeErrorPolicy< R > Class Template Reference

```
#include <policy.hpp>
```

Public Member Functions

- MetricValue< T > metric (std::size_t i) const
- bool is_fail () const
- void compute_error (const R &a, const R &b)

Static Public Member Functions

- static constexpr std::size_t metric_count ()
- static constexpr std::string_view metric_name (std::size_t i)

Private Types

- using T = std::remove_cvref_t<std::ranges::range_value_t<R>>

Private Member Functions

- MetricValue< T > get_max () const
- MetricValue< T > get_mean () const
- MetricValue< T > get_total () const

Private Attributes

- T max_error_ = T(0)
- T total_error_ = T(0)
- std::size_t elem_cnt_ = 0
- bool valid_ = true
- std::string err_msg_

Static Private Attributes

- static constexpr std::array names {"Max|err|", "Mean|err|", "Total|err|"}

8.15.1 Member Typedef Documentation

8.15.1.1 T

```
template<typename R>
using comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::T = std::remove_cvref_↩
t<std::ranges::range_value_t<R>>  [private]
```

8.15.2 Member Function Documentation

8.15.2.1 compute_error()

```
template<typename R>
void comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::compute_error (
            const R & a,
            const R & b)  [inline]
```

8.15.2.2 get_max()

```
template<typename R>
MetricValue< T > comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::get_max ()
const  [inline], [private]
```

8.15.2.3 get_mean()

```
template<typename R>
MetricValue< T > comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::get_mean ()
const  [inline], [private]
```

8.15.2.4 get_total()

```
template<typename R>
MetricValue< T > comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::get_total ()
const  [inline], [private]
```

8.15.2.5 is_fail()

```
template<typename R>
bool comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::is_fail () const  [inline]
```

8.15.2.6 metric()

```
template<typename R>
MetricValue< T > comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::metric (
            std::size_t i) const  [inline]
```

8.15.2.7 metric_count()

```
template<typename R>
constexpr std::size_t comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::metric_↩
count ()  [inline], [static], [constexpr]
```

8.15.2.8 metric_name()

```
template<typename R>
constexpr std::string_view comppare::internal::policy::autopolicy::RangeErrorPolicy< R >↩
::metric_name (
            std::size_t i)  [inline], [static], [constexpr]
```

8.15.3 Member Data Documentation

8.15.3.1 elem_cnt_

```
template<typename R>
std::size_t comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::elem_cnt_ = 0
[private]
```

8.15.3.2 err_msg_

```
template<typename R>
std::string comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::err_msg_  [private]
```

8.15.3.3 max_error_

```
template<typename R>
T comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::max_error_ = T(0)  [private]
```

8.15.3.4 names

```
template<typename R>
std::array comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::names {"Max|err|",
"Mean|err|", "Total|err|"}  [static], [constexpr], [private]
```

8.15.3.5 total_error_

```
template<typename R>
T comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::total_error_ = T(0)  [private]
```

8.15.3.6 valid_

```
template<typename R>
bool comppare::internal::policy::autopolicy::RangeErrorPolicy< R >::valid_ = true  [private]
```

The documentation for this class was generated from the following file:

- include/comppare/internal/policy.hpp

8.16 comppare::spec< Value, Policy > Struct Template Reference

The documentation for this struct was generated from the following file:

- include/comppare/comppare.hpp

8.17 comppare::spec< spec< Value, Policy >, void > Struct Template Reference

```
#include <comppare.hpp>
```

Public Types

- using value_t = Value
- using policy_t = Policy

8.17.1 Member Typedef Documentation

8.17.1.1 policy_t

```
template<typename Value, typename Policy>
using comppare::spec< spec< Value, Policy >, void >::policy_t = Policy
```

8.17.1.2 value_t

```
template<typename Value, typename Policy>
using comppare::spec< spec< Value, Policy >, void >::value_t = Value
```

The documentation for this struct was generated from the following file:

- include/comppare/comppare.hpp

8.18 comppare::spec< Value, void > Struct Template Reference

```
#include <comppare.hpp>
```

Public Types

- using value_t = Value
- using policy_t = internal::policy::autopolicy::AutoPolicy_t<Value>

8.18.1 Member Typedef Documentation

8.18.1.1 policy_t

```
template<typename Value>
using comppare::spec< Value, void >::policy_t = internal::policy::autopolicy::AutoPolicy_t<Value>
```

8.18.1.2 value_t

```
template<typename Value>
using comppare::spec< Value, void >::value_t = Value
```

The documentation for this struct was generated from the following file:

- include/comppare/comppare.hpp

8.19 comppare::internal::policy::autopolicy::StringEqualPolicy Class Reference

`#include <policy.hpp>`

Public Member Functions

- MetricValue< std::string > metric (std::size_t) const
- bool is_fail () const
- void compute_error (const std::string &a, const std::string &b)

Static Public Member Functions

- static constexpr std::size_t metric_count ()
- static constexpr std::string_view metric_name (std::size_t)

Private Attributes

- bool eq_ {true}

Static Private Attributes

- static constexpr std::array names {"Equal?"}

8.19.1 Member Function Documentation

8.19.1.1 compute_error()

```
void comppare::internal::policy::autopolicy::StringEqualPolicy::compute_error (
            const std::string & a,
            const std::string & b)  [inline]
```

8.19.1.2 is_fail()

```
bool comppare::internal::policy::autopolicy::StringEqualPolicy::is_fail () const  [inline]
```

8.19.1.3 metric()

```
MetricValue< std::string > comppare::internal::policy::autopolicy::StringEqualPolicy::metric (
            std::size_t ) const  [inline]
```

8.19.1.4 metric_count()

```
constexpr std::size_t comppare::internal::policy::autopolicy::StringEqualPolicy::metric_count
() [inline], [static], [constexpr]
```

8.19.1.5 metric_name()

```
constexpr std::string_view comppare::internal::policy::autopolicy::StringEqualPolicy::metric↩
_name (
            std::size_t )  [inline], [static], [constexpr]
```

8.19.2 Member Data Documentation

8.19.2.1 eq_

```
bool comppare::internal::policy::autopolicy::StringEqualPolicy::eq_ {true}  [private]
```

8.19.2.2 names

```
std::array comppare::internal::policy::autopolicy::StringEqualPolicy::names {"Equal?"}  [static],
[constexpr], [private]
```

The documentation for this class was generated from the following file:

- include/comppare/internal/policy.hpp

Chapter 9

File Documentation

9.1 include/comppare/comppare.hpp File Reference

This file is the main include file for the ComPPare framework.

```
#include <chrono>
#include <cmath>
#include <functional>
#include <iomanip>
#include <iostream>
#include <string>
#include <vector>
#include <array>
#include <memory>
#include <string_view>
#include <concepts>
#include <stdexcept>
#include <comppare/internal/ansi.hpp>
#include <comppare/internal/config.hpp>
#include <comppare/internal/helper.hpp>
#include <comppare/internal/policy.hpp>
#include <comppare/plugin/plugin.hpp>
```

Classes

- struct comppare::spec< Value, void >
- struct comppare::spec< spec< Value, Policy >, void >
- struct **comppare::spec< Value, Policy >**
- class comppare::InputContext< Inputs >

  InputContext class template to hold input parameters for the comparison framework.
- class comppare::InputContext< Inputs >::OutputContext< Specs >

  OutputContext class template to hold output parameters and manage implementations.
- struct comppare::InputContext< Inputs >::OutputContext< Specs >::Impl

  Internal container representing one registered implementation.

Namespaces

- namespace comppare

  ComPPare framework main namespace.

Concepts

- concept comppare::OutSpec

Macros

- #define HOTLOOPSTART auto &&hotloop_body = [&]() { /* start of lambda */
- #define COMPPARE_HOTLOOP_BENCH
- #define HOTLOOPEND
- #define HOTLOOP(LOOP_BODY)
- #define MANUAL_TIMER_START auto t_manual_start = comppare::config::clock_t::now();
- #define MANUAL_TIMER_END
- #define SET_ITERATION_TIME(TIME)
- #define GPU_HOTLOOPSTART auto &&hotloop_body = [&]() { /* start of lambda */

Typedefs

- template<typename Value, typename Policy>
  using comppare::set_policy = spec<Value, Policy>

Functions

- template<typename T>
  void comppare::DoNotOptimize (T const &value)

  Prevents the compiler from optimizing away the given value.
- template<typename T>
  void comppare::DoNotOptimize (T &value)

  Prevents the compiler from optimizing away the given value.
- template<typename T>
  void comppare::DoNotOptimize (T &&value)

  Prevents the compiler from optimizing away the given value.
- void comppare::ClobberMemory ()

9.1.1 Detailed Description

This file is the main include file for the ComPPare framework.

9.1.2 Macro Definition Documentation

9.1.2.1 COMPPARE_HOTLOOP_BENCH

```
#define COMPPARE_HOTLOOP_BENCH
```

Value:

```
    /* Warm-up */                                                          \
    auto warmup_t0 = comppare::config::clock_t::now();                     \
    for (std::size_t i = 0; i < comppare::config::warmup_iters(); ++i)     \
        hotloop_body();                                                    \
    auto warmup_t1 = comppare::config::clock_t::now();                     \
    comppare::config::set_warmup_us(warmup_t0, warmup_t1);                 \
                                                                           \
    /* Timed */                                                            \
    comppare::config::reset_roi_us();                                      \
    auto t0 = comppare::config::clock_t::now();                            \
    for (std::size_t i = 0; i < comppare::config::bench_iters(); ++i)      \
        hotloop_body();                                                    \
    auto t1 = comppare::config::clock_t::now();                            \
                                                                           \
    if (comppare::config::get_roi_us() == double(0.0))                     \
        comppare::config::set_roi_us(t0, t1);
```

9.1.2.2 GPU_HOTLOOPSTART

```
#define GPU_HOTLOOPSTART  auto &&hotloop_body = [&]() { /* start of lambda */
```

9.1.2.3 HOTLOOP

```
#define HOTLOOP(
                LOOP_BODY)
```

Value:

```
    HOTLOOPSTART LOOP_BODY HOTLOOPEND
```

9.1.2.4 HOTLOOPEND

```
#define HOTLOOPEND
```

Value:

```
    }                   \
    ; /* end lambda */ \
                        \
    COMPPARE_HOTLOOP_BENCH;
```

9.1.2.5 HOTLOOPSTART

```
#define HOTLOOPSTART  auto &&hotloop_body = [&]() { /* start of lambda */
```

9.1.2.6 MANUAL_TIMER_END

```
#define MANUAL_TIMER_END
```

Value:

```
    auto t_manual_stop = comppare::config::clock_t::now(); \
    SET_ITERATION_TIME(t_manual_stop - t_manual_start);
```

9.1.2.7 MANUAL_TIMER_START

```
#define MANUAL_TIMER_START  auto t_manual_start = comppare::config::clock_t::now();
```

9.1.2.8 SET_ITERATION_TIME

```
#define SET_ITERATION_TIME(
              TIME)
```

Value:

```
    comppare::config::increment_roi_us(TIME);
```

9.2 comppare.hpp

Go to the documentation of this file.

```
00001 /*
00002
00003 Copyright 2025 | Leong Fan FUNG | funglf | stanleyfunglf@gmail.com
00004
00005 Permission is hereby granted, free of charge, to any person obtaining a copy
00006 of this software and associated documentation files (the "Software"), to deal
00007 in the Software without restriction, including without limitation the rights
00008 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00009 copies of the Software, and to permit persons to whom the Software is
00010 furnished to do so, subject to the following conditions:
00011
00012 The above copyright notice and this permission notice shall be included in
00013 all copies or substantial portions of the Software.
00014
00015 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00016 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00017 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00018 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00019 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00020 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00021 SOFTWARE.
00022
00023 */
00024
00030
00031 #pragma once
00032 #include <chrono>
00033 #include <cmath>
00034 #include <functional>
00035 #include <iomanip>
00036 #include <iostream>
00037 #include <string>
00038 #include <vector>
00039 #include <array>
00040 #include <memory>
00041 #include <string_view>
00042 #include <concepts>
00043 #include <stdexcept>
00044
00045 #include <comppare/internal/ansi.hpp>
00046 #include <comppare/internal/config.hpp>
00047 #include <comppare/internal/helper.hpp>
00048 #include <comppare/internal/policy.hpp>
00049 #include <comppare/plugin/plugin.hpp>
00050
00051 #if defined(HAVE_GOOGLE_BENCHMARK) && defined(HAVE_NV_BENCH)
00052 #error "Please only use one Plugin."
00053 #endif
00054
00055 #if defined(HAVE_GOOGLE_BENCHMARK)
00056 #include "comppare/plugin/google_benchmark/google_benchmark.hpp"
00057 #elif defined(HAVE_NV_BENCH)
00058 #include "comppare/plugin/nvbench/nvbench.hpp"
00059 #endif
00060
00065 namespace comppare
00066 {
```

```
00067     /*
00068     DoNotOptimize() and ClobberMemory() are utility functions to prevent compiler optimizations
00069
00070     Reference:
00071     CppCon 2015: Chandler Carruth "Tuning C++: Benchmarks, and CPUs, and Compilers! Oh My!"
00072     Google Benchmark: https://github.com/google/benchmark
00073     */
00074
00075     // Copyright 2015 Google Inc. All rights reserved.
00076     //
00077     // Licensed under the Apache License, Version 2.0 (the "License");
00078     // you may not use this file except in compliance with the License.
00079     // You may obtain a copy of the License at
00080     //
00081     //     http://www.apache.org/licenses/LICENSE-2.0
00082     //
00083     // Unless required by applicable law or agreed to in writing, software
00084     // distributed under the License is distributed on an "AS IS" BASIS,
00085     // WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
00086     // See the License for the specific language governing permissions and
00087     // limitations under the License.
00088
00097     template <typename T>
00098     inline __attribute__((always_inline)) void DoNotOptimize(T const &value)
00099     {
00100         asm volatile("" : : "r,m"(value) : "memory");
00101     }
00102
00111     template <typename T>
00112     inline __attribute__((always_inline)) void DoNotOptimize(T &value)
00113     {
00114 #if defined(__clang__)
00115         asm volatile("" : "+r,m"(value) : : "memory");
00116 #else
00117         asm volatile("" : "+m,r"(value) : : "memory");
00118 #endif
00119     }
00120
00129     template <typename T>
00130     inline __attribute__((always_inline)) void DoNotOptimize(T &&value)
00131     {
00132 #if defined(__clang__)
00133         asm volatile("" : "+r,m"(value) : : "memory");
00134 #else
00135         asm volatile("" : "+m,r"(value) : : "memory");
00136 #endif
00137     }
00138
00139     // This implementation is verbatim from Google Benchmark's benchmark::ClobberMemory(),
00140     // licensed under Apache2.0. No changes have been made.
00141     inline __attribute__((always_inline)) void ClobberMemory()
00142     {
00143         std::atomic_signal_fence(std::memory_order_acq_rel);
00144     }
00145
00146     template <typename Value, typename Policy = void>
00147     struct spec;
00148
00149     template <typename Value>
00150         requires internal::policy::autopolicy::SupportedByAutoPolicy<Value>
00151     struct spec<Value, void>
00152     {
00153         using value_t = Value;
00154         using policy_t = internal::policy::autopolicy::AutoPolicy_t<Value>;
00155     };
00156
00157     template <typename Value, typename Policy>
00158     struct spec<spec<Value, Policy>, void>
00159     {
00160         using value_t = Value;
00161         using policy_t = Policy;
00162     };
00163
00164     template <typename Value, typename Policy>
00165         requires comppare::internal::policy::ErrorPolicy<Value, Policy>
00166     struct spec<Value, Policy>
00167     {
00168         using value_t = Value;
00169         using policy_t = Policy;
00170     };
00171
00172     template <typename Value, typename Policy>
00173     using set_policy = spec<Value, Policy>;
00174
00175     template <typename T>
00176     concept OutSpec =
00177         comppare::internal::policy::ErrorPolicy<
```

```
00178             typename spec<T>::value_t,
00179             typename spec<T>::policy_t>;
00180
00186     template <typename... Inputs>
00187     class InputContext
00188     {
00189     public:
00195         template <OutSpec... Specs>
00196         class OutputContext
00197         {
00198         private:
00199             template <typename S>
00200             using val_t = typename spec<S>::value_t;
00201             template <typename S>
00202             using pol_t = typename spec<S>::policy_t;
00203
00204             // using Outputs = typename Specs::value_t;
00205             // Alias for the user-provided function signature:
00206             // (const Inputs&..., Outputs&..., size_t iterations, double& roi_us)
00207             using Func = std::function<void(const Inputs &..., val_t<Specs> &...)>;
00208
00209             // Holds each input and output type in a tuple
00210             using InTup = std::tuple<Inputs...>;
00211             using OutTup = std::tuple<val_t<Specs>...>;
00212             using PolicyTup = std::tuple<pol_t<Specs>...>;
00213
00214             // reference to output parameter/data
00215             using OutPtr = std::shared_ptr<OutTup>;
00216             using OutVec = std::vector<OutPtr>;
00217
00218             // Tuple to hold all input parameters/data
00219             InTup inputs_;
00220             // Reference output tuple to hold the outputs of the first implementation
00221             OutVec outputs_;
00222             PolicyTup policies_ref_;
00223
00224             // Number of output arguments -- sizeof... is used to get the number of elements in a pack
00225             // https://en.cppreference.com/w/cpp/language/sizeof....html
00226             static constexpr size_t NUM_OUT = sizeof...(Specs);
00227
00228             std::shared_ptr<plugin::Plugin<InTup, OutTup» plugins_;
00229
00235             void register_plugin(const std::shared_ptr<plugin::Plugin<InTup, OutTup>> &p)
00236             {
00237                 if (!plugins_)
00238                     plugins_ = p;
00239                 else if (plugins_ != p)
00240                     throw std::logic_error("Multiple plugins are not supported");
00241             }
00242
00259             struct Impl
00260             {
00261                 std::string name;
00262                 Func fn;
00263
00264                 InTup *inputs_ptr;
00265                 OutputContext *parent_ctx;
00266
00267                 std::unique_ptr<OutTup> plugin_output = nullptr; // output for plugin runs
00268
00269 #ifdef HAVE_GOOGLE_BENCHMARK
00270                 decltype(auto) google_benchmark()
00271                 {
00272                     return attach<plugin::google_benchmark::GoogleBenchmarkPlugin>();
00273                 }
00274 #endif
00275
00276 #ifdef HAVE_NV_BENCH
00277                 decltype(auto) nvbench()
00278                 {
00279                     return attach<plugin::nvbenchplugin::nvbenchPlugin>();
00280                 }
00281 #endif
00282
00289                 template <template <class, class> class Plugin>
00290                     requires comppare::plugin::ValidPlugin<Plugin, InTup, OutTup, Func>
00291                 decltype(auto) attach()
00292                 {
00293                     auto adp = Plugin<InTup, OutTup>::instance();
00294
00295                     parent_ctx->register_plugin(adp);
00296
00297                     plugin_output = std::make_unique<OutTup>();
00298
00299                     return adp->register_impl(name, fn, *inputs_ptr, *plugin_output);
00300                 }
00301             };
```

```
00302
00303             // Vector to hold all implementations
00304             std::vector<Impl> impls_;
00305
00306             // helpers -----------------------------------------------------------
00307             template <std::size_t I>
00308             static constexpr auto spec_metric_count() { return std::tuple_element_t<I,
      PolicyTup>::metric_count(); }
00309             template <std::size_t I>
00310             static constexpr auto spec_metric_name(std::size_t m) { return std::tuple_element_t<I,
      PolicyTup>::metric_name(m); }
00311
00312             static constexpr int PRINT_COL_WIDTH = 20;
00313
00314             void print_header() const
00315             {
00316                 std::cout « std::left « comppare::internal::ansi::BOLD
00317                           « "*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=\n============ "
00318                           « comppare::internal::ansi::ITALIC("ComPPare Framework")
00319                           « " ============\n=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*"
00320                           « comppare::internal::ansi::BOLD_OFF « "\n\n";
00321                 std::cout
00322                     « std::left « std::setw(30) « "Number of implementations: "
00323                     « std::right « std::setw(10) « impls_.size() « "\n"
00324                     « std::left « std::setw(30) « "Warmup iterations: "
00325                     « std::right « std::setw(10) « comppare::config::warmup_iters() « "\n"
00326                     « std::left « std::setw(30) « "Benchmark iterations: "
00327                     « std::right « std::setw(10) « comppare::config::bench_iters() « "\n"
00328                     « std::left «
      comppare::internal::ansi::BOLD("=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*") « "\n\n";
00329
00330                 // Print header for the output table
00331                 std::cout « comppare::internal::ansi::UNDERLINE « comppare::internal::ansi::BOLD
00332                           « std::left
00333                           « std::setw(PRINT_COL_WIDTH) « "Implementation"
00334                           « std::right
00335                           « std::setw(PRINT_COL_WIDTH) « "ROI µs/Iter"
00336                           « std::setw(PRINT_COL_WIDTH) « "Func µs"
00337                           « std::setw(PRINT_COL_WIDTH) « "Ovhd µs";
00338
00339                 // prints metric header
00340                 auto &&_print_metric_header = [this]<std::size_t I>()
00341                 {
00342                     for (std::size_t m = 0; m < this->template spec_metric_count<I>(); ++m)
00343                     {
00344                         std::cout « std::setw(PRINT_COL_WIDTH)
00345                                   « comppare::internal::ansi::UNDERLINE(
00346                                         comppare::internal::ansi::BOLD(
00347                                             std::string(this->template spec_metric_name<I>(m)) + "["
      + std::to_string(I) + "]"));
00348                     }
00349                 };
00350                 // lambda to call print metric header across each metric by unpacking I
00351                 [&]<std::size_t... I>(std::index_sequence<I...>)
00352                 {
00353                     (_print_metric_header.template operator()<I>(), ...);
00354                 }(std::make_index_sequence<NUM_OUT>{});
00355
00356                 std::cout « std::endl;
00357             }
00358
00359             void compute_errors(PolicyTup &errs, const OutTup &test, const OutTup &ref)
00360             {
00361                 auto &&_compute_errors = [&]<std::size_t I>()
00362                 { comppare::internal::policy::compute_error(std::get<I>(errs), std::get<I>(test),
      std::get<I>(ref)); };
00363
00364                 [&]<std::size_t... I>(std::index_sequence<I...>)
00365                 {
00366                     (_compute_errors.template operator()<I>(), ...);
00367                 }(std::make_index_sequence<NUM_OUT>{});
00368             }
00369
00370             bool any_fail(const PolicyTup &errs) const
00371             {
00372                 auto &&_any_fail = [&]<std::size_t I>() -> bool
00373                 {
00374                     return comppare::internal::policy::is_fail(std::get<I>(errs));
00375                 };
00376
00377                 return [&]<std::size_t... I>(std::index_sequence<I...>) -> bool
00378                 {
00379                     bool fail = false;
00380                     ((fail |= _any_fail.template operator()<I>()), ...);
00381                     return fail;
00382                 }(std::make_index_sequence<NUM_OUT>{});
00383             }
```

```
00384
00385             void print_metrics(const PolicyTup &errs) const
00386             {
00387                 auto &&_print_metrics = [this, &errs]<std::size_t I>()
00388                 {
00389                     for (std::size_t m = 0; m < spec_metric_count<I>(); ++m)
00390                         std::cout « std::setw(PRINT_COL_WIDTH) « std::scientific «
      std::get<I>(errs).metric(m);
00391                 };
00392
00393                 [&]<std::size_t... I>(std::index_sequence<I...>)
00394                 {
00395                     (_print_metrics.template operator()<I>(), ...);
00396                 }(std::make_index_sequence<NUM_OUT>{});
00397             }
00398
00399             inline OutPtr get_output_by_index_(const size_t idx) const
00400             {
00401                 if (outputs_.empty())
00402                     throw std::logic_error("run() has not been executed");
00403                 if (idx >= outputs_.size())
00404                     throw std::out_of_range("Index out of range for outputs");
00405
00406                 return outputs_[idx];
00407             }
00408
00409             inline OutPtr get_output_by_name_(const std::string_view name) const
00410             {
00411                 if (outputs_.empty())
00412                     throw std::logic_error("run() has not been executed");
00413                 for (size_t i = 0; i < impls_.size(); ++i)
00414                 {
00415                     if (impls_[i].name == name)
00416                         return outputs_[i];
00417                 }
00418                 std::stringstream os;
00419                 os « "Output with name '" « name « "' not found";
00420                 throw std::invalid_argument(os.str());
00421             }
00422
00423             void unpack_output_(const OutTup &outtup, val_t<Specs> *...outs) const
00424             {
00425                 std::apply(
00426                     [&](auto &...outtup_elem)
00427                     {
00428                         ((*outs = outtup_elem), ...);
00429                     },
00430                     outtup);
00431             }
00432
00433         public:
00434             // Constructor to initialize the OutputContext with inputs
00435             // This is used to hold and pass the same input arguments/data for all implementations
00436             // The inputs are perfectly forwarded -- for instance taking ownership when moving
00437             template <typename... Ins>
00438             explicit OutputContext(Ins &&...ins)
00439                 : inputs_(std::forward<Ins>(ins)...) {}
00440
00441             OutputContext(const OutputContext &other) = delete;
00442             OutputContext &operator=(const OutputContext &other) = delete;
00443             OutputContext(OutputContext &&other) = delete;
00444             OutputContext &operator=(OutputContext &&other) = delete;
00445
00446             // Function to set a reference implementation
00447             template <typename F>
00448                 requires std::invocable<F, const Inputs &..., val_t<Specs> &...>
00449             Impl &set_reference(std::string name, F &&f)
00450             {
00451                 impls_.insert(impls_.begin(), {std::move(name), Func(std::forward<F>(f)), &inputs_,
      this});
00452                 return impls_.front();
00453             }
00454
00455             // Function to add an implementation to the comparison
00456             template <typename F>
00457                 requires std::invocable<F, const Inputs &..., val_t<Specs> &...>
00458             Impl &add(std::string name, F &&f)
00459             {
00460                 impls_.push_back({std::move(name), Func(std::forward<F>(f)), &inputs_, this});
00461                 return impls_.back();
00462             }
00463
00464             /*
00465             Getter for the output results
00466             */
00467
00468             // returns a shared pointer to the reference output
```

```
00469             // std::shared_ptr<std::tuple<Outputs...>>
00470             const OutPtr get_reference_output() const
00471             {
00472                 return get_output_by_index_(0);
00473             }
00474
00475             const OutPtr get_output(const size_t idx) const
00476             {
00477                 return get_output_by_index_(idx);
00478             }
00479
00480             const OutPtr get_output(const std::string_view name) const
00481             {
00482                 return get_output_by_name_(name);
00483             }
00484
00485             // Unpack the outputs into the provided pointers
00486             template <typename U = void>
00487                 requires(sizeof...(Specs) > 0)
00488             void get_reference_output(val_t<Specs> *...outs) const
00489             {
00490                 const auto &outtup = *get_output_by_index_(0);
00491                 unpack_output_(outtup, outs...);
00492             }
00493
00494             template <typename U = void>
00495                 requires(sizeof...(Specs) > 0)
00496             void get_output(const size_t idx, val_t<Specs> *...outs) const
00497             {
00498                 const auto &outtup = *get_output_by_index_(idx);
00499                 unpack_output_(outtup, outs...);
00500             }
00501
00502             template <typename U = void>
00503                 requires(sizeof...(Specs) > 0)
00504             void get_output(const std::string_view name, val_t<Specs> *...outs) const
00505             {
00506                 const auto &outtup = *get_output_by_name_(name);
00507                 unpack_output_(outtup, outs...);
00508             }
00509
00510             /*
00511             Runs the comparison for all added implementations.
00512             Optional Arguments:
00513             - argc: Number of command line arguments
00514             - argv: Array of command line arguments
00515             This function will parse the command line arguments to set warmup, benchmark iterations
    and tolerance for floating point errors.
00516             */
00517             void run(int argc = 0,
00518                      char **argv = nullptr)
00519             {
00520                 comppare::internal::helper::parse_args(argc, argv);
00521
00522                 if (impls_.empty())
00523                 {
00524                     std::cerr « "\n*----------*\nNo implementations added to the ComPPare
    Framework.\n*----------*\n";
00525                     return;
00526                 }
00527
00528                 outputs_.reserve(impls_.size()); // reserve space for outputs -- resize and use index
    works too.
00529
00530                 print_header();
00531
00532                 // Main loop to iterate over all implementations
00533                 for (size_t k = 0; k < impls_.size(); ++k)
00534                 {
00535                     // Get the current implementation
00536                     auto &impl = impls_[k];
00537
00538                     OutTup outs;
00539
00540                     double func_duration;
00541                     /*
00542                     use std::apply to unpack the inputs and outputs completely to do 1 function call
    of the implementation
00543                     this is equivalent to calling:
00544                     impl.fn(inputs[0], inputs[1], ..., outputs[0], outputs[1], iters, roi_us);
00545                     */
00546                     std::apply([&](auto const &...in)
00547                                { std::apply(
00548                                      [&](auto &...out)
00549                                      {
00550                                          auto func_start = comppare::config::clock_t::now();
00551                                          impl.fn(in..., out...);
```

```
00552                                       auto func_end = comppare::config::clock_t::now();
00553                                       func_duration = std::chrono::duration<double,
    std::micro>(func_end - func_start).count();
00554                                   },
00555                                   outs); },
00556                               inputs_);
00557
00558                     // Calculate the time taken by the function in microseconds
00559                     double roi_us = comppare::config::get_roi_us();
00560                     double warmup_us = comppare::config::get_warmup_us();
00561                     double func_us = func_duration - warmup_us;
00562                     double ovhd_us = func_us - roi_us;
00563
00564                     roi_us /= static_cast<double>(comppare::config::bench_iters());
00565
00566                     PolicyTup errs{};
00567                     if (k)
00568                     {
00569                         compute_errors(errs, outs, *outputs_[0]);
00570                     }
00571                     outputs_.push_back(std::make_shared<OutTup>(std::move(outs)));
00572                     // print row
00573                     std::cout « comppare::internal::ansi::RESET
00574                               « std::left « std::setw(PRINT_COL_WIDTH) «
    comppare::internal::ansi::GREEN(impl.name)
00575                               « std::fixed « std::setprecision(2) « std::right
00576                               « comppare::internal::ansi::YELLOW
00577                               « std::setw(PRINT_COL_WIDTH) « roi_us
00578                               « comppare::internal::ansi::DIM
00579                               « std::setw(PRINT_COL_WIDTH) « func_us
00580                               « std::setw(PRINT_COL_WIDTH) « ovhd_us
00581                               « comppare::internal::ansi::RESET;
00582
00583                     print_metrics(errs);
00584                     if (k && any_fail(errs))
00585                         std::cout « comppare::internal::ansi::BG_RED("<-- FAIL");
00586                     std::cout « '\n';
00587
00588                 } /* for impls */
00589
00590                 comppare::current_state::set_using_plugin(true);
00591                 if (plugins_)
00592                 {
00593                     plugins_->initialize(argc, argv);
00594                     plugins_->run();
00595                 }
00596                 comppare::current_state::set_using_plugin(false);
00597             } /* run */
00598         }; /* OutputContext */
00599     }; /* InputContext */
00600 } // namespace comppare
00601
00602 #define HOTLOOPSTART \
00603     auto &&hotloop_body = [&]() { /* start of lambda */
00604
00605 #define COMPPARE_HOTLOOP_BENCH                                          \
00606     /* Warm-up */                                                       \
00607     auto warmup_t0 = comppare::config::clock_t::now();                  \
00608     for (std::size_t i = 0; i < comppare::config::warmup_iters(); ++i)  \
00609         hotloop_body();                                                 \
00610     auto warmup_t1 = comppare::config::clock_t::now();                  \
00611     comppare::config::set_warmup_us(warmup_t0, warmup_t1);              \
00612                                                                         \
00613     /* Timed */                                                         \
00614     comppare::config::reset_roi_us();                                   \
00615     auto t0 = comppare::config::clock_t::now();                         \
00616     for (std::size_t i = 0; i < comppare::config::bench_iters(); ++i)   \
00617         hotloop_body();                                                 \
00618     auto t1 = comppare::config::clock_t::now();                         \
00619                                                                         \
00620     if (comppare::config::get_roi_us() == double(0.0))                  \
00621         comppare::config::set_roi_us(t0, t1);
00622
00623 #ifdef PLUGIN_HOTLOOP_BENCH
00624 #define HOTLOOPEND                                  \
00625     }                                               \
00626     ; /* end lambda */                              \
00627                                                     \
00628     if (comppare::current_state::using_plugin())    \
00629     {                                               \
00630         PLUGIN_HOTLOOP_BENCH;                       \
00631     }                                               \
00632     else                                            \
00633     {                                               \
00634         COMPPARE_HOTLOOP_BENCH;                     \
00635     }
00636 #else
```

```
00637 #define HOTLOOPEND     \
00638     }                  \
00639     ; /* end lambda */ \
00640                        \
00641     COMPPARE_HOTLOOP_BENCH;
00642 #endif
00643
00644 #define HOTLOOP(LOOP_BODY) \
00645     HOTLOOPSTART LOOP_BODY HOTLOOPEND
00646
00647 #define MANUAL_TIMER_START \
00648     auto t_manual_start = comppare::config::clock_t::now();
00649
00650 #define MANUAL_TIMER_END                                    \
00651     auto t_manual_stop = comppare::config::clock_t::now(); \
00652     SET_ITERATION_TIME(t_manual_stop - t_manual_start);
00653
00654 #ifdef PLUGIN_HOTLOOP_BENCH
00655 #define SET_ITERATION_TIME(TIME)                  \
00656     if (comppare::current_state::using_plugin())  \
00657     {                                             \
00658         PLUGIN_SET_ITERATION_TIME(TIME);          \
00659     }                                             \
00660     else                                          \
00661     {                                             \
00662         comppare::config::increment_roi_us(TIME); \
00663     }
00664 #else
00665 #define SET_ITERATION_TIME(TIME) \
00666     comppare::config::increment_roi_us(TIME);
00667 #endif
00668
00669 #define GPU_HOTLOOPSTART \
00670     auto &&hotloop_body = [&]() { /* start of lambda */
00671
00672 #if defined(__CUDACC__)
00673 #define GPU_HOTLOOPEND                                                  \
00674     }                                                                   \
00675     ; /* end lambda */                                                  \
00676     /* Warm-up */                                                       \
00677     cudaEvent_t start_, stop_;                                          \
00678     cudaEventCreate(&start_);                                           \
00679     cudaEventCreate(&stop_);                                            \
00680     cudaEventRecord(start_);                                            \
00681     for (std::size_t i = 0; i < comppare::config::warmup_iters(); ++i) \
00682         hotloop_body();                                                 \
00683     cudaEventRecord(stop_);                                             \
00684     cudaEventSynchronize(stop_);                                        \
00685     float ms_warmup_;                                                   \
00686     cudaEventElapsedTime(&ms_warmup_, start_, stop_);                   \
00687     comppare::config::set_warmup_us(1e3 * ms_warmup_);                  \
00688                                                                         \
00689     /* Timed */                                                         \
00690     comppare::config::reset_roi_us();                                   \
00691     cudaEventRecord(start_);                                            \
00692     for (std::size_t i = 0; i < comppare::config::bench_iters(); ++i)  \
00693         hotloop_body();                                                 \
00694     cudaEventRecord(stop_);                                             \
00695     cudaEventSynchronize(stop_);                                        \
00696     float ms_;                                                          \
00697     cudaEventElapsedTime(&ms_, start_, stop_);                          \
00698     if (comppare::config::get_roi_us() == double(0.0))                  \
00699         comppare::config::set_roi_us(1e3 * ms_);                        \
00700     cudaEventDestroy(start_);                                           \
00701     cudaEventDestroy(stop_);
00702
00703 #elif defined(__HIPCC__)
00704 #define GPU_HOTLOOPEND                                                  \
00705     }                                                                   \
00706     ; /* end lambda */                                                  \
00707     /* Warm-up */                                                       \
00708     hipEvent_t start_, stop_;                                           \
00709     hipEventCreate(&start_);                                            \
00710     hipEventCreate(&stop_);                                             \
00711     hipEventRecord(start_);                                             \
00712     for (std::size_t i = 0; i < comppare::config::warmup_iters(); ++i) \
00713         hotloop_body();                                                 \
00714     hipEventRecord(stop_);                                              \
00715     hipEventSynchronize(stop_);                                         \
00716     float ms_warmup_;                                                   \
00717     hipEventElapsedTime(&ms_warmup_, start_, stop_);                    \
00718     comppare::config::set_warmup_us(1e3 * ms_warmup_);                  \
00719                                                                         \
00720     /* Timed */                                                         \
00721     comppare::config::reset_roi_us();                                   \
00722     hipEventRecord(start_);                                             \
00723     for (std::size_t i = 0; i < comppare::config::bench_iters(); ++i)  \
```

```
00724         hotloop_body();                                                 \
00725     hipEventRecord(stop_);                                              \
00726     hipEventSynchronize(stop_);                                         \
00727     float ms_;                                                          \
00728     hipEventElapsedTime(&ms_, start_, stop_);                           \
00729     if (comppare::config::get_roi_us() == double(0.0))                  \
00730         comppare::config::set_roi_us(1e3 * ms_);                        \
00731     hipEventDestroy(start_);                                            \
00732     hipEventDestroy(stop_);
00733 #endif
00734
00735 #if defined(__CUDACC__)
00736 #define GPU_MANUAL_TIMER_START                          \
00737     cudaEvent_t start_manual_timer, stop_manual_timer; \
00738     cudaEventCreate(&start_manual_timer);               \
00739     cudaEventCreate(&stop_manual_timer);                \
00740     cudaEventRecord(start_manual_timer);
00741
00742 #define GPU_MANUAL_TIMER_END                                                    \
00743     cudaEventRecord(stop_manual_timer);                                         \
00744     cudaEventSynchronize(stop_manual_timer);                                    \
00745     float ms_manual;                                                            \
00746     cudaEventElapsedTime(&ms_manual, start_manual_timer, stop_manual_timer); \
00747     SET_ITERATION_TIME(1e3 * ms_manual);                                        \
00748     cudaEventDestroy(start_manual_timer);                                       \
00749     cudaEventDestroy(stop_manual_timer);
00750
00751 #elif defined(__HIPCC__)
00752 #define GPU_MANUAL_TIMER_START                          \
00753     hipEvent_t start_manual_timer, stop_manual_timer; \
00754     hipEventCreate(&start_manual_timer);               \
00755     hipEventCreate(&stop_manual_timer);                \
00756     hipEventRecord(start_manual_timer);
00757
00758 #define GPU_MANUAL_TIMER_END                                                    \
00759     hipEventRecord(stop_manual_timer);                                          \
00760     hipEventSynchronize(stop_manual_timer);                                     \
00761     float ms_manual;                                                            \
00762     hipEventElapsedTime(&ms_manual, start_manual_timer, stop_manual_timer); \
00763     SET_ITERATION_TIME(1e3 * ms_manual);                                        \
00764     hipEventDestroy(start_manual_timer);                                        \
00765     hipEventDestroy(stop_manual_timer);
00766
00767 #endif
```

9.3 include/comppare/internal/ansi.hpp File Reference

```
#include <ostream>
#include <sstream>
#include <string>
#include <type_traits>
#include <concepts>
#include <utility>
#include <comppare/internal/concepts.hpp>
```

Classes

- class comppare::internal::ansi::AnsiWrapper< T >

Namespaces

- namespace comppare

  ComPPare framework main namespace.
- namespace comppare::internal
- namespace comppare::internal::ansi

Macros

- #define ANSI_DEFINE(NAME, ON, OFF)

Typedefs

- template<comppare::internal::concepts::Streamable T>
  using comppare::internal::ansi::AW = AnsiWrapper<std::decay_t<T>>

Functions

- comppare::internal::ansi::ANSI_DEFINE (RESET, "0", "0")
- comppare::internal::ansi::ANSI_DEFINE (BOLD, "1", "22")
- comppare::internal::ansi::ANSI_DEFINE (DIM, "2", "22")
- comppare::internal::ansi::ANSI_DEFINE (ITALIC, "3", "23")
- comppare::internal::ansi::ANSI_DEFINE (UNDERLINE, "4", "24")
- comppare::internal::ansi::ANSI_DEFINE (BLINK, "5", "25")
- comppare::internal::ansi::ANSI_DEFINE (REVERSE, "7", "27")
- comppare::internal::ansi::ANSI_DEFINE (HIDDEN, "8", "28")
- comppare::internal::ansi::ANSI_DEFINE (STRIKE, "9", "29")
- comppare::internal::ansi::ANSI_DEFINE (BLACK, "30", "39")
- comppare::internal::ansi::ANSI_DEFINE (RED, "31", "39")
- comppare::internal::ansi::ANSI_DEFINE (GREEN, "32", "39")
- comppare::internal::ansi::ANSI_DEFINE (YELLOW, "33", "39")
- comppare::internal::ansi::ANSI_DEFINE (BLUE, "34", "39")
- comppare::internal::ansi::ANSI_DEFINE (MAGENTA, "35", "39")
- comppare::internal::ansi::ANSI_DEFINE (CYAN, "36", "39")
- comppare::internal::ansi::ANSI_DEFINE (WHITE, "37", "39")
- comppare::internal::ansi::ANSI_DEFINE (BRIGHT_BLACK, "90", "39")
- comppare::internal::ansi::ANSI_DEFINE (BRIGHT_RED, "91", "39")
- comppare::internal::ansi::ANSI_DEFINE (BRIGHT_GREEN, "92", "39")
- comppare::internal::ansi::ANSI_DEFINE (BRIGHT_YELLOW, "93", "39")
- comppare::internal::ansi::ANSI_DEFINE (BRIGHT_BLUE, "94", "39")
- comppare::internal::ansi::ANSI_DEFINE (BRIGHT_MAGENTA, "95", "39")
- comppare::internal::ansi::ANSI_DEFINE (BRIGHT_CYAN, "96", "39")
- comppare::internal::ansi::ANSI_DEFINE (BRIGHT_WHITE, "97", "39")
- comppare::internal::ansi::ANSI_DEFINE (BG_BLACK, "40", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_RED, "41", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_GREEN, "42", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_YELLOW, "43", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_BLUE, "44", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_MAGENTA, "45", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_CYAN, "46", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_WHITE, "47", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_BRIGHT_BLACK, "100", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_BRIGHT_RED, "101", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_BRIGHT_GREEN, "102", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_BRIGHT_YELLOW, "103", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_BRIGHT_BLUE, "104", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_BRIGHT_MAGENTA, "105", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_BRIGHT_CYAN, "106", "49")
- comppare::internal::ansi::ANSI_DEFINE (BG_BRIGHT_WHITE, "107", "49")

9.3.1 Macro Definition Documentation

9.3.1.1 ANSI_DEFINE

```
#define ANSI_DEFINE(
                NAME,
                ON,
                OFF)
```

9.4 ansi.hpp

Go to the documentation of this file.

```
00001 /*
00002
00003 Copyright 2025 | Leong Fan FUNG | funglf | stanleyfunglf@gmail.com
00004
00005 Permission is hereby granted, free of charge, to any person obtaining a copy
00006 of this software and associated documentation files (the "Software"), to deal
00007 in the Software without restriction, including without limitation the rights
00008 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00009 copies of the Software, and to permit persons to whom the Software is
00010 furnished to do so, subject to the following conditions:
00011
00012 The above copyright notice and this permission notice shall be included in
00013 all copies or substantial portions of the Software.
00014
00015 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00016 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00017 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00018 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00019 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00020 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00021 SOFTWARE.
00022
00023 */
00024 #pragma once
00025 #include <ostream>
00026 #include <sstream>
00027 #include <string>
00028 #include <type_traits>
00029 #include <concepts>
00030 #include <utility>
00031
00032 #include <comppare/internal/concepts.hpp>
00033
00034 /*
00035 ECMA-48 "Select Graphic Rendition" (SGR) sequences are defined here:
00036 https://man7.org/linux/man-pages/man4/console_codes.4.html
00037 */
00038 namespace comppare::internal::ansi
00039 {
00040     /*
00041     Wrapper Class to apply ANSI styles to any streamable type. [template <Streamable T>]
00042     It wraps the value and applies the "ON" and "OFF" ANSI codes around it
00043     */
00044     template <comppare::internal::concepts::Streamable T>
00045     class AnsiWrapper
00046     {
00047         const char *on_;  // "on" code
00048         const char *off_; // "off" code
00049         T val_;           // the value to be wrapped
00050
00051     public:
00052         AnsiWrapper(const char *on, const char *off, T v)
00053             : on_(on), off_(off), val_(std::move(v)) {}
00054
00055         /*
00056         Overloaded operator« to stream the value with ANSI codes.
00057         It takes in the output stream state and applies it to the value.
00058         */
00059         friend std::ostream &operator«(std::ostream &os, AnsiWrapper const &w)
00060         {
00061             // Save the current state of the stream
00062             std::ios saved(nullptr);
00063             saved.copyfmt(os);
```

```
00064
00065             // convert the value into a string
00066             std::ostringstream tmp;
00067             tmp.copyfmt(os);
00068             tmp « w.val_;
00069             std::string body = std::move(tmp).str();
00070
00071             // apply the ANSI codes
00072             os.width(0);
00073             os « w.on_ « body « w.off_;
00074
00075             // Restore the original state of the stream -- formatting, precision, etc.
00076             os.copyfmt(saved);
00077             return os;
00078         }
00079     };
00080     template <comppare::internal::concepts::Streamable T>
00081     using AW = AnsiWrapper<std::decay_t<T>>;
00082
00083 /*
00084 Generator Macro for each ANSI style and color with the AnsiWrapper class.
00085 */
00086 #define ANSI_DEFINE(NAME, ON, OFF)
    \
00087     /* Actual ANSI codes for the style/color (eg: BLACK_ON_CODE = "\033[30m") */
    \
00088     inline constexpr const char *NAME##_ON_CODE = "\033[" ON "m";
    \
00089     inline constexpr const char *NAME##_OFF_CODE = "\033[" OFF "m";
    \
00090
    \
00091     /* Wrapper class for the different usage patterns */
    \
00092     /*
    https://learn.microsoft.com/en-us/cpp/standard-library/overloading-the-output-operator-for-your-own-classes?view=msvc-1
    */ \
00093     class NAME##ON_t
    \
00094     {
    \
00095     public:
    \
00096         /* Usage: std::cout « comppare::internal::ansi::BOLD « "Hello world"; */
    \
00097         friend std::ostream &operator«(std::ostream &os, NAME##ON_t)
    \
00098         {
    \
00099             /* Save the current state of the stream and apply the ANSI code */
    \
00100             std::ios saved(nullptr);
    \
00101             saved.copyfmt(os);
    \
00102             os.width(0);
    \
00103             os « NAME##_ON_CODE;
    \
00104             os.copyfmt(saved);
    \
00105             return os;
    \
00106         }
    \
00107
    \
00108         /* Usage: std::cout « comppare::internal::ansi::BOLD("Hello world") */
    \
00109         template <comppare::internal::concepts::Streamable T>
    \
00110         auto operator()(T &&v) const
    \
00111         {
    \
00112             return AW<T>(NAME##_ON_CODE, NAME##_OFF_CODE, std::forward<T>(v));
    \
00113         }
    \
00114     };
    \
00115     /* Instance of the ON class */
    \
00116     inline constexpr NAME##ON_t NAME{};
    \
00117     class NAME##_OFF_t
    \
```

```
    { \
    public: \
        /* Usage: std::cout « ... « comppare::internal::ansi::BOLD_OFF; */ \
        friend std::ostream &operator«(std::ostream &os, NAME##_OFF_t) \
        { \
            /* Save the current state of the stream and apply the ANSI code */ \
            std::ios saved(nullptr); \
            saved.copyfmt(os); \
            os.width(0); \
            os « NAME##_OFF_CODE; \
            os.copyfmt(saved); \
            return os; \
        } \
    }; \
    /* Instance of the OFF class */ \
    inline constexpr NAME##_OFF_t NAME##_OFF{};

    /*
    Below are the ANSI escape codes for various styles and colors.
    Each style has a corresponding ON and OFF code.
    */

    /* STYLES 0-9 */
    ANSI_DEFINE(RESET,      "0", "0");
    ANSI_DEFINE(BOLD,       "1", "22");
    ANSI_DEFINE(DIM,        "2", "22");
    ANSI_DEFINE(ITALIC,     "3", "23");
    ANSI_DEFINE(UNDERLINE,  "4", "24");
    ANSI_DEFINE(BLINK,      "5", "25");
    ANSI_DEFINE(REVERSE,    "7", "27");
    ANSI_DEFINE(HIDDEN,     "8", "28");
    ANSI_DEFINE(STRIKE,     "9", "29");

    /* FOREGROUND (TEXT) COLOURS 30-37/90-97 */
    ANSI_DEFINE(BLACK,      "30", "39");
    ANSI_DEFINE(RED,        "31", "39");
    ANSI_DEFINE(GREEN,      "32", "39");
    ANSI_DEFINE(YELLOW,     "33", "39");
    ANSI_DEFINE(BLUE,       "34", "39");
    ANSI_DEFINE(MAGENTA,    "35", "39");
    ANSI_DEFINE(CYAN,       "36", "39");
    ANSI_DEFINE(WHITE,      "37", "39");

    ANSI_DEFINE(BRIGHT_BLACK,   "90", "39");
    ANSI_DEFINE(BRIGHT_RED,     "91", "39");
    ANSI_DEFINE(BRIGHT_GREEN,   "92", "39");
    ANSI_DEFINE(BRIGHT_YELLOW,  "93", "39");
    ANSI_DEFINE(BRIGHT_BLUE,    "94", "39");
    ANSI_DEFINE(BRIGHT_MAGENTA, "95", "39");
    ANSI_DEFINE(BRIGHT_CYAN,    "96", "39");
    ANSI_DEFINE(BRIGHT_WHITE,   "97", "39");

    /* BACKGROUND COLOURS (40-47 / 100-107) */
    ANSI_DEFINE(BG_BLACK,   "40", "49");
    ANSI_DEFINE(BG_RED,     "41", "49");
    ANSI_DEFINE(BG_GREEN,   "42", "49");
    ANSI_DEFINE(BG_YELLOW,  "43", "49");
    ANSI_DEFINE(BG_BLUE,    "44", "49");
    ANSI_DEFINE(BG_MAGENTA, "45", "49");
    ANSI_DEFINE(BG_CYAN,    "46", "49");
    ANSI_DEFINE(BG_WHITE,   "47", "49");

    ANSI_DEFINE(BG_BRIGHT_BLACK,    "100", "49");
    ANSI_DEFINE(BG_BRIGHT_RED,      "101", "49");
    ANSI_DEFINE(BG_BRIGHT_GREEN,    "102", "49");
    ANSI_DEFINE(BG_BRIGHT_YELLOW,   "103", "49");
    ANSI_DEFINE(BG_BRIGHT_BLUE,     "104", "49");
    ANSI_DEFINE(BG_BRIGHT_MAGENTA,  "105", "49");
    ANSI_DEFINE(BG_BRIGHT_CYAN,     "106", "49");
    ANSI_DEFINE(BG_BRIGHT_WHITE,    "107", "49");

#undef ANSI_DEFINE
```

```
00190 }
```

9.5 include/comppare/internal/concepts.hpp File Reference

```
#include <concepts>
#include <type_traits>
#include <ranges>
#include <string>
#include <ostream>
```

Namespaces

- namespace comppare

  ComPPare framework main namespace.
- namespace comppare::internal
- namespace comppare::internal::concepts

Concepts

- concept comppare::internal::concepts::Streamable
- concept comppare::internal::concepts::FloatingPoint
- concept comppare::internal::concepts::Integral
- concept comppare::internal::concepts::Arithmetic
- concept comppare::internal::concepts::String
- concept comppare::internal::concepts::Void
- concept comppare::internal::concepts::RangeOfArithmetic

9.6 concepts.hpp

Go to the documentation of this file.

```
00001 /*
00002
00003 Copyright 2025 | Leong Fan FUNG | funglf | stanleyfunglf@gmail.com
00004
00005 Permission is hereby granted, free of charge, to any person obtaining a copy
00006 of this software and associated documentation files (the "Software"), to deal
00007 in the Software without restriction, including without limitation the rights
00008 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00009 copies of the Software, and to permit persons to whom the Software is
00010 furnished to do so, subject to the following conditions:
00011
00012 The above copyright notice and this permission notice shall be included in
00013 all copies or substantial portions of the Software.
00014
00015 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00016 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00017 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00018 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00019 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00020 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00021 SOFTWARE.
00022
00023 */
00024 #pragma once
00025 #include <concepts>
00026 #include <type_traits>
00027 #include <ranges>
00028 #include <string>
```

```
00029 #include <ostream>
00030
00031 namespace comppare::internal::concepts
00032 {
00033     /*
00034     Concept for valid type that can be streamed to an output stream.
00035     For example, std::string, int, double, etc; NOT std::vector<int>, std::map<int, int>, etc.
00036     */
00037     template <typename T>
00038     concept Streamable =
00039         requires(std::ostream &os, T v) { { os « v } -> std::same_as<std::ostream&>; };
00040
00041     template <typename T>
00042     concept FloatingPoint = std::floating_point<std::remove_cvref_t<T»;
00043
00044     template <typename T>
00045     concept Integral = std::integral<std::remove_cvref_t<T»;
00046
00047     template <typename T>
00048     concept Arithmetic = FloatingPoint<T> || Integral<T>;
00049
00050     template <typename T>
00051     concept String = std::same_as<std::remove_cvref_t<T>, std::string>;
00052
00053     template <typename T>
00054     concept Void = std::is_void_v<std::remove_cvref_t<T»;
00055
00056     // Concept for a range of arithmetic types, excluding strings
00057     template <typename R>
00058     concept RangeOfArithmetic =
00059         std::ranges::forward_range<std::remove_cvref_t<R» &&
      Arithmetic<std::remove_cvref_t<std::ranges::range_value_t<std::remove_cvref_t<R>»> &&
      (!String<std::remove_cvref_t<R>>);
00060
00061 }
```

9.7 include/comppare/internal/config.hpp File Reference

```
#include <chrono>
#include <utility>
#include <string_view>
```

Classes

- class comppare::config
- class comppare::current_state

Namespaces

- namespace comppare

  ComPPare framework main namespace.

9.8 config.hpp

Go to the documentation of this file.

```
00001 /*
00002
00003 Copyright 2025 | Leong Fan FUNG | funglf | stanleyfunglf@gmail.com
00004
00005 Permission is hereby granted, free of charge, to any person obtaining a copy
00006 of this software and associated documentation files (the "Software"), to deal
00007 in the Software without restriction, including without limitation the rights
00008 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
```

```
00009 copies of the Software, and to permit persons to whom the Software is
00010 furnished to do so, subject to the following conditions:
00011
00012 The above copyright notice and this permission notice shall be included in
00013 all copies or substantial portions of the Software.
00014
00015 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00016 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00017 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00018 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00019 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00020 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00021 SOFTWARE.
00022
00023 */
00024 #pragma once
00025 #include <chrono>
00026 #include <utility>
00027 #include <string_view>
00028
00029 namespace comppare
00030 {
00031     /*
00032     Singleton Class for configuration settings
00033     Could be global struct -- currently using singleton to ensure.
00034     */
00035     class config
00036     {
00037     public:
00038         using clock_t = std::chrono::steady_clock;
00039         using time_point_t = std::chrono::time_point<clock_t>;
00040
00041         config(const config &) = delete;
00042         config(config &&) = delete;
00043         config &operator=(const config &) = delete;
00044         config &operator=(config &&) = delete;
00045
00046         static uint64_t warmup_iters() { return instance().warmup_iters_; }
00047         static void set_warmup_iters(uint64_t v) { instance().warmup_iters_ = v; }
00048
00049         static uint64_t bench_iters() { return instance().bench_iters_; }
00050         static void set_bench_iters(uint64_t v) { instance().bench_iters_ = v; }
00051
00052         static void reset_roi_us() { instance().roi_ = double(0.0); }
00053
00054         static void set_roi_us(const time_point_t &start, const time_point_t &end) { instance().roi_ =
      std::chrono::duration<double, std::micro>(end - start).count(); }
00055         static void set_roi_us(const float start, const float end) { instance().roi_ =
      static_cast<double>(end - start); }
00056         static void set_roi_us(const double start, const double end) { instance().roi_ = end - start;
      }
00057
00058         template <typename Rep, typename Period>
00059         static void set_roi_us(std::chrono::duration<Rep, Period> v)
00060         {
00061             double micros = std::chrono::duration<double, std::micro>(v).count();
00062             instance().roi_ = micros;
00063         }
00064         static void set_roi_us(const double v) { instance().roi_ = v; }
00065         static void set_roi_us(const float v) { instance().roi_ = static_cast<double>(v); }
00066
00067         template <typename Rep, typename Period>
00068         static void increment_roi_us(std::chrono::duration<Rep, Period> v)
00069         {
00070             double micros = std::chrono::duration<double, std::micro>(v).count();
00071             instance().roi_ += micros;
00072         }
00073         static void increment_roi_us(const double v) { instance().roi_ += v; }
00074         static void increment_roi_us(const float v) { instance().roi_ += static_cast<double>(v); }
00075
00076         static void set_warmup_us(const time_point_t &start, const time_point_t &end) {
      instance().warmup_ = std::chrono::duration<double, std::micro>(end - start).count(); }
00077         static void set_warmup_us(const float start, const float end) { instance().warmup_ =
      static_cast<double>(end - start); }
00078         static void set_warmup_us(const double start, const double end) { instance().warmup_ = end -
      start; }
00079
00080         template <typename Rep, typename Period>
00081         static void set_warmup_us(std::chrono::duration<Rep, Period> v)
00082         {
00083             double micros = std::chrono::duration<double, std::micro>(v).count();
00084             instance().warmup_ = micros;
00085         }
00086         static void set_warmup_us(const double v) { instance().warmup_ = v; }
00087         static void set_warmup_us(const float v) { instance().warmup_ = static_cast<double>(v); }
00088
00089         static double get_roi_us() { return instance().roi_; }
```

```
00090         static double get_warmup_us() { return instance().warmup_; }
00091
00092         template <std::floating_point T>
00093         static T &fp_tolerance()
00094         {
00095             static T tol = std::numeric_limits<T>::epsilon() * 1e3; // Default tolerance
00096             return tol;
00097         }
00098
00099         template <std::floating_point T>
00100         static void set_fp_tolerance(T v)
00101         {
00102             fp_tolerance<T>() = v;
00103         }
00104
00105         static void set_all_fp_tolerance(long double v)
00106         {
00107             fp_tolerance<float>() = static_cast<float>(v);
00108             fp_tolerance<double>() = static_cast<double>(v);
00109             fp_tolerance<long double>() = v;
00110         }
00111
00112     private:
00113         config() = default;
00114
00115         static config &instance()
00116         {
00117             static config inst;
00118             return inst;
00119         }
00120
00121         double roi_{0.0};
00122         double warmup_{0.0};
00123         uint64_t warmup_iters_{100};
00124         uint64_t bench_iters_{100};
00125     };
00126
00127     /*
00128     Singleton Class for the current state
00129     */
00130     class current_state
00131     {
00132     public:
00133         current_state(const current_state &) = delete;
00134         current_state(current_state &&) = delete;
00135         current_state &operator=(const current_state &) = delete;
00136         current_state &operator=(current_state &&) = delete;
00137
00138         static bool using_plugin() { return instance().using_plugin_; }
00139         static void set_using_plugin(bool v) { instance().using_plugin_ = v; }
00140
00141         static std::string_view impl_name() { return instance().impl_name_; }
00142         static void set_impl_name(std::string_view name) { instance().impl_name_ = name; }
00143
00144     private:
00145         current_state() = default;
00146
00147         static current_state &instance()
00148         {
00149             static current_state inst;
00150             return inst;
00151         }
00152
00153         bool using_plugin_{false};
00154         std::string_view impl_name_;
00155     };
00156
00157 }
```

9.9 include/comppare/internal/helper.hpp File Reference

```
#include <charconv>
#include <string>
#include <string_view>
#include <stdexcept>
#include <cstdint>
#include <comppare/internal/config.hpp>
```

Namespaces

- namespace comppare

  ComPPare framework main namespace.
- namespace comppare::internal
- namespace comppare::internal::helper

Functions

- template<typename T>
  T comppare::internal::helper::get_arg_value (std::string_view option, const char *nextArg)
- static void comppare::internal::helper::parse_args (int argc, char **argv)

9.10 helper.hpp

Go to the documentation of this file.

```
00001 /*
00002
00003 Copyright 2025 | Leong Fan FUNG | funglf | stanleyfunglf@gmail.com
00004
00005 Permission is hereby granted, free of charge, to any person obtaining a copy
00006 of this software and associated documentation files (the "Software"), to deal
00007 in the Software without restriction, including without limitation the rights
00008 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00009 copies of the Software, and to permit persons to whom the Software is
00010 furnished to do so, subject to the following conditions:
00011
00012 The above copyright notice and this permission notice shall be included in
00013 all copies or substantial portions of the Software.
00014
00015 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00016 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00017 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00018 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00019 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00020 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00021 SOFTWARE.
00022
00023 */
00024 #pragma once
00025 #include <charconv>
00026 #include <string>
00027 #include <string_view>
00028 #include <stdexcept>
00029 #include <cstdint>
00030
00031 #include <comppare/internal/config.hpp>
00032
00033 namespace comppare::internal::helper
00034 {
00035     template <typename T>
00036     T get_arg_value(std::string_view option, const char *nextArg)
00037     {
00038         std::string_view valstr;
00039         if (auto eq = option.find('='); eq != std::string_view::npos)
00040             valstr = option.substr(eq + 1);
00041         else if (nextArg)
00042             valstr = nextArg;
00043         else
00044             throw std::invalid_argument(std::string(option) + " requires a value");
00045
00046
00047         if constexpr (std::same_as<T, std::string>)
00048         {
00049             return std::string(valstr);
00050         }
00051         else if constexpr (std::is_integral_v<T>) // if integral type
00052         {
00053             size_t idx = 0;
00054             std::string s{valstr};
00055
00056             if constexpr (std::is_signed_v<T>) // if signed
00057             {
```

```
00058                 long long tmp = std::stoll(s, &idx);
00059                 if (idx != s.size()) // if not all characters were processed
00060                     throw std::invalid_argument("invalid integer for " + std::string(option));
00061
00062                 if (tmp < std::numeric_limits<T>::min() ||
00063                     tmp > std::numeric_limits<T>::max()) // if out of range for T
00064                     throw std::out_of_range("integer out of range for " + std::string(option));
00065
00066                 return static_cast<T>(tmp);
00067             }
00068             else // unsigned
00069             {
00070                 if (!s.empty() && s.front() == '-') // if negative -- reject
00071                     throw std::invalid_argument("invalid unsigned integer for " +
      std::string(option));
00072
00073                 unsigned long long tmp = std::stoull(s, &idx);
00074                 if (idx != s.size()) // if not all characters were processed
00075                     throw std::invalid_argument("invalid unsigned integer for " +
      std::string(option));
00076
00077                 if (tmp > std::numeric_limits<T>::max()) // if out of range for T
00078                     throw std::out_of_range("unsigned integer out of range for " +
      std::string(option));
00079
00080                 return static_cast<T>(tmp);
00081             }
00082         }
00083         else if constexpr (std::is_floating_point_v<T>)
00084         {
00085             size_t idx = 0;
00086             long double tmp = std::stold(std::string(valstr), &idx);
00087             if (idx != valstr.size())
00088                 throw std::invalid_argument("invalid floating-point for " + std::string(option));
00089             return static_cast<T>(tmp);
00090         }
00091         else
00092         {
00093             static_assert(std::is_arithmetic_v<T> || std::same_as<T, std::string>,
00094                           "get_arg_value supports only arithmetic types or std::string");
00095         }
00096     }
00097
00098     static inline void parse_args(int argc, char **argv)
00099     {
00100         if (!argv)
00101             return;
00102
00103         for (int i = 1; i < argc; ++i)
00104         {
00105             std::string_view arg = argv[i];
00106
00107             auto get_next_arg_if_needed = [&](std::string_view a) -> const char *
00108             {
00109                 return (a.find('=') == std::string_view::npos && i + 1 < argc)
00110                            ? argv[++i]
00111                            : nullptr;
00112             };
00113
00114             if (arg.rfind("--warmup", 0) == 0)
00115             {
00116                 auto w = get_arg_value<std::uint64_t>(arg, get_next_arg_if_needed(arg));
00117                 comppare::config::set_warmup_iters(w);
00118             }
00119             else if (arg.rfind("--iter", 0) == 0)
00120             {
00121                 auto n = get_arg_value<std::uint64_t>(arg, get_next_arg_if_needed(arg));
00122                 comppare::config::set_bench_iters(n);
00123             }
00124             else if (arg.rfind("--tolerance", 0) == 0)
00125             {
00126                 auto tol = get_arg_value<long double>(arg, get_next_arg_if_needed(arg));
00127                 comppare::config::set_all_fp_tolerance(tol);
00128             }
00129         }
00130     }
00131 } // namespace comppare::internal::helper
```

9.11 include/comppare/internal/policy.hpp File Reference

```
#include <concepts>
#include <ranges>
```

```
#include <string>
#include <string_view>
#include <type_traits>
#include <utility>
#include <limits>
#include <ostream>
#include <sstream>
#include <variant>
#include <stdexcept>
#include <comppare/internal/concepts.hpp>
#include <comppare/internal/ansi.hpp>
```

Classes

- class comppare::internal::policy::MetricValue< T >
- struct comppare::internal::policy::is_metric_value< typename >
- struct comppare::internal::policy::is_metric_value< MetricValue< U > >
- class comppare::internal::policy::autopolicy::ArithmeticErrorPolicy< T >
- class comppare::internal::policy::autopolicy::StringEqualPolicy
- class comppare::internal::policy::autopolicy::RangeErrorPolicy< R >
- struct comppare::internal::policy::autopolicy::AutoPolicy< T >

Namespaces

- namespace comppare

  ComPPare framework main namespace.
- namespace comppare::internal
- namespace comppare::internal::policy
- namespace comppare::internal::policy::autopolicy

Concepts

- concept comppare::internal::policy::MetricValueSpec
- concept comppare::internal::policy::ErrorPolicy
- concept comppare::internal::policy::autopolicy::SupportedByAutoPolicy

Typedefs

- template<typename T>
  using comppare::internal::policy::autopolicy::AutoPolicy_t = typename AutoPolicy<T>::type

Functions

- template<class EP, class V, class Tol>
  void comppare::internal::policy::compute_error (EP &ep, const V &a, const V &b, Tol tol)
- template<class EP, class V>
  void comppare::internal::policy::compute_error (EP &ep, const V &a, const V &b)
- template<class EP, class Tol>
  bool comppare::internal::policy::is_fail (const EP &ep, Tol tol)
- template<class EP>
  bool comppare::internal::policy::is_fail (const EP &ep)

Variables

- template<typename M>
  constexpr bool comppare::internal::policy::is_metric_value_v = is_metric_value<std::remove_cv_t<M>>↩
  ::value

9.12 policy.hpp

Go to the documentation of this file.

```
00001 /*
00002
00003 Copyright 2025 | Leong Fan FUNG | funglf | stanleyfunglf@gmail.com
00004
00005 Permission is hereby granted, free of charge, to any person obtaining a copy
00006 of this software and associated documentation files (the "Software"), to deal
00007 in the Software without restriction, including without limitation the rights
00008 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00009 copies of the Software, and to permit persons to whom the Software is
00010 furnished to do so, subject to the following conditions:
00011
00012 The above copyright notice and this permission notice shall be included in
00013 all copies or substantial portions of the Software.
00014
00015 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00016 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00017 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00018 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00019 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00020 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00021 SOFTWARE.
00022
00023 */
00024 #pragma once
00025 #include <concepts>
00026 #include <ranges>
00027 #include <string>
00028 #include <string_view>
00029 #include <type_traits>
00030 #include <utility>
00031 #include <limits>
00032 #include <ostream>
00033 #include <sstream>
00034 #include <variant>
00035 #include <stdexcept>
00036
00037 #include <comppare/internal/concepts.hpp>
00038 #include <comppare/internal/ansi.hpp>
00039
00040 namespace comppare::internal::policy
00041 {
00042     /*
00043     MetricValue is a wrapper for a value that can be streamed to an output stream.
00044     It provides an overloaded operator« to stream the value or Error Message if the value is invalid
      or fails.
00045
00046     T value_ is the value of the metric.
00047     bool is_fail_ indicates if the metric has failed.
00048     bool valid_ indicates if the metric is valid. (eg. invalid if size mismatch between 2 vectors)
00049     std::string_view err_msg_ is an error message if the metric is invalid. (eg. outputs "size
      mismatch" if the size of 2 vectors is different)
00050
00051     Note: This could be replaced with std::optional<T, string> in C++23
00052     */
00053     template <comppare::internal::concepts::Streamable T>
00054     class MetricValue
00055     {
00056         T value_;
00057         bool is_fail_{false};
00058
00059         bool valid_{true};
00060         std::string_view err_msg_;
00061
00062     public:
00063         MetricValue(T v) : value_(v), err_msg_(""), valid_(true), is_fail_(false) {}
00064         MetricValue(T v, bool is_fail) : value_(v), is_fail_(is_fail), valid_(true), err_msg_("") {}
00065         MetricValue(T v, bool is_fail, bool valid, std::string_view msg) : value_(v),
      is_fail_(is_fail), valid_(valid), err_msg_(msg) {}
00066
00067         // overloaded operator« to stream the value or error message
```

```
00068         friend std::ostream &
00069         operator«(std::ostream &os, MetricValue<T> const &mv)
00070         {
00071             std::ios saved(nullptr);
00072             saved.copyfmt(os);
00073 
00074             std::ostringstream tmp;
00075             tmp.copyfmt(os);
00076             if (mv.valid_ && !mv.is_fail_)
00077                 tmp « mv.value_;
00078             else if (mv.valid_ && mv.is_fail_)
00079                 tmp « comppare::internal::ansi::RED(mv.value_);
00080             else
00081                 tmp « comppare::internal::ansi::RED(mv.err_msg_);
00082             std::string body = std::move(tmp).str();
00083 
00084             os.width(0);
00085             os « body;
00086 
00087             os.copyfmt(saved);
00088             return os;
00089         }
00090     };
00091     template <typename>
00092     struct is_metric_value : std::false_type
00093     {
00094     };
00095 
00096     template <typename U>
00097     struct is_metric_value<MetricValue<U» : std::true_type
00098     {
00099     };
00100 
00101     template <typename M>
00102     inline constexpr bool is_metric_value_v = is_metric_value<std::remove_cv_t<M>>::value;
00103 
00104     template <typename M>
00105     concept MetricValueSpec = is_metric_value_v<M>;
00106 
00107     /*
00108     Concept for a valid Error Policy
00109 
00110     It requires:
00111     - metric_count() to return the number of metrics
00112     - metric_name(std::size_t i) to return the name of the metric at index i
00113     - compute_error(const Val &a, const Val &b, double tol) to compute the error
00114       between two values a and b with a given tolerance tol -- or not
00115     - metric(std::size_t) to return the value of the metric as MetricValue<T>
00116     - is_fail() to return true if the error exceeds the tolerance
00117     */
00118     template <typename Val, typename EP>
00119     concept ErrorPolicy = requires
00120     // static members
00121     {
00122         { EP::metric_count() } -> std::convertible_to<std::size_t>;
00123         { EP::metric_name(std::size_t{}) } -> std::convertible_to<std::string_view>;
00124     } &&
00125                             // compute error -- either with or without tolerance
00126                             (requires(EP ep, const Val &a, const Val &b, double t) { ep.compute_error(a,
      b, t); } || requires(EP ep, const Val &a, const Val &b) { ep.compute_error(a, b); }) &&
00127                             // metric() returns value of the metric
00128                             (requires(EP ep, std::size_t i) {
00129             { ep.metric(i) } -> MetricValueSpec; } || requires(EP ep, std::size_t i) {
00130             { ep.metric(i) } -> std::convertible_to<double>; } || requires(EP ep, std::size_t i) {
00131             { ep.metric(i) } -> std::same_as<std::string>; }) &&
00132                             // is_fail() -- either with or without tolerance
00133                             (requires(EP ep, double t) {
00134             { ep.is_fail(t) } -> std::convertible_to<bool>; } || requires(EP ep) {
00135             { ep.is_fail() } -> std::convertible_to<bool>; });
00136 
00137     template <class EP, class V, class Tol>
00138     inline void compute_error(EP &ep, const V &a, const V &b, Tol tol)
00139     {
00140         ep.compute_error(a, b, tol);
00141     }
00142 
00143     template <class EP, class V>
00144     inline void compute_error(EP &ep, const V &a, const V &b)
00145     {
00146         ep.compute_error(a, b);
00147     }
00148 
00149     template <class EP, class Tol>
00150     inline bool is_fail(const EP &ep, Tol tol)
00151     {
00152         return ep.is_fail(tol);
00153     }
```

```
00154
00155     template <class EP>
00156     inline bool is_fail(const EP &ep)
00157     {
00158         return ep.is_fail();
00159     }
00160
00161     namespace autopolicy
00162     {
00163         template <typename T>
00164         concept SupportedByAutoPolicy =
00165             comppare::internal::concepts::Arithmetic<T> ||
00166             comppare::internal::concepts::String<T> ||
00167             comppare::internal::concepts::RangeOfArithmetic<T>;
00168
00169         /*
00170         Error Policy for scalar/numbers
00171         */
00172         template <typename T>
00173             requires comppare::internal::concepts::Arithmetic<T>
00174         class ArithmeticErrorPolicy
00175         {
00176             T error_ = T(0);
00177             std::string err_msg_;
00178             bool valid_ = true;
00179
00180             static constexpr std::array names{"Total|err|"};
00181
00182         public:
00183             static constexpr std::size_t metric_count() { return 1; }
00184             static constexpr std::string_view metric_name(std::size_t) { return names[0]; }
00185
00186             MetricValue<T> metric(std::size_t) const
00187             {
00188                 return MetricValue<T>(error_, is_fail(), valid_, err_msg_);
00189             }
00190
00191             bool is_fail() const { return !valid_ || error_ > comppare::config::fp_tolerance<T>(); }
00192
00193             void compute_error(const T &a, const T &b)
00194             {
00195                 if constexpr (!std::is_floating_point_v<T>)
00196                 {
00197                     if (!std::isfinite(a) || !std::isfinite(b))
00198                     {
00199                         error_ = std::numeric_limits<T>::quiet_NaN();
00200                         err_msg_ = "NAN/INF";
00201                         valid_ = false;
00202                         return;
00203                     }
00204                 }
00205
00206                 T e = std::abs(a - b);
00207                 if (e <= comppare::config::fp_tolerance<T>())
00208                     return;
00209                 error_ = e;
00210             }
00211         };
00212
00213         /*
00214         Error Policy for strings
00215         */
00216         class StringEqualPolicy
00217         {
00218             bool eq_{true};
00219
00220             static constexpr std::array names{"Equal?"};
00221
00222         public:
00223             static constexpr std::size_t metric_count() { return 1; }
00224             static constexpr std::string_view metric_name(std::size_t) { return names[0]; }
00225
00226             MetricValue<std::string> metric(std::size_t) const
00227             {
00228                 return MetricValue<std::string>(eq_ ? "true" : "false", is_fail());
00229             }
00230
00231             bool is_fail() const { return !eq_; }
00232
00233             void compute_error(const std::string &a, const std::string &b) { eq_ = (a == b); }
00234         };
00235
00236         /*
00237         Error Policy for ranges of arithmetic types
00238         eg. std::vector<int>, std::deque<float>, etc.
00239         */
00240         template <typename R>
```

```
00241             requires comppare::internal::concepts::RangeOfArithmetic<R>
00242         class RangeErrorPolicy
00243         {
00244             using T = std::remove_cvref_t<std::ranges::range_value_t<R>;
00245 
00246             T max_error_ = T(0);
00247             T total_error_ = T(0);
00248             std::size_t elem_cnt_ = 0;
00249 
00250             bool valid_ = true;
00251             std::string err_msg_;
00252 
00253             static constexpr std::array names{"Max|err|", "Mean|err|", "Total|err|"};
00254 
00255             MetricValue<T> get_max() const
00256             {
00257                 return MetricValue<T>(max_error_, is_fail(), valid_, err_msg_);
00258             }
00259 
00260             MetricValue<T> get_mean() const
00261             {
00262                 if (elem_cnt_ && valid_)
00263                     return MetricValue<T>(total_error_ / static_cast<T>(elem_cnt_), is_fail(), valid_,
    err_msg_);
00264                 else
00265                     return MetricValue<T>(T(0), is_fail(), valid_, err_msg_);
00266             }
00267 
00268             MetricValue<T> get_total() const
00269             {
00270                 return MetricValue<T>(total_error_, is_fail(), valid_, err_msg_);
00271             }
00272 
00273         public:
00274             static constexpr std::size_t metric_count() { return names.size(); }
00275             static constexpr std::string_view metric_name(std::size_t i) { return names[i]; }
00276 
00277             MetricValue<T> metric(std::size_t i) const
00278             {
00279                 switch (i)
00280                 {
00281                 case 0:
00282                     return get_max();
00283                 case 1:
00284                     return get_mean();
00285                 case 2:
00286                     return get_total();
00287                 default:
00288                     throw std::out_of_range("Invalid metric index");
00289                 }
00290             }
00291 
00292             bool is_fail() const
00293             {
00294                 if (!valid_)
00295                     return true;
00296 
00297                 if constexpr (std::is_floating_point_v<T>)
00298                     return max_error_ > comppare::config::fp_tolerance<T>();
00299                 else // integral types
00300                     return max_error_ > T(0);
00301             }
00302 
00303             void compute_error(const R &a, const R &b)
00304             {
00305                 if (std::ranges::size(a) != std::ranges::size(b))
00306                 {
00307                     valid_ = false;
00308                     err_msg_ = "Size mismatch";
00309                     elem_cnt_ = 0;
00310                     return;
00311                 }
00312 
00313                 auto ia = std::ranges::begin(a);
00314                 auto ib = std::ranges::begin(b);
00315                 for (; ia != std::ranges::end(a) && ib != std::ranges::end(b); ++ia, ++ib)
00316                 {
00317                     if constexpr (std::is_floating_point_v<T>)
00318                     {
00319                         if (!std::isfinite(*ia) || !std::isfinite(*ib))
00320                         {
00321                             max_error_ = std::numeric_limits<T>::quiet_NaN();
00322                             total_error_ = std::numeric_limits<T>::quiet_NaN();
00323                             elem_cnt_ = 0;
00324 
00325                             valid_ = false;
00326                             err_msg_ = "NAN/INF";
```

```
00327                             return;
00328                         }
00329                     }
00330
00331                     T diff = std::abs(*ia - *ib);
00332                     if constexpr (std::is_floating_point_v<T>)
00333                     {
00334                         if (diff <= comppare::config::fp_tolerance<T>())
00335                             continue;
00336                     }
00337                     total_error_ += diff;
00338                     max_error_ = std::max(max_error_, diff);
00339                     ++elem_cnt_;
00340                 }
00341             }
00342         };
00343
00344         /*
00345         AutoPolicy is a helper to deduce the appropriate error policy
00346         Currently only supports: scalar types, strings, and ranges of arithmetic types (see above
      concepts)
00347         */
00348         template <typename T>
00349         struct AutoPolicy;
00350
00351         template <typename T>
00352             requires comppare::internal::concepts::Arithmetic<T>
00353         struct AutoPolicy<T>
00354         {
00355             using type = ArithmeticErrorPolicy<std::remove_cvref_t<T>>;
00356         };
00357
00358         template <typename T>
00359             requires comppare::internal::concepts::String<T>
00360         struct AutoPolicy<T>
00361         {
00362             using type = StringEqualPolicy;
00363         };
00364
00365         template <typename T>
00366             requires comppare::internal::concepts::RangeOfArithmetic<T>
00367         struct AutoPolicy<T>
00368         {
00369             using type = RangeErrorPolicy<T>;
00370         };
00371
00372         template <typename T>
00373         using AutoPolicy_t = typename AutoPolicy<T>::type;
00374     }
00375 }
```

9.13 include/comppare/plugin/google_benchmark/google_↩ benchmark.hpp File Reference

9.14 google_benchmark.hpp

Go to the documentation of this file.

```
00001 /*
00002
00003 Copyright 2025 | Leong Fan FUNG | funglf | stanleyfunglf@gmail.com
00004
00005 Permission is hereby granted, free of charge, to any person obtaining a copy
00006 of this software and associated documentation files (the "Software"), to deal
00007 in the Software without restriction, including without limitation the rights
00008 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00009 copies of the Software, and to permit persons to whom the Software is
00010 furnished to do so, subject to the following conditions:
00011
00012 The above copyright notice and this permission notice shall be included in
00013 all copies or substantial portions of the Software.
00014
00015 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00016 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00017 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00018 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00019 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
```

```
00020 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00021 SOFTWARE.
00022
00023 */
00024 #pragma once
00025 #ifdef HAVE_GOOGLE_BENCHMARK
00026 #include <utility>
00027 #include <tuple>
00028 #include <ostream>
00029 #include <iomanip>
00030 #include <iostream>
00031 #include <sstream>
00032 #include <string>
00033 #include <vector>
00034 #include <algorithm>
00035 #include <cstring>
00036 #include <benchmark/benchmark.h>
00037
00038 #include "comppare/plugin/plugin.hpp"
00039 #include "comppare/internal/ansi.hpp"
00040
00041 namespace comppare::plugin::google_benchmark
00042 {
00043     class state
00044     {
00045     public:
00046         state(const state &) = delete;
00047         state &operator=(const state &) = delete;
00048         state(state &&) = delete;
00049         state &operator=(state &&) = delete;
00050
00051         static state &instance()
00052         {
00053             static state inst;
00054             return inst;
00055         }
00056
00057         static void set_state(benchmark::State &st)
00058         {
00059             instance().st_ = &st;
00060         }
00061
00062         static benchmark::State &get_state()
00063         {
00064             if (!instance().st_)
00065                 throw std::runtime_error("Benchmark state is not set.");
00066             return *instance().st_;
00067         }
00068
00069     private:
00070         state() = default;
00071         benchmark::State *st_ = nullptr;
00072     };
00073
00074     // TODO: Merge this into GoogleBenchmarkPlugin
00075     class google_benchmark_manager
00076     {
00077     public:
00078         google_benchmark_manager() = default;
00079         ~google_benchmark_manager() = default;
00080
00081         void initialize(int &argc, char **argv)
00082         {
00083             auto [tmp_argc, tmp_argv] = gbench_parser_.parse(argc, argv);
00084             gbench_argc = tmp_argc;
00085             gbench_argv = tmp_argv;
00086             print_benchmark_header();
00087
00088             benchmark::Initialize(&gbench_argc, gbench_argv);
00089             benchmark::ReportUnrecognizedArguments(gbench_argc, gbench_argv);
00090         }
00091
00092         template <typename Func, typename... Args>
00093         benchmark::internal::Benchmark *add_gbench(const char *name, Func f, Args &&...args)
00094         {
00095             std::tuple<Args...> cargs(std::forward<Args>(args)...);
00096
00097             auto benchptr = benchmark::RegisterBenchmark(
00098                 name,
00099                 [f, cargs = std::move(cargs)](benchmark::State &st) mutable
00100                 {
00101                     comppare::plugin::google_benchmark::state::set_state(st);
00102                     std::apply([&](auto &&...unpacked)
00103                                { f(std::forward<decltype(unpacked)>(unpacked)...); }, cargs);
00104                     benchmark::ClobberMemory();
00105                 });
00106
```

```
            return benchptr;
        }

        void run()
        {
            benchmark::RunSpecifiedBenchmarks();
            benchmark::Shutdown();
        }

    private:
        int gbench_argc;
        char** gbench_argv;
        comppare::plugin::PluginArgParser gbench_parser_{"--gbench"};

        void print_benchmark_header()
        {

            std::cout « "\n"
                      « std::left « comppare::internal::ansi::BOLD
                      « "*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=\n============= "
                      « comppare::internal::ansi::ITALIC("Google Benchmark")
                      « " =============\n=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*"
                      « comppare::internal::ansi::BOLD_OFF « "\n\n";

            std::cout « "Google Benchmark cmdline arguments:\n";
            for (int i = 0; i < gbench_argc; ++i)
            {
                std::cout « std::setw(2) « std::right « " "
                          « "  [" « i « "] " « std::quoted(gbench_argv[i]) « "\n";
            }

            std::cout « std::left
                      « comppare::internal::ansi::BOLD("=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*")
                      « "\n\n";
        }
    };

    template <class InTup, class OutTup>
    class GoogleBenchmarkPlugin final : public Plugin<InTup, OutTup>
    {
        using Self = GoogleBenchmarkPlugin<InTup, OutTup>;
        comppare::plugin::google_benchmark::google_benchmark_manager gb_;

    public:
        GoogleBenchmarkPlugin(const GoogleBenchmarkPlugin &) = delete;
        GoogleBenchmarkPlugin &operator=(const GoogleBenchmarkPlugin &) = delete;

        static std::shared_ptr<Self> instance()
        {
            static std::shared_ptr<Self> inst{new Self};
            return inst;
        }

        template <class Func>
        benchmark::internal::Benchmark *register_impl(const std::string &name,
                                                       Func &&user_fn,
                                                       const InTup &inputs,
                                                       OutTup &outs)
        {
            return std::apply([&](auto const &...in_vals)
                              { return std::apply([&](auto &&...outs_vals)
                                                  { return gb_.add_gbench(name.c_str(),
                                                                           std::forward<Func>(user_fn),
                                                                           in_vals..., outs_vals...);
}, outs); }, inputs);
        }

        void initialize(int &argc, char **argv) override
        {
            gb_.initialize(argc, argv);
        }
        void run() override
        {
            gb_.run();
        }

    private:
        GoogleBenchmarkPlugin() = default;
    };

    template <comppare::internal::concepts::FloatingPoint T>
    inline void SetIterationTime(T time)
    {
        benchmark::State &st = comppare::plugin::google_benchmark::state::get_state();
        st.SetIterationTime(static_cast<double>(time * 1e-6));
    }

```

```
00193     template <typename Rep, typename Period>
00194     inline void SetIterationTime(std::chrono::duration<Rep, Period> time)
00195     {
00196         benchmark::State &st = comppare::plugin::google_benchmark::state::get_state();
00197         double elapsed_seconds =
      std::chrono::duration_cast<std::chrono::duration<double»(time).count();
00198         st.SetIterationTime(elapsed_seconds);
00199     }
00200
00201 }
00202
00203 #define PLUGIN_HOTLOOP_BENCH                                                    \
00204     benchmark::State &st = comppare::plugin::google_benchmark::state::get_state(); \
00205     for (auto _ : st)                                                           \
00206     {                                                                           \
00207         hotloop_body();                                                         \
00208     }
00209
00210 #define PLUGIN_SET_ITERATION_TIME(TIME) \
00211     comppare::plugin::google_benchmark::SetIterationTime(TIME);
00212
00213 #endif // HAVE_GOOGLE_BENCHMARK
```

9.15 include/comppare/plugin/nvbench/nvbench.hpp File Reference

9.16 nvbench.hpp

Go to the documentation of this file.

```
00001 /*
00002
00003 Copyright 2025 | Leong Fan FUNG | funglf | stanleyfunglf@gmail.com
00004
00005 Permission is hereby granted, free of charge, to any person obtaining a copy
00006 of this software and associated documentation files (the "Software"), to deal
00007 in the Software without restriction, including without limitation the rights
00008 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00009 copies of the Software, and to permit persons to whom the Software is
00010 furnished to do so, subject to the following conditions:
00011
00012 The above copyright notice and this permission notice shall be included in
00013 all copies or substantial portions of the Software.
00014
00015 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00016 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00017 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00018 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00019 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00020 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00021 SOFTWARE.
00022
00023 */
00024 #pragma once
00025 #ifdef HAVE_NV_BENCH
00026 #include <nvbench/nvbench.cuh>
00027 #include "comppare/plugin/plugin.hpp"
00028
00029 #include <memory>
00030 #include <functional>
00031 #include <sstream>
00032
00033 namespace comppare::plugin::nvbenchplugin
00034 {
00035     template <typename F>
00036     struct NvbenchCallable
00037     {
00038         F f;
00039
00040         void operator()(nvbench::state &st, nvbench::type_list<>)
00041         {
00042             f(st);
00043         }
00044
00045         NvbenchCallable(F _f) : f(std::move(_f)) {}
00046         ~NvbenchCallable() = default;
00047
00048         NvbenchCallable(const NvbenchCallable &other) = default;
00049         NvbenchCallable &operator=(const NvbenchCallable &other) = default;
00050         NvbenchCallable(NvbenchCallable &&other) = default;
```

```
00051         NvbenchCallable &operator=(NvbenchCallable &&other) = default;
00052     };
00053
00054     class state
00055     {
00056     public:
00057         state(const state &) = delete;
00058         state &operator=(const state &) = delete;
00059         state(state &&) = delete;
00060         state &operator=(state &&) = delete;
00061
00062         static state &instance()
00063         {
00064             static state inst;
00065             return inst;
00066         }
00067
00068         static void set_state(nvbench::state *st)
00069         {
00070             instance().st_ = st;
00071         }
00072
00073         static nvbench::state *get_state()
00074         {
00075             if (!instance().st_)
00076                 throw std::runtime_error("Benchmark state is not set.");
00077             return instance().st_;
00078         }
00079
00080     private:
00081         state() = default;
00082         nvbench::state *st_ = nullptr;
00083     };
00084
00085     class nvbench_manager
00086     {
00087     public:
00088         nvbench_manager() = default;
00089         ~nvbench_manager() = default;
00090
00091         void initialize(int &argc, char **argv)
00092         {
00093             auto [tmp_argc, tmp_argv] = nvbench_parser_.parse(argc, argv);
00094             nvbench_argc = tmp_argc;
00095             nvbench_argv = tmp_argv;
00096             print_benchmark_header();
00097         }
00098
00099         template <typename Func, typename... Args>
00100         nvbench::benchmark_base &add_nvbench(const char *name, Func f, Args &&...args)
00101         {
00102             std::tuple<std::decay_t<Args>...> cargs(std::forward<Args>(args)...);
00103
00104             auto nvbench_wrapper = [f, cargs = std::move(cargs)](nvbench::state &st) mutable
00105             {
00106                 comppare::plugin::nvbenchplugin::state::set_state(&st);
00107                 std::apply([&](auto &&...unpacked)
00108                            { f(std::forward<decltype(unpacked)>(unpacked)...); }, cargs);
00109             };
00110
00111             using Callable = NvbenchCallable<decltype(nvbench_wrapper)>;
00112
00113             return nvbench::benchmark_manager::get()
00114
.add(std::make_unique<nvbench::benchmark<Callable>>(Callable{std::move(nvbench_wrapper)}))
00115                 .set_name(name);
00116         }
00117
00118         void run()
00119         {
00120             NVBENCH_MAIN_INITIALIZE(nvbench_argc, nvbench_argv);
00121             {
00122                 NVBENCH_MAIN_PARSE(nvbench_argc, nvbench_argv);
00123
00124                 NVBENCH_MAIN_PRINT_PREAMBLE(parser);
00125                 NVBENCH_MAIN_RUN_BENCHMARKS(parser);
00126                 NVBENCH_MAIN_PRINT_EPILOGUE(parser);
00127
00128                 NVBENCH_MAIN_PRINT_RESULTS(parser);
00129             } /* Tear down parser before finalization */
00130             NVBENCH_MAIN_FINALIZE();
00131         }
00132
00133     private:
00134         int nvbench_argc;
00135         char** nvbench_argv;
00136         comppare::plugin::PluginArgParser nvbench_parser_{"--nvbench"};
```

```
        void print_benchmark_header()
        {

            std::cout « "\n"
                      « std::left « comppare::internal::ansi::BOLD
                      « "*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*\n============= "
                      « comppare::internal::ansi::ITALIC("nvbench")
                      « " =============\n*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*"
                      « comppare::internal::ansi::BOLD_OFF « "\n\n";

            std::cout « "nvbench cmdline arguments:\n";
            for (int i = 0; i < nvbench_argc; ++i)
            {
                std::cout « std::setw(2) « std::right « " "
                          « "  [" « i « "] " « std::quoted(nvbench_argv[i]) « "\n";
            }

            std::cout « std::left
                      « comppare::internal::ansi::BOLD("*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*")
                      « "\n\n";
        }
    };

    template <class InTup, class OutTup>
    class nvbenchPlugin final : public Plugin<InTup, OutTup>
    {
        using Self = nvbenchPlugin<InTup, OutTup>;

    private:
        nvbenchPlugin() = default;
        comppare::plugin::nvbenchplugin::nvbench_manager nb_;

    public:
        nvbenchPlugin(const nvbenchPlugin &) = delete;
        nvbenchPlugin &operator=(const nvbenchPlugin &) = delete;

        static std::shared_ptr<Self> instance()
        {
            static std::shared_ptr<Self> inst{new Self};
            return inst;
        }

        template <class Func>
        nvbench::benchmark_base &register_impl(const std::string &name,
                                               Func &&user_fn,
                                               const InTup &inputs,
                                               OutTup &outs)
        {
            return std::apply([&](auto&&... in_vals) -> nvbench::benchmark_base& {
                return std::apply([&](auto&&... out_vals) -> nvbench::benchmark_base& {
                    return nb_.add_nvbench(name.c_str(),
                                          std::forward<Func>(user_fn),
                                          std::forward<decltype(in_vals)>(in_vals)...,
                                          std::forward<decltype(out_vals)>(out_vals)...);
                }, outs);
            }, inputs);
        }

        void initialize(int &argc, char **argv) override
        {
            nb_.initialize(argc, argv);
        }
        void run() override
        {
            nb_.run();
        }
    };
}

#define PLUGIN_HOTLOOP_BENCH          \
    auto state_ = comppare::plugin::nvbenchplugin::state::get_state(); \
    state_->exec([&](nvbench::launch &launch) { hotloop_body(); });

#endif
```

9.17 include/comppare/plugin/plugin.hpp File Reference

```
#include <concepts>
#include <utility>
```

```
#include <iomanip>
#include <stdexcept>
#include <sstream>
#include <string>
#include <vector>
```

Classes

- class comppare::plugin::Plugin< InTup, OutTup >
- class comppare::plugin::PluginArgParser

Namespaces

- namespace comppare

  ComPPare framework main namespace.
- namespace comppare::plugin

Concepts

- concept comppare::plugin::ValidPlugin

9.18 plugin.hpp

Go to the documentation of this file.

```
00001 /*
00002
00003 Copyright 2025 | Leong Fan FUNG | funglf | stanleyfunglf@gmail.com
00004
00005 Permission is hereby granted, free of charge, to any person obtaining a copy
00006 of this software and associated documentation files (the "Software"), to deal
00007 in the Software without restriction, including without limitation the rights
00008 to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
00009 copies of the Software, and to permit persons to whom the Software is
00010 furnished to do so, subject to the following conditions:
00011
00012 The above copyright notice and this permission notice shall be included in
00013 all copies or substantial portions of the Software.
00014
00015 THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
00016 IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
00017 FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
00018 AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
00019 LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
00020 OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
00021 SOFTWARE.
00022
00023 */
00024 #pragma once
00025 #include <concepts>
00026 #include <utility>
00027 #include <iomanip>
00028 #include <stdexcept>
00029 #include <sstream>
00030 #include <string>
00031 #include <vector>
00032
00033
00034 namespace comppare::plugin
00035 {
00036     template <class InTup, class OutTup>
00037     class Plugin
00038     {
00039     public:
00040         virtual ~Plugin() = default;
```

```
00041         virtual void initialize(int & /*argc*/, char ** /*argv*/) {}
00042         virtual void run() {}
00043     };
00044
00045     template <template <class,class> class P, class InTup, class OutTup, class Func>
00046     concept ValidPlugin =
00047         requires { { P<InTup, OutTup>::instance() } -> std::same_as<std::shared_ptr<P<InTup, OutTup>>>;
    }
00048         &&
00049         requires(const std::string& name, Func&& user_fn, const InTup& inputs, OutTup& outputs)
00050         { std::declval<P<InTup, OutTup>&>().register_impl(name, user_fn, inputs, outputs); }
00051         &&
00052         std::derived_from<P<InTup, OutTup>, plugin::Plugin<InTup, OutTup>>;
00053
00054
00055     class PluginArgParser {
00056     public:
00057         explicit PluginArgParser(std::string header, bool strict_missing_value = false)
00058             : header_(std::move(header)), strict_(strict_missing_value) {}
00059
00060         PluginArgParser(const PluginArgParser&) = delete;
00061         PluginArgParser& operator=(const PluginArgParser&) = delete;
00062         PluginArgParser(PluginArgParser&&) = default;
00063         PluginArgParser& operator=(PluginArgParser&&) = default;
00064         ~PluginArgParser() = default;
00065
00066         [[nodiscard]] std::pair<int, char**> parse(int argc, char** argv) {
00067             args_.clear();
00068             cargv_.clear();
00069
00070             if (argc <= 0 || argv == nullptr || argv[0] == nullptr) {
00071                 args_.push_back(header_.empty() ? "program" : header_);
00072             } else {
00073                 args_.emplace_back(argv[0]);
00074             }
00075
00076             const std::string eq_prefix = header_ + "=";
00077             std::vector<std::string> tokens;
00078
00079             for (int i = 1; i < argc; ++i) {
00080                 if (!argv || argv[i] == nullptr) break;
00081                 const std::string cur(argv[i]);
00082
00083                 if (starts_with(cur, eq_prefix)) {
00084                     const std::string value = cur.substr(eq_prefix.size());
00085                     append_tokens(tokens, value);
00086                 } else if (cur == header_) {
00087                     if (i + 1 < argc && argv[i + 1] != nullptr) {
00088                         const std::string value(argv[++i]); // consume value
00089                         append_tokens(tokens, value);
00090                     } else if (strict_) {
00091                         throw std::invalid_argument(header_ + " requires a value");
00092                     }
00093                 }
00094             }
00095
00096             for (const auto& t : tokens) args_.push_back(t);
00097
00098             cargv_.reserve(args_.size() + 1);
00099             for (const auto& s : args_) cargv_.push_back(const_cast<char*>(s.c_str()));
00100             cargv_.push_back(nullptr);
00101
00102             return { static_cast<int>(args_.size()), cargv_.data() };
00103         }
00104
00105         [[nodiscard]] int argc() const { return static_cast<int>(args_.size()); }
00106         [[nodiscard]] char** argv()    { return cargv_.data(); } // valid after parse()
00107         [[nodiscard]] const std::vector<std::string>& args() const { return args_; }
00108
00109     private:
00110         std::string header_;
00111         bool strict_;
00112         std::vector<std::string> args_;
00113         std::vector<char*> cargv_;
00114
00115         static bool starts_with(const std::string& s, const std::string& prefix) {
00116             return s.size() >= prefix.size() && s.compare(0, prefix.size(), prefix) == 0;
00117         }
00118
00119         // shell-like split that respects quotes: R"(--x "a b" c)" -> {"--x","a b","c"}
00120         static std::vector<std::string> split_shell_like(const std::string& s) {
00121             std::vector<std::string> out;
00122             std::istringstream iss(s);
00123             std::string tok;
00124             while (iss >> std::quoted(tok)) out.push_back(tok);
00125             return out;
00126         }
```

```
00127
00128         static void append_tokens(std::vector<std::string>& dst, const std::string& value) {
00129             auto toks = split_shell_like(value);
00130             dst.insert(dst.end(), toks.begin(), toks.end());
00131         }
00132     };
00133
00134 } // namespace comppare::plugin
```